

My Project

Generated by Doxygen 1.10.0

1 MSDScript	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	7
3.1 Class List	7
4 File Index	11
4.1 File List	11
5 Class Documentation	13
5.1 Add Class Reference	13
5.1.1 Constructor & Destructor Documentation	13
5.1.1.1 Add()	13
5.1.2 Member Function Documentation	14
5.1.2.1 equals()	14
5.1.2.2 has_variable()	14
5.1.2.3 interp()	14
5.1.2.4 pretty_print()	14
5.1.2.5 pretty_print_dr()	14
5.1.2.6 print()	14
5.1.2.7 subst()	15
5.2 Catch::always_false< T > Struct Template Reference	15
5.3 Catch::Detail::Approx Class Reference	15
5.4 Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch > Struct Template Reference	16
5.4.1 Member Function Documentation	17
5.4.1.1 describe()	17
5.5 Catch::Generators::as< T > Struct Template Reference	17
5.6 Catch::AssertionHandler Class Reference	18
5.7 Catch::AssertionInfo Struct Reference	18
5.8 Catch::AssertionReaction Struct Reference	18
5.9 Catch::AutoReg Struct Reference	19
5.10 Catch::BinaryExpr< LhsT, RhsT > Class Template Reference	19
5.11 Catch::Capturer Class Reference	20
5.12 Catch::Matchers::StdString::CasedString Struct Reference	20
5.13 Catch::CaseSensitive Struct Reference	21
5.14 Catch_global_namespace_dummy Struct Reference	21
5.15 Catch::Generators::ChunkGenerator< T > Class Template Reference	21
5.15.1 Member Function Documentation	22
5.15.1.1 get()	22
5.15.1.2 next()	22
5.16 Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc > Struct Template Reference	22
5.16.1 Member Function Documentation	23

5.16.1.1 describe()	23
5.17 Catch::Matchers::StdString::ContainsMatcher Struct Reference	23
5.18 Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch > Struct Template Reference	25
5.18.1 Member Function Documentation	26
5.18.1.1 describe()	26
5.19 Catch::Counts Struct Reference	26
5.20 Catch::Decomposer Struct Reference	26
5.21 Catch::Matchers::StdString::EndsWithMatcher Struct Reference	27
5.22 Catch::Detail::EnumInfo Struct Reference	28
5.23 Catch::Matchers::StdString::EqualsMatcher Struct Reference	28
5.24 Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch > Struct Template Reference	29
5.24.1 Member Function Documentation	30
5.24.1.1 describe()	30
5.25 Catch::Matchers::Exception::ExceptionMessageMatcher Class Reference	30
5.25.1 Member Function Documentation	31
5.25.1.1 describe()	31
5.26 Catch::ExceptionTranslatorRegistrar Class Reference	31
5.27 Expr Class Reference	32
5.28 Catch::ExprLhs< LhsT > Class Template Reference	32
5.29 Catch::Generators::FilterGenerator< T, Predicate > Class Template Reference	33
5.29.1 Member Function Documentation	33
5.29.1.1 get()	33
5.29.1.2 next()	34
5.30 Catch::Generators::FixedValuesGenerator< T > Class Template Reference	34
5.30.1 Member Function Documentation	34
5.30.1.1 get()	34
5.30.1.2 next()	35
5.31 Catch::GeneratorException Class Reference	35
5.32 Catch::Generators::Generators< T > Class Template Reference	35
5.32.1 Member Function Documentation	36
5.32.1.1 get()	36
5.32.1.2 next()	36
5.33 Catch::Generators::GeneratorUntypedBase Class Reference	36
5.34 Catch::Generators::GeneratorWrapper< T > Class Template Reference	37
5.35 Catch::IConfig Struct Reference	37
5.36 Catch::IContext Struct Reference	38
5.37 Catch::IExceptionTranslator Struct Reference	38
5.38 Catch::IExceptionTranslatorRegistry Struct Reference	38
5.39 Catch::Generators::IGenerator< T > Struct Template Reference	39
5.40 Catch::IGeneratorTracker Struct Reference	39
5.41 Catch::IMutableContext Struct Reference	40
5.42 Catch::IMutableEnumValuesRegistry Struct Reference	40

5.43 Catch::IMutableRegistryHub Struct Reference	41
5.44 Catch::IRegistryHub Struct Reference	41
5.45 Catch::IResultCapture Struct Reference	41
5.46 Catch::IRunner Struct Reference	42
5.47 Catch::is_callable< T > Struct Template Reference	42
5.48 Catch::is_callable< Fun(Args...) > Struct Template Reference	42
5.49 Catch::is_callable_tester Struct Reference	43
5.50 Catch::is_range< T > Struct Template Reference	43
5.51 Catch::detail::is_range_impl< T, typename > Struct Template Reference	43
5.52 Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >())) >::type > Struct Template Reference	44
5.53 Catch::Detail::IsStreamInsertable< T > Class Template Reference	44
5.54 Catch::IStream Struct Reference	44
5.55 Catch::Generators::IteratorGenerator< T > Class Template Reference	44
5.55.1 Member Function Documentation	45
5.55.1.1 get()	45
5.55.1.2 next()	45
5.56 Catch::ITestCaseRegistry Struct Reference	45
5.57 Catch::ITestInvoker Struct Reference	46
5.58 Catch::ITransientExpression Struct Reference	46
5.59 Catch::LazyExpression Class Reference	47
5.60 Catch::Generators::MapGenerator< T, U, Func > Class Template Reference	47
5.60.1 Member Function Documentation	48
5.60.1.1 get()	48
5.60.1.2 next()	48
5.61 Catch::Matchers::Impl::MatchAllOf< ArgT > Struct Template Reference	48
5.61.1 Member Function Documentation	49
5.61.1.1 describe()	49
5.62 Catch::Matchers::Impl::MatchAnyOf< ArgT > Struct Template Reference	49
5.62.1 Member Function Documentation	50
5.62.1.1 describe()	50
5.63 Catch::Matchers::Impl::MatcherBase< T > Struct Template Reference	51
5.64 Catch::Matchers::Impl::MatcherMethod< ObjectT > Struct Template Reference	52
5.65 Catch::Matchers::Impl::MatcherUntypedBase Class Reference	52
5.66 Catch::MatchExpr< ArgT, MatcherT > Class Template Reference	53
5.66.1 Member Function Documentation	53
5.66.1.1 streamReconstructedExpression()	53
5.67 Catch::Matchers::Impl::MatchNotOf< ArgT > Struct Template Reference	54
5.67.1 Member Function Documentation	55
5.67.1.1 describe()	55
5.68 Catch::MessageBuilder Struct Reference	55
5.69 Catch::MessageInfo Struct Reference	56

5.70 Catch::MessageStream Struct Reference	56
5.71 Mult Class Reference	57
5.71.1 Constructor & Destructor Documentation	57
5.71.1.1 Mult()	57
5.71.2 Member Function Documentation	58
5.71.2.1 equals()	58
5.71.2.2 has_variable()	58
5.71.2.3 interp()	58
5.71.2.4 pretty_print()	58
5.71.2.5 pretty_print_dr()	58
5.71.2.6 print()	58
5.71.2.7 subst()	59
5.72 Catch::NameAndTags Struct Reference	59
5.73 Catch::NonCopyable Class Reference	59
5.74 Num Class Reference	60
5.74.1 Constructor & Destructor Documentation	60
5.74.1.1 Num()	60
5.74.2 Member Function Documentation	61
5.74.2.1 equals()	61
5.74.2.2 has_variable()	61
5.74.2.3 interp()	61
5.74.2.4 pretty_print()	61
5.74.2.5 pretty_print_dr()	61
5.74.2.6 print()	61
5.74.2.7 subst()	62
5.75 Catch::Option< T > Class Template Reference	62
5.76 Catch::pluralise Struct Reference	62
5.77 Catch::Matchers::Generic::PredicateMatcher< T > Class Template Reference	63
5.77.1 Member Function Documentation	64
5.77.1.1 describe()	64
5.77.1.2 match()	64
5.78 Catch::Generators::RandomFloatingGenerator< Float > Class Template Reference	64
5.78.1 Member Function Documentation	65
5.78.1.1 get()	65
5.78.1.2 next()	65
5.79 Catch::Generators::RandomIntegerGenerator< Integer > Class Template Reference	65
5.79.1 Member Function Documentation	66
5.79.1.1 get()	66
5.79.1.2 next()	66
5.80 Catch::Generators::RangeGenerator< T > Class Template Reference	66
5.80.1 Member Function Documentation	67
5.80.1.1 get()	67

5.80.1.2 next()	67
5.81 Catch::Matchers::StdString::RegexMatcher Struct Reference	67
5.81.1 Member Function Documentation	68
5.81.1.1 describe()	68
5.82 Catch::RegistrarForTagAliases Struct Reference	68
5.83 Catch::Generators::RepeatGenerator< T > Class Template Reference	68
5.83.1 Member Function Documentation	69
5.83.1.1 get()	69
5.83.1.2 next()	69
5.84 Catch::ResultDisposition Struct Reference	69
5.85 Catch::ResultWas Struct Reference	70
5.86 Catch::ReusableStringStream Class Reference	70
5.87 Catch::RunTests Struct Reference	70
5.88 Catch::ScopedMessage Class Reference	71
5.89 Catch::Section Class Reference	71
5.90 Catch::SectionEndInfo Struct Reference	71
5.91 Catch::SectionInfo Struct Reference	72
5.92 Catch::ShowDurations Struct Reference	72
5.93 Catch::SimplePcg32 Class Reference	72
5.94 Catch::Generators::SingleValueGenerator< T > Class Template Reference	73
5.94.1 Member Function Documentation	73
5.94.1.1 get()	73
5.94.1.2 next()	74
5.95 Catch::SourceLineInfo Struct Reference	74
5.96 Catch::Matchers::StdString::StartsWithMatcher Struct Reference	74
5.97 Catch::StreamEndStop Struct Reference	76
5.98 Catch::StringMaker< T, typename > Struct Template Reference	76
5.99 Catch::StringMaker< bool > Struct Reference	76
5.100 Catch::StringMaker< Catch::Detail::Approx > Struct Reference	76
5.101 Catch::StringMaker< char * > Struct Reference	77
5.102 Catch::StringMaker< char > Struct Reference	77
5.103 Catch::StringMaker< char const * > Struct Reference	77
5.104 Catch::StringMaker< char[SZ]> Struct Template Reference	77
5.105 Catch::StringMaker< double > Struct Reference	78
5.106 Catch::StringMaker< float > Struct Reference	78
5.107 Catch::StringMaker< int > Struct Reference	78
5.108 Catch::StringMaker< long > Struct Reference	78
5.109 Catch::StringMaker< long long > Struct Reference	79
5.110 Catch::StringMaker< R C::* > Struct Template Reference	79
5.111 Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStreamInsertable< R >::value >::type > Struct Template Reference	79
5.112 Catch::StringMaker< signed char > Struct Reference	79

5.113 Catch::StringMaker< signed char[SZ]> Struct Template Reference	80
5.114 Catch::StringMaker< std::nullptr_t > Struct Reference	80
5.115 Catch::StringMaker< std::string > Struct Reference	80
5.116 Catch::StringMaker< std::wstring > Struct Reference	80
5.117 Catch::StringMaker< T * > Struct Template Reference	81
5.118 Catch::StringMaker< T[SZ]> Struct Template Reference	81
5.119 Catch::StringMaker< unsigned char > Struct Reference	81
5.120 Catch::StringMaker< unsigned char[SZ]> Struct Template Reference	81
5.121 Catch::StringMaker< unsigned int > Struct Reference	82
5.122 Catch::StringMaker< unsigned long > Struct Reference	82
5.123 Catch::StringMaker< unsigned long long > Struct Reference	82
5.124 Catch::StringMaker< wchar_t * > Struct Reference	82
5.125 Catch::StringMaker< wchar_t const * > Struct Reference	83
5.126 Catch::Matchers::StdString::StringMatcherBase Struct Reference	83
5.126.1 Member Function Documentation	84
5.126.1.1 describe()	84
5.127 Catch::StringRef Class Reference	84
5.127.1 Detailed Description	85
5.128 Catch::Generators::TakeGenerator< T > Class Template Reference	85
5.128.1 Member Function Documentation	85
5.128.1.1 get()	85
5.128.1.2 next()	86
5.129 Catch::TestCase Class Reference	86
5.130 Catch::TestCaseInfo Struct Reference	87
5.131 Catch::TestFailureException Struct Reference	88
5.132 Catch::TestInvokerAsMethod< C > Class Template Reference	88
5.132.1 Member Function Documentation	88
5.132.1.1 invoke()	88
5.133 Catch::Timer Class Reference	89
5.134 Catch::Totals Struct Reference	89
5.135 Catch::true_given< typename > Struct Template Reference	89
5.136 Catch::UnaryExpr< LhsT > Class Template Reference	90
5.137 Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch > Struct Template Reference	90
5.137.1 Member Function Documentation	91
5.137.1.1 describe()	91
5.138 Catch::UseColour Struct Reference	91
5.139 Var Class Reference	92
5.139.1 Constructor & Destructor Documentation	92
5.139.1.1 Var()	92
5.139.2 Member Function Documentation	93
5.139.2.1 equals()	93

5.139.2.2 <code>has_variable()</code>	93
5.139.2.3 <code>interp()</code>	93
5.139.2.4 <code>pretty_print()</code>	93
5.139.2.5 <code>pretty_print_dr()</code>	93
5.139.2.6 <code>print()</code>	93
5.139.2.7 <code>subst()</code>	94
5.140 <code>Catch::detail::void_type<... ></code> Struct Template Reference	94
5.141 <code>Catch::WaitForKeypress</code> Struct Reference	94
5.142 <code>Catch::WarnAbout</code> Struct Reference	94
5.143 <code>Catch::Matchers::Floating::WithinAbsMatcher</code> Struct Reference	95
5.143.1 Member Function Documentation	95
5.143.1.1 <code>describe()</code>	95
5.144 <code>Catch::Matchers::Floating::WithinRelMatcher</code> Struct Reference	96
5.144.1 Member Function Documentation	96
5.144.1.1 <code>describe()</code>	96
5.145 <code>Catch::Matchers::Floating::WithinUlpsMatcher</code> Struct Reference	97
5.145.1 Member Function Documentation	97
5.145.1.1 <code>describe()</code>	97
6 File Documentation	99
6.1 <code>/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp</code>	99
6.2 <code>/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/cmdline.h</code>	313
6.3 <code>/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/Expr.h</code>	314
Index	317

Chapter 1

MSDScript

Author

Tailang Cao

Second Author (if applicable)

Date

06-02-2024

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Catch::Detail::Approx	15
Catch::Generators::as< T >	17
Catch::AssertionHandler	18
Catch::AssertionInfo	18
Catch::AssertionReaction	18
Catch::Capturer	20
Catch::Matchers::StdString::CasedString	20
Catch::CaseSensitive	21
Catch_global_namespace_dummy	21
Catch::Counts	26
Catch::Decomposer	26
Catch::Detail::EnumInfo	28
std::exception	
Catch::GeneratorException	35
Catch::ExceptionTranslatorRegistrar	31
Expr	32
Add	13
Mult	57
Num	60
Var	92
Catch::ExprLhs< LhsT >	32
std::false_type	
Catch::always_false< T >	15
Catch::detail::is_range_impl< T, typename >	43
Catch::is_range< T >	43
Catch::Generators::GeneratorUntypedBase	36
Catch::Generators::IGenerator< std::vector< T > >	39
Catch::Generators::ChunkGenerator< T >	21
Catch::Generators::IGenerator< Float >	39
Catch::Generators::RandomFloatingGenerator< Float >	64
Catch::Generators::IGenerator< Integer >	39
Catch::Generators::RandomIntegerGenerator< Integer >	65
Catch::Generators::IGenerator< T >	39
Catch::Generators::FilterGenerator< T, Predicate >	33

Catch::Generators::FixedValuesGenerator< T >	34
Catch::Generators::Generators< T >	35
Catch::Generators::IteratorGenerator< T >	44
Catch::Generators::MapGenerator< T, U, Func >	47
Catch::Generators::RangeGenerator< T >	66
Catch::Generators::RepeatGenerator< T >	68
Catch::Generators::SingleValueGenerator< T >	73
Catch::Generators::TakeGenerator< T >	85
Catch::Generators::GeneratorWrapper< T >	37
Catch::Generators::GeneratorWrapper< U >	37
Catch::IContext	38
Catch::IMutableContext	40
Catch::IExceptionTranslator	38
Catch::IExceptionTranslatorRegistry	38
Catch::IGeneratorTracker	39
Catch::IMutableEnumValuesRegistry	40
Catch::IMutableRegistryHub	41
Catch::IRegistryHub	41
Catch::IResultCapture	41
Catch::IRunner	42
Catch::is_callable< T >	42
Catch::is_callable< Fun(Args...)>	42
Catch::is_callable_tester	43
Catch::Detail::IsStreamInsertable< T >	44
Catch::IStream	44
Catch::ITestCaseRegistry	45
Catch::ITestInvoker	46
Catch::TestInvokerAsMethod< C >	88
Catch::ITransientExpression	46
Catch::BinaryExpr< LhsT, RhsT >	19
Catch::MatchExpr< ArgT, MatcherT >	53
Catch::UnaryExpr< LhsT >	90
Catch::LazyExpression	47
Catch::Matchers::Impl::MatcherMethod< ObjectT >	52
Catch::Matchers::Impl::MatcherBase< std::exception >	51
Catch::Matchers::Impl::MatcherBase< double >	51
Catch::Matchers::Impl::MatcherBase< ArgT >	51
Catch::Matchers::Impl::MatchAllOf< ArgT >	48
Catch::Matchers::Impl::MatchAnyOf< ArgT >	49
Catch::Matchers::Impl::MatchNotOf< ArgT >	54
Catch::Matchers::Impl::MatcherBase< std::string >	51
Catch::Matchers::Impl::MatcherMethod< ArgT >	52
Catch::Matchers::Impl::MatcherMethod< double >	52
Catch::Matchers::Impl::MatcherMethod< std::exception >	52
Catch::Matchers::Impl::MatcherMethod< std::string >	52
Catch::Matchers::Impl::MatcherMethod< T >	52
Catch::Matchers::Impl::MatcherBase< std::vector< T, AllocMatch > >	51
Catch::Matchers::Impl::MatcherBase< std::vector< T, Alloc > >	51
Catch::Matchers::Impl::MatcherBase< T >	51
Catch::Matchers::Exception::ExceptionMessageMatcher	30
Catch::Matchers::Floating::WithinAbsMatcher	95
Catch::Matchers::Floating::WithinRelMatcher	96
Catch::Matchers::Floating::WithinUlpsMatcher	97
Catch::Matchers::Generic::PredicateMatcher< T >	63
Catch::Matchers::StdString::RegexMatcher	67
Catch::Matchers::StdString::StringMatcherBase	83
Catch::Matchers::StdString::ContainsMatcher	23

Catch::Matchers::StdString::EndsWithMatcher	27
Catch::Matchers::StdString::EqualsMatcher	28
Catch::Matchers::StdString::StartsWithMatcher	74
Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >	16
Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >	22
Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >	25
Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >	29
Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >	90
Catch::Matchers::Impl::MatcherUntypedBase	52
Catch::Matchers::Impl::MatcherBase< std::exception >	51
Catch::Matchers::Impl::MatcherBase< double >	51
Catch::Matchers::Impl::MatcherBase< ArgT >	51
Catch::Matchers::Impl::MatcherBase< std::string >	51
Catch::Matchers::Impl::MatcherBase< std::vector< T, AllocMatch > >	51
Catch::Matchers::Impl::MatcherBase< std::vector< T, Alloc > >	51
Catch::Matchers::Impl::MatcherBase< T >	51
Catch::MessageInfo	56
Catch::MessageStream	56
Catch::MessageBuilder	55
Catch::NameAndTags	59
Catch::NonCopyable	59
Catch::AutoReg	19
Catch::IConfig	37
Catch::ReusableStringStream	70
Catch::Section	71
Catch::Option< T >	62
Catch::pluralise	62
Catch::RegistrarForTagAliases	68
Catch::ResultDisposition	69
Catch::ResultWas	70
Catch::RunTests	70
Catch::ScopedMessage	71
Catch::SectionEndInfo	71
Catch::SectionInfo	72
Catch::ShowDurations	72
Catch::SimplePcg32	72
Catch::SourceLineInfo	74
Catch::StreamEndStop	76
Catch::StringMaker< T, typename >	76
Catch::StringMaker< bool >	76
Catch::StringMaker< Catch::Detail::Approx >	76
Catch::StringMaker< char * >	77
Catch::StringMaker< char >	77
Catch::StringMaker< char const * >	77
Catch::StringMaker< char[SZ]>	77
Catch::StringMaker< double >	78
Catch::StringMaker< float >	78
Catch::StringMaker< int >	78
Catch::StringMaker< long >	78
Catch::StringMaker< long long >	79
Catch::StringMaker< R C::* >	79
Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStream← Insertable< R >::value >::type >	79
Catch::StringMaker< signed char >	79
Catch::StringMaker< signed char[SZ]>	80
Catch::StringMaker< std::nullptr_t >	80
Catch::StringMaker< std::string >	80

Catch::StringMaker< std::wstring >	80
Catch::StringMaker< T * >	81
Catch::StringMaker< T[SZ]>	81
Catch::StringMaker< unsigned char >	81
Catch::StringMaker< unsigned char[SZ]>	81
Catch::StringMaker< unsigned int >	82
Catch::StringMaker< unsigned long >	82
Catch::StringMaker< unsigned long long >	82
Catch::StringMaker< wchar_t * >	82
Catch::StringMaker< wchar_t const * >	83
Catch::StringRef	84
Catch::TestCaseInfo	87
Catch::TestCase	86
Catch::TestFailureException	88
Catch::Timer	89
Catch::Totals	89
std::true_type	
Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >()))>::type >	44
Catch::true_given< typename >	89
Catch::UseColour	91
Catch::detail::void_type<... >	94
Catch::WaitForKeypress	94
Catch::WarnAbout	94

Chapter 3

Class Index

3.1 Class List

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

Add	13
Catch::always_false< T >	15
Catch::Detail::Approx	15
Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >	16
Catch::Generators::as< T >	17
Catch::AssertionHandler	18
Catch::AssertionInfo	18
Catch::AssertionReaction	18
Catch::AutoReg	19
Catch::BinaryExpr< LhsT, RhsT >	19
Catch::Capturer	20
Catch::Matchers::StdString::CasedString	20
Catch::CaseSensitive	21
Catch_global_namespace_dummy	21
Catch::Generators::ChunkGenerator< T >	21
Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >	22
Catch::Matchers::StdString::ContainsMatcher	23
Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >	25
Catch::Counts	26
Catch::Decomposer	26
Catch::Matchers::StdString::EndsWithMatcher	27
Catch::Detail::EnumInfo	28
Catch::Matchers::StdString::EqualsMatcher	28
Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >	29
Catch::Matchers::Exception::ExceptionMessageMatcher	30
Catch::ExceptionTranslatorRegistrar	31
Expr	32
Catch::ExprLhs< LhsT >	32
Catch::Generators::FilterGenerator< T, Predicate >	33
Catch::Generators::FixedValuesGenerator< T >	34
Catch::GeneratorException	35
Catch::Generators::Generators< T >	35
Catch::Generators::GeneratorUntypedBase	36
Catch::Generators::GeneratorWrapper< T >	37
Catch::IConfig	37

Catch::IContext	38
Catch::IExceptionTranslator	38
Catch::IExceptionTranslatorRegistry	38
Catch::Generators::IGenerator< T >	39
Catch::IGeneratorTracker	39
Catch::IMutableContext	40
Catch::IMutableEnumValuesRegistry	40
Catch::IMutableRegistryHub	41
Catch::IRegistryHub	41
Catch::IResultCapture	41
Catch::IRunner	42
Catch::is_callable< T >	42
Catch::is_callable< Fun(Args...)>	42
Catch::is_callable_tester	43
Catch::is_range< T >	43
Catch::detail::is_range_impl< T, typename >	43
Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >()))>::type >	44
Catch::Detail::IsStreamInsertable< T >	44
Catch::IStream	44
Catch::Generators::IteratorGenerator< T >	44
Catch::ITestCaseRegistry	45
Catch::ITestInvoker	46
Catch::ITransientExpression	46
Catch::LazyExpression	47
Catch::Generators::MapGenerator< T, U, Func >	47
Catch::Matchers::Impl::MatchAllOf< ArgT >	48
Catch::Matchers::Impl::MatchAnyOf< ArgT >	49
Catch::Matchers::Impl::MatcherBase< T >	51
Catch::Matchers::Impl::MatcherMethod< ObjectT >	52
Catch::Matchers::Impl::MatcherUntypedBase	52
Catch::MatchExpr< ArgT, MatcherT >	53
Catch::Matchers::Impl::MatchNotOf< ArgT >	54
Catch::MessageBuilder	55
Catch::MessageInfo	56
Catch::MessageStream	56
Mult	57
Catch::NameAndTags	59
Catch::NonCopyable	59
Num	60
Catch::Option< T >	62
Catch::pluralise	62
Catch::Matchers::Generic::PredicateMatcher< T >	63
Catch::Generators::RandomFloatingGenerator< Float >	64
Catch::Generators::RandomIntegerGenerator< Integer >	65
Catch::Generators::RangeGenerator< T >	66
Catch::Matchers::StdString::RegexMatcher	67
Catch::RegistrarForTagAliases	68
Catch::Generators::RepeatGenerator< T >	68
Catch::ResultDisposition	69
Catch::ResultWas	70
Catch::ReusableStringStream	70
Catch::RunTests	70
Catch::ScopedMessage	71
Catch::Section	71
Catch::SectionEndInfo	71
Catch::SectionInfo	72
Catch::ShowDurations	72
Catch::SimplePcg32	72

Catch::Generators::SingleValueGenerator< T >	73
Catch::SourceLineInfo	74
Catch::Matchers::StdString::StartsWithMatcher	74
Catch::StreamEndStop	76
Catch::StringMaker< T, typename >	76
Catch::StringMaker< bool >	76
Catch::StringMaker< Catch::Detail::Approx >	76
Catch::StringMaker< char * >	77
Catch::StringMaker< char >	77
Catch::StringMaker< char const * >	77
Catch::StringMaker< char[SZ]>	77
Catch::StringMaker< double >	78
Catch::StringMaker< float >	78
Catch::StringMaker< int >	78
Catch::StringMaker< long >	78
Catch::StringMaker< long long >	79
Catch::StringMaker< R C::* >	79
Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStreamInsertable< R >::value >::value > >	79
Catch::StringMaker< signed char >	79
Catch::StringMaker< signed char[SZ]>	80
Catch::StringMaker< std::nullptr_t >	80
Catch::StringMaker< std::string >	80
Catch::StringMaker< std::wstring >	80
Catch::StringMaker< T * >	81
Catch::StringMaker< T[SZ]>	81
Catch::StringMaker< unsigned char >	81
Catch::StringMaker< unsigned char[SZ]>	81
Catch::StringMaker< unsigned int >	82
Catch::StringMaker< unsigned long >	82
Catch::StringMaker< unsigned long long >	82
Catch::StringMaker< wchar_t * >	82
Catch::StringMaker< wchar_t const * >	83
Catch::Matchers::StdString::StringMatcherBase	83
Catch::StringRef	84
Catch::Generators::TakeGenerator< T >	85
Catch::TestCase	86
Catch::TestCaseInfo	87
Catch::TestFailureException	88
Catch::TestInvokerAsMethod< C >	88
Catch::Timer	89
Catch::Totals	89
Catch::true_given< typename >	89
Catch::UnaryExpr< LhsT >	90
Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >	90
Catch::UseColour	91
Var	92
Catch::detail::void_type<... >	94
Catch::WaitForKeypress	94
Catch::WarnAbout	94
Catch::Matchers::Floating::WithinAbsMatcher	95
Catch::Matchers::Floating::WithinRelMatcher	96
Catch::Matchers::Floating::WithinUlpMatcher	97

Chapter 4

File Index

4.1 File List

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

/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/ catch.hpp	99
/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/ cmdline.h	313
/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/ Expr.h	314

Chapter 5

Class Documentation

5.1 Add Class Reference

Inheritance diagram for Add:



Public Member Functions

- [Add](#) ([Expr](#) *lhs, [Expr](#) *rhs)
- bool [equals](#) ([Expr](#) *e)
- int [interp](#) ()
- bool [has_variable](#) ()
- [Expr](#) * [subst](#) (std::string, [Expr](#) *s)
- void [print](#) (std::ostream &ot)
- void [pretty_print_dr](#) (std::ostream &ot)
- void [pretty_print](#) (std::ostream &ot, precedence_t prec)

Public Member Functions inherited from [Expr](#)

- std::string [to_string](#) ()
- std::string [to_pretty_string](#) ()

Public Attributes

- [Expr](#) * lhs
- [Expr](#) * rhs

5.1.1 Constructor & Destructor Documentation

5.1.1.1 Add()

```
Add::Add (  
    Expr * lhs,  
    Expr * rhs )
```

this is the [Add](#) function

Parameters

<i>lhs</i>	
<i>rhs</i>	

5.1.2 Member Function Documentation

5.1.2.1 equals()

```
bool Add::equals (
    Expr * e ) [virtual]
```

Implements [Expr](#).

5.1.2.2 has_variable()

```
bool Add::has_variable ( ) [virtual]
```

Implements [Expr](#).

5.1.2.3 interp()

```
int Add::interp ( ) [virtual]
```

Implements [Expr](#).

5.1.2.4 pretty_print()

```
void Add::pretty_print (
    std::ostream & ot,
    precedence_t prec ) [virtual]
```

Implements [Expr](#).

5.1.2.5 pretty_print_dr()

```
void Add::pretty_print_dr (
    std::ostream & ot ) [virtual]
```

Implements [Expr](#).

5.1.2.6 print()

```
void Add::print (
    std::ostream & ot ) [virtual]
```

Implements [Expr](#).

5.1.2.7 subst()

```
Expr * Add::subst (
    std::string replace,
    Expr * s ) [virtual]
```

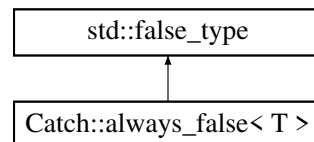
Implements [Expr](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/Expr.h
- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/Expr.cpp

5.2 Catch::always_false< T > Struct Template Reference

Inheritance diagram for Catch::always_false< T >:



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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.3 Catch::Detail::Approx Class Reference

Public Member Functions

- **Approx** ([double](#) value)
- **Approx operator-** () [const](#)
- [template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>](#)
Approx operator() ([T const](#) &value)
- [template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>](#)
Approx ([T const](#) &value)
- [template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>](#)
Approx & epsilon ([T const](#) &[newEpsilon](#))
- [template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>](#)
Approx & margin ([T const](#) &[newMargin](#))
- [template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>](#)
Approx & scale ([T const](#) &[newScale](#))
- [std::string toString](#) () [const](#)

Static Public Member Functions

- [static Approx custom](#) ()

Friends

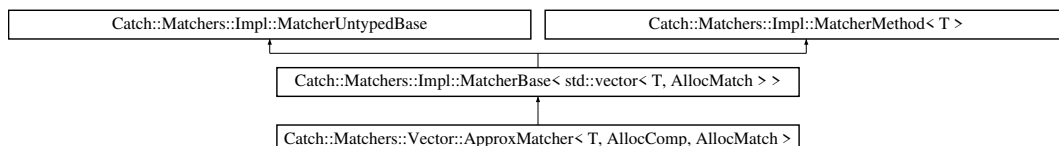
- `template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool operator==(const T &lhs, Approx const &rhs)`
- `template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool operator==(Approx const &lhs, const T &rhs)`
- `template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool operator!=(T const &lhs, Approx const &rhs)`
- `template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool operator!=(Approx const &lhs, T const &rhs)`
- `template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool operator<= (T const &lhs, Approx const &rhs)`
- `template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool operator<= (Approx const &lhs, T const &rhs)`
- `template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool operator>= (T const &lhs, Approx const &rhs)`
- `template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool operator>= (Approx const &lhs, T const &rhs)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.4 Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch > Struct Template Reference

Inheritance diagram for `Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >`:



Public Member Functions

- **ApproxMatcher** (`std::vector< T, AllocComp >` const &comparator)
- **bool match** (`std::vector< T, AllocMatch >` const &v) const override
- `std::string describe` () const override
- `template<typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> ApproxMatcher & epsilon` (`T const &newEpsilon`)
- `template<typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> ApproxMatcher & margin` (`T const &newMargin`)
- `template<typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> ApproxMatcher & scale` (`T const &newScale`)

Public Member Functions inherited from `Catch::Matchers::Impl::MatcherBase< T >`

- `MatchAllOf< T >` **operator&&** (`MatcherBase const &other`) const
- `MatchAnyOf< T >` **operator||** (`MatcherBase const &other`) const
- `MatchNotOf< T >` **operator!** () const

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** ([MatcherUntypedBase const &](#))=default
- [MatcherUntypedBase &](#) **operator=** ([MatcherUntypedBase const &](#))=delete
- `std::string` **toString** () [const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- `virtual bool` **match** ([T const &arg](#)) [const](#)=0

Public Attributes

- `std::vector< T, AllocComp >` [const](#) & **m_comparator**
- [Catch::Detail::Approx](#) **approx** = [Catch::Detail::Approx::custom](#)()

Additional Inherited Members**Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- `std::string` **m_cachedToString**

5.4.1 Member Function Documentation**5.4.1.1 describe()**

```
template<typename T , typename AllocComp , typename AllocMatch >
std::string Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >::describe ( )
const [inline], [override], [virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.5 Catch::Generators::as< T > Struct Template Reference

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.6 Catch::AssertionHandler Class Reference

Public Member Functions

- **AssertionHandler** ([StringRef](#) const ¯oName, [SourceLineInfo](#) const &lineInfo, [StringRef](#) captured←→ Expression, [ResultDisposition::Flags](#) resultDisposition)
- `template<typename T >`
[void](#) **handleExpr** ([ExprLhs](#)< T > const &expr)
- [void](#) **handleExpr** ([ITransientExpression](#) const &expr)
- [void](#) **handleMessage** ([ResultWas::OfType](#) resultType, [StringRef](#) const &message)
- [void](#) **handleExceptionThrownAsExpected** ()
- [void](#) **handleUnexpectedExceptionNotThrown** ()
- [void](#) **handleExceptionNotThrownAsExpected** ()
- [void](#) **handleThrowingCallSkipped** ()
- [void](#) **handleUnexpectedInflightException** ()
- [void](#) **complete** ()
- [void](#) **setCompleted** ()
- [auto](#) **allowThrows** () const -> [bool](#)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.7 Catch::AssertionInfo Struct Reference

Public Attributes

- [StringRef](#) macroName
- [SourceLineInfo](#) lineInfo
- [StringRef](#) capturedExpression
- [ResultDisposition::Flags](#) resultDisposition

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.8 Catch::AssertionReaction Struct Reference

Public Attributes

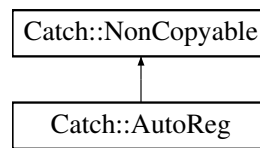
- [bool](#) shouldDebugBreak = [false](#)
- [bool](#) shouldThrow = [false](#)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.9 Catch::AutoReg Struct Reference

Inheritance diagram for Catch::AutoReg:



Public Member Functions

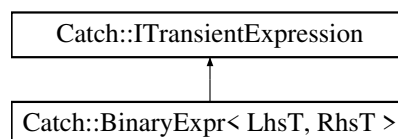
- **AutoReg** (ITestInvoker *invoker, SourceLineInfo const &lineInfo, StringRef const &classOrMethod, NameAndTags const &nameAndTags) noexcept

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.10 Catch::BinaryExpr< LhsT, RhsT > Class Template Reference

Inheritance diagram for Catch::BinaryExpr< LhsT, RhsT >:



Public Member Functions

- **BinaryExpr** (bool comparisonResult, LhsT lhs, StringRef op, RhsT rhs)
- template<typename T>
 auto operator&& (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T>
 auto operator|| (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T>
 auto operator== (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T>
 auto operator!= (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T>
 auto operator> (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T>
 auto operator< (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T>
 auto operator>= (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T>
 auto operator<= (T) const -> BinaryExpr< LhsT, RhsT const & > const

Public Member Functions inherited from [Catch::ITransientExpression](#)

- [auto isBinaryExpression \(\) const](#) -> [bool](#)
- [auto getResult \(\) const](#) -> [bool](#)
- [ITransientExpression \(bool isBinaryExpression, bool result\)](#)

Additional Inherited Members

Public Attributes inherited from [Catch::ITransientExpression](#)

- [bool m_isBinaryExpression](#)
- [bool m_result](#)

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.11 [Catch::Capturer](#) Class Reference

Public Member Functions

- [Capturer \(StringRef macroName, SourceLineInfo const &lineInfo, ResultWas::OfType resultType, StringRef names\)](#)
- [void captureValue \(size_t index, std::string const &value\)](#)
- [template<typename T > void captureValues \(size_t index, T const &value\)](#)
- [template<typename T, typename... Ts> void captureValues \(size_t index, T const &value, Ts const &... values\)](#)

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.12 [Catch::Matchers::StdString::CasedString](#) Struct Reference

Public Member Functions

- [CasedString \(std::string const &str, CaseSensitive::Choice caseSensitivity\)](#)
- [std::string adjustString \(std::string const &str\) const](#)
- [std::string caseSensitivitySuffix \(\) const](#)

Public Attributes

- [CaseSensitive::Choice m_caseSensitivity](#)
- [std::string m_str](#)

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.13 Catch::CaseSensitive Struct Reference

Public Types

- enum **Choice** { **Yes** , **No** }

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

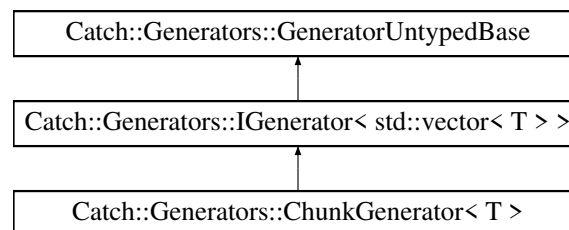
5.14 Catch_global_namespace_dummy Struct Reference

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.15 Catch::Generators::ChunkGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::ChunkGenerator< T >:



Public Member Functions

- **ChunkGenerator** ([size_t](#) size, [GeneratorWrapper< T >](#) generator)
- `std::vector< T > const & get () const` override
- `bool next ()` override

Additional Inherited Members

Public Types inherited from [Catch::Generators::IGenerator< std::vector< T > >](#)

- [using](#) type

5.15.1 Member Function Documentation

5.15.1.1 get()

```
template<typename T >
std::vector< T > const & Catch::Generators::ChunkGenerator< T >::get ( ) const [inline],
[override], [virtual]
```

Implements [Catch::Generators::IGenerator< std::vector< T > >](#).

5.15.1.2 next()

```
template<typename T >
bool Catch::Generators::ChunkGenerator< T >::next ( ) [inline], [override], [virtual]
```

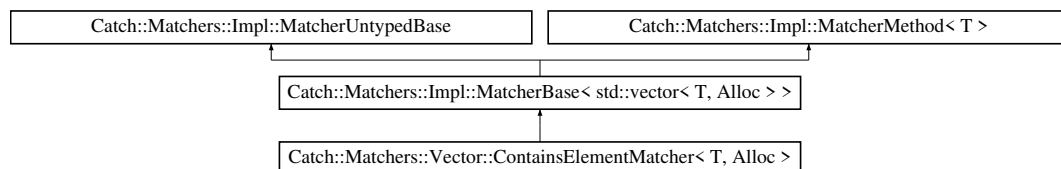
Implements [Catch::Generators::GeneratorUntypedBase](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.16 [Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >](#) Struct Template Reference

Inheritance diagram for [Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >](#):



Public Member Functions

- **ContainsElementMatcher** ([T const](#) &[comparator](#))
- **bool match** ([std::vector< T, Alloc > const](#) &[v](#)) [const override](#)
- [std::string describe](#) () [const override](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- [MatchAllOf< T > operator&&](#) ([MatcherBase const](#) &[other](#)) [const](#)
- [MatchAnyOf< T > operator||](#) ([MatcherBase const](#) &[other](#)) [const](#)
- [MatchNotOf< T > operator!](#) () [const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** ([MatcherUntypedBase const &](#))=default
- [MatcherUntypedBase &](#) **operator=** ([MatcherUntypedBase const &](#))=delete
- `std::string` **toString** () [const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- `virtual bool` **match** ([T const &arg](#)) [const=0](#)

Public Attributes

- [T const](#) & **m_comparator**

Additional Inherited Members**Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- `std::string` **m_cachedToString**

5.16.1 Member Function Documentation**5.16.1.1 describe()**

```
template<typename T , typename Alloc >
std::string Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >::describe ( ) const
[inline], [override], [virtual]
```

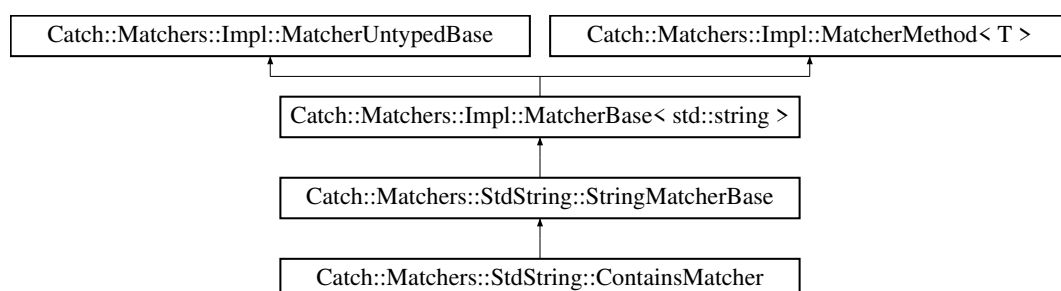
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.17 Catch::Matchers::StdString::ContainsMatcher Struct Reference

Inheritance diagram for [Catch::Matchers::StdString::ContainsMatcher](#):



Public Member Functions

- **ContainsMatcher** ([CasedString const](#) &comparator)
- **bool match** (std::string [const](#) &source) [const override](#)

Public Member Functions inherited from [Catch::Matchers::StdString::StringMatcherBase](#)

- **StringMatcherBase** (std::string [const](#) &operation, [CasedString const](#) &comparator)
- std::string **describe** () [const override](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- **MatchAllOf**< T > **operator&&** ([MatcherBase const](#) &other) [const](#)
- **MatchAnyOf**< T > **operator||** ([MatcherBase const](#) &other) [const](#)
- **MatchNotOf**< T > **operator!** () [const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** ([MatcherUntypedBase const](#) &)=default
- **MatcherUntypedBase** & **operator=** ([MatcherUntypedBase const](#) &)=delete
- std::string **toString** () [const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- **virtual bool match** ([T const](#) &arg) [const=0](#)

Additional Inherited Members**Public Attributes inherited from [Catch::Matchers::StdString::StringMatcherBase](#)**

- [CasedString](#) **m_comparator**
- std::string **m_operation**

Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

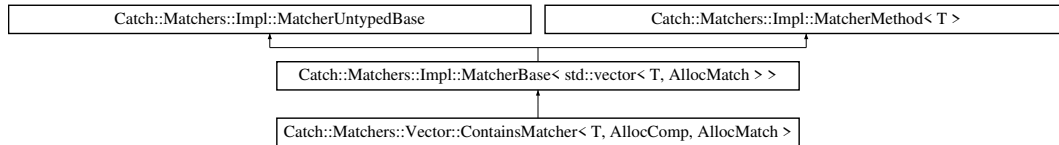
- std::string **m_cachedToString**

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.18 Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch > Struct Template Reference

Inheritance diagram for Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >:



Public Member Functions

- **ContainsMatcher** (std::vector< T, AllocComp > const &comparator)
- **bool match** (std::vector< T, AllocMatch > const &v) const override
- std::string **describe** () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- **MatchAllOf**< T > **operator&&** (MatcherBase const &other) const
- **MatchAnyOf**< T > **operator||** (MatcherBase const &other) const
- **MatchNotOf**< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- **MatcherUntypedBase & operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- **virtual bool match** (T const &arg) const=0

Public Attributes

- std::vector< T, AllocComp > const & **m_comparator**

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

5.18.1 Member Function Documentation

5.18.1.1 describe()

```
template<typename T , typename AllocComp , typename AllocMatch >
std::string Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >::describe ( )
const [inline], [override], [virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.19 Catch::Counts Struct Reference

Public Member Functions

- [Counts operator-](#) ([Counts const](#) &[other](#)) [const](#)
- [Counts & operator+=](#) ([Counts const](#) &[other](#))
- [std::size_t total](#) () [const](#)
- [bool allPassed](#) () [const](#)
- [bool allOk](#) () [const](#)

Public Attributes

- [std::size_t passed](#) = 0
- [std::size_t failed](#) = 0
- [std::size_t failedButOk](#) = 0

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.20 Catch::Decomposer Struct Reference

Public Member Functions

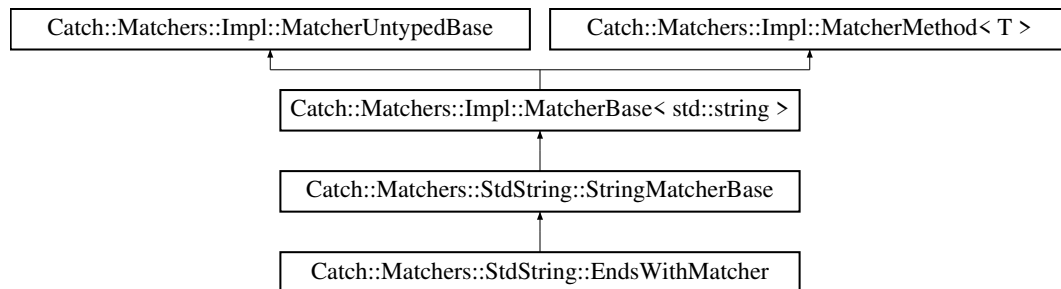
- [template<typename T >](#)
[auto operator<=](#) ([T const](#) &[lhs](#)) -> [ExprLhs< T const & >](#)
- [auto operator<=](#) ([bool value](#)) -> [ExprLhs< bool >](#)

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.21 Catch::Matchers::StdString::EndsWithMatcher Struct Reference

Inheritance diagram for Catch::Matchers::StdString::EndsWithMatcher:



Public Member Functions

- **EndsWithMatcher** ([CasedString](#) const &comparator)
- **bool match** (std::string const &source) const override

Public Member Functions inherited from [Catch::Matchers::StdString::StringMatcherBase](#)

- **StringMatcherBase** (std::string const &operation, [CasedString](#) const &comparator)
- std::string **describe** () const override

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- **MatchAllOf< T > operator&&** ([MatcherBase](#) const &other) const
- **MatchAnyOf< T > operator||** ([MatcherBase](#) const &other) const
- **MatchNotOf< T > operator!** () const

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** ([MatcherUntypedBase](#) const &)=default
- **MatcherUntypedBase & operator=** ([MatcherUntypedBase](#) const &)=delete
- std::string **toString** () const

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- **virtual bool match** (T const &arg) const=0

Additional Inherited Members

Public Attributes inherited from [Catch::Matchers::StdString::StringMatcherBase](#)

- [CasedString](#) m_comparator
- std::string m_operation

Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- `std::string m_cachedToString`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.22 [Catch::Detail::EnumInfo](#) Struct Reference

Public Member Functions

- [StringRef](#) `lookup` (`int` value) `const`

Public Attributes

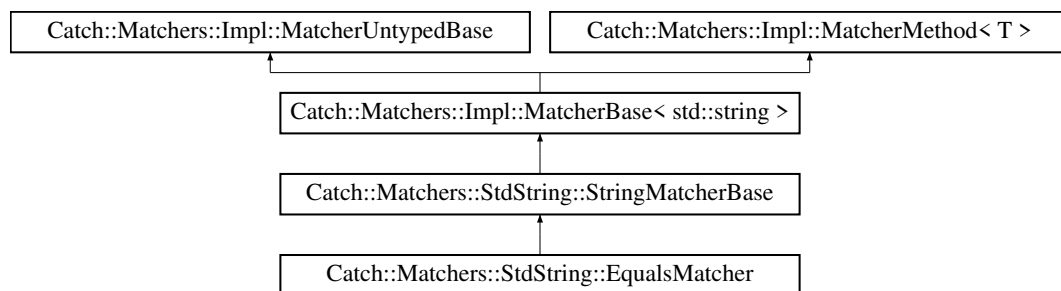
- [StringRef](#) `m_name`
- `std::vector< std::pair< int, StringRef > >` `m_values`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.23 [Catch::Matchers::StdString::EqualsMatcher](#) Struct Reference

Inheritance diagram for [Catch::Matchers::StdString::EqualsMatcher](#):



Public Member Functions

- `EqualsMatcher` (`CasedString` `const` &`comparator`)
- `bool match` (`std::string` `const` &`source`) `const` override

Public Member Functions inherited from [Catch::Matchers::StdString::StringMatcherBase](#)

- `StringMatcherBase` (`std::string` `const` &`operation`, `CasedString` `const` &`comparator`)
- `std::string describe` () `const` override

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- [MatchAllOf< T > operator&& \(MatcherBase const &other\) const](#)
- [MatchAnyOf< T > operator|| \(MatcherBase const &other\) const](#)
- [MatchNotOf< T > operator! \(\) const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [MatcherUntypedBase \(MatcherUntypedBase const &\)=default](#)
- [MatcherUntypedBase & operator= \(MatcherUntypedBase const &\)=delete](#)
- [std::string toString \(\) const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- [virtual bool match \(T const &arg\) const=0](#)

Additional Inherited Members**Public Attributes inherited from [Catch::Matchers::StdString::StringMatcherBase](#)**

- [CasedString m_comparator](#)
- [std::string m_operation](#)

Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

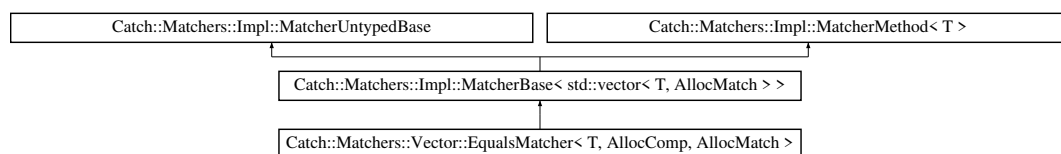
- [std::string m_cachedToString](#)

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

- [/Users/taolangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.24 [Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >](#) Struct Template Reference

Inheritance diagram for [Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >](#):

**Public Member Functions**

- [EqualsMatcher \(std::vector< T, AllocComp > const &comparator\)](#)
- [bool match \(std::vector< T, AllocMatch > const &v\) const override](#)
- [std::string describe \(\) const override](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- [MatchAllOf< T > operator&& \(MatcherBase const &other\) const](#)
- [MatchAnyOf< T > operator|| \(MatcherBase const &other\) const](#)
- [MatchNotOf< T > operator! \(\) const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [MatcherUntypedBase \(MatcherUntypedBase const &\)=default](#)
- [MatcherUntypedBase & operator= \(MatcherUntypedBase const &\)=delete](#)
- [std::string toString \(\) const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- [virtual bool match \(T const &arg\) const=0](#)

Public Attributes

- [std::vector< T, AllocComp > const & m_comparator](#)

Additional Inherited Members

Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [std::string m_cachedToString](#)

5.24.1 Member Function Documentation

5.24.1.1 describe()

```
template<typename T , typename AllocComp , typename AllocMatch >
std::string Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >::describe ( )
const [inline], [override], [virtual]
```

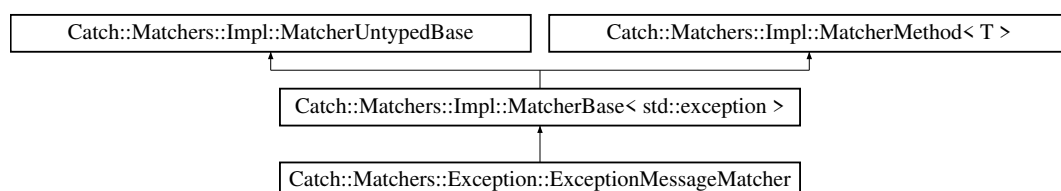
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.25 Catch::Matchers::Exception::ExceptionMessageMatcher Class Reference

Inheritance diagram for [Catch::Matchers::Exception::ExceptionMessageMatcher](#):



Public Member Functions

- **ExceptionMessageMatcher** (std::string const &message)
- **bool match** (std::exception const &ex) const override
- std::string **describe** () const override

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- **MatchAllOf< T > operator&&** (MatcherBase const &other) const
- **MatchAnyOf< T > operator||** (MatcherBase const &other) const
- **MatchNotOf< T > operator!** () const

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- **MatcherUntypedBase & operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- **virtual bool match** (T const &arg) const=0

Additional Inherited Members**Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- std::string **m_cachedToString**

5.25.1 Member Function Documentation**5.25.1.1 describe()**

```
std::string Catch::Matchers::Exception::ExceptionMessageMatcher::describe ( ) const [override],
[virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.26 Catch::ExceptionTranslatorRegistrar Class Reference**Public Member Functions**

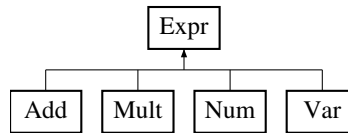
- **template<typename T >**
ExceptionTranslatorRegistrar (std::string(*translateFunction)(T &))

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.27 Expr Class Reference

Inheritance diagram for Expr:



Public Member Functions

- virtual bool **equals** (Expr *e)=0
- virtual int **interp** ()=0
- virtual bool **has_variable** ()=0
- virtual Expr * **subst** (std::string, Expr *s)=0
- virtual void **print** (std::ostream &ot)=0
- virtual void **pretty_print** (std::ostream &ot, precedence_t prec)=0
- std::string **to_string** ()
- virtual void **pretty_print_dr** (std::ostream &ot)=0
- std::string **to_pretty_string** ()

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/Expr.h

5.28 Catch::ExprLhs< LhsT > Class Template Reference

Public Member Functions

- ExprLhs (LhsT lhs)
- template<typename Rhst >
auto operator== (Rhst const &rhs) -> BinaryExpr< LhsT, Rhst const & > const
- auto operator== (bool rhs) -> BinaryExpr< LhsT, bool > const
- template<typename Rhst >
auto operator!= (Rhst const &rhs) -> BinaryExpr< LhsT, Rhst const & > const
- auto operator!= (bool rhs) -> BinaryExpr< LhsT, bool > const
- template<typename Rhst >
auto operator> (Rhst const &rhs) -> BinaryExpr< LhsT, Rhst const & > const
- template<typename Rhst >
auto operator< (Rhst const &rhs) -> BinaryExpr< LhsT, Rhst const & > const
- template<typename Rhst >
auto operator>= (Rhst const &rhs) -> BinaryExpr< LhsT, Rhst const & > const
- template<typename Rhst >
auto operator<= (Rhst const &rhs) -> BinaryExpr< LhsT, Rhst const & > const
- template<typename Rhst >
auto operator| (Rhst const &rhs) -> BinaryExpr< LhsT, Rhst const & > const
- template<typename Rhst >
auto operator& (Rhst const &rhs) -> BinaryExpr< LhsT, Rhst const & > const

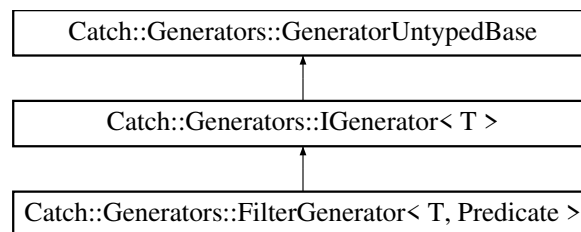
- `template<typename RhsT >`
`auto operator^ (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const & > const`
- `template<typename RhsT >`
`auto operator&& (RhsT const &) -> BinaryExpr< LhsT, RhsT const & > const`
- `template<typename RhsT >`
`auto operator|| (RhsT const &) -> BinaryExpr< LhsT, RhsT const & > const`
- `auto makeUnaryExpr () const -> UnaryExpr< LhsT >`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.29 Catch::Generators::FilterGenerator< T, Predicate > Class Template Reference

Inheritance diagram for `Catch::Generators::FilterGenerator< T, Predicate >`:



Public Member Functions

- `template<typename P = Predicate>`
`FilterGenerator (P &&pred, GeneratorWrapper< T > &&generator)`
- `T const & get () const override`
- `bool next () override`

Additional Inherited Members

Public Types inherited from `Catch::Generators::IGenerator< T >`

- `using type = T`

5.29.1 Member Function Documentation

5.29.1.1 get()

```

template<typename T , typename Predicate >
T const & Catch::Generators::FilterGenerator< T, Predicate >::get ( ) const [inline], [override],
[virtual]
  
```

Implements `Catch::Generators::IGenerator< T >`.

5.29.1.2 next()

```
template<typename T , typename Predicate >
bool Catch::Generators::FilterGenerator< T, Predicate >::next ( ) [inline], [override], [virtual]
```

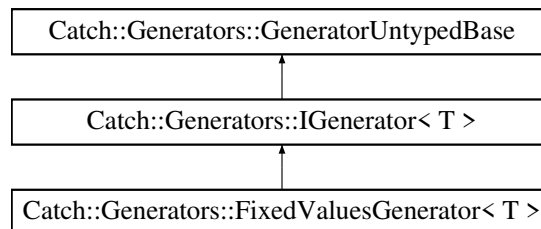
Implements [Catch::Generators::GeneratorUntypedBase](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.30 Catch::Generators::FixedValuesGenerator< T > Class Template Reference

Inheritance diagram for `Catch::Generators::FixedValuesGenerator< T >`:



Public Member Functions

- **FixedValuesGenerator** (std::initializer_list< T > values)
- T const & get () const override
- bool next () override

Additional Inherited Members

Public Types inherited from [Catch::Generators::IGenerator< T >](#)

- using type = T

5.30.1 Member Function Documentation

5.30.1.1 get()

```
template<typename T >
T const & Catch::Generators::FixedValuesGenerator< T >::get ( ) const [inline], [override],
[virtual]
```

Implements [Catch::Generators::IGenerator< T >](#).

5.30.1.2 next()

```
template<typename T >
bool Catch::Generators::FixedValuesGenerator< T >::next ( ) [inline], [override], [virtual]
```

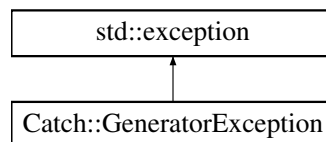
Implements [Catch::Generators::GeneratorUntypedBase](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.31 Catch::GeneratorException Class Reference

Inheritance diagram for Catch::GeneratorException:



Public Member Functions

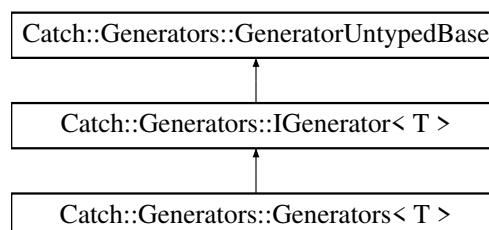
- **GeneratorException** ([const char *msg](#))
- [const char * what \(\) const noexcept override final](#)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.32 Catch::Generators::Generators< T > Class Template Reference

Inheritance diagram for Catch::Generators::Generators< T >:



Public Member Functions

- [template<typename... Gs>](#)
Generators ([Gs &&... moreGenerators](#))
- [T const & get \(\) const override](#)
- [bool next \(\) override](#)

Additional Inherited Members

Public Types inherited from `Catch::Generators::IGenerator< T >`

- `using type = T`

5.32.1 Member Function Documentation

5.32.1.1 `get()`

```
template<typename T >
T const & Catch::Generators::Generators< T >::get ( ) const [inline], [override], [virtual]
```

Implements `Catch::Generators::IGenerator< T >`.

5.32.1.2 `next()`

```
template<typename T >
bool Catch::Generators::Generators< T >::next ( ) [inline], [override], [virtual]
```

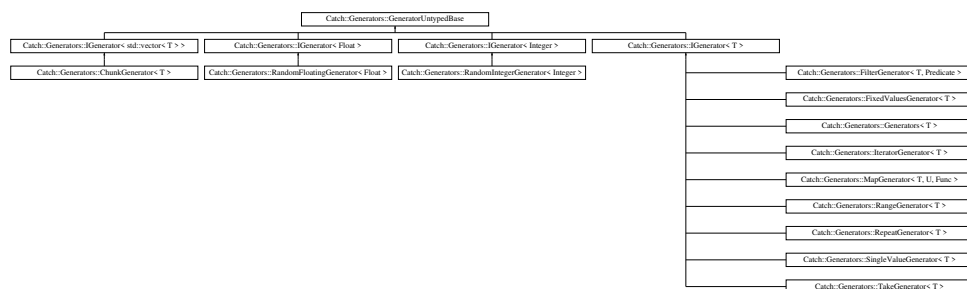
Implements `Catch::Generators::GeneratorUntypedBase`.

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.33 `Catch::Generators::GeneratorUntypedBase` Class Reference

Inheritance diagram for `Catch::Generators::GeneratorUntypedBase`:



Public Member Functions

- `virtual bool next ()=0`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.34 Catch::Generators::GeneratorWrapper< T > Class Template Reference

Public Member Functions

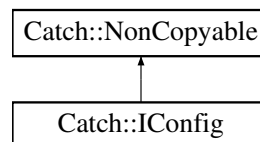
- **GeneratorWrapper** (std::unique_ptr< IGenerator< T > > generator)
- T const & **get** () const
- bool **next** ()

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.35 Catch::IConfig Struct Reference

Inheritance diagram for Catch::IConfig:



Public Member Functions

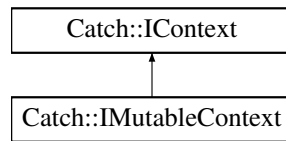
- virtual bool **allowThrows** () const =0
- virtual std::ostream & **stream** () const =0
- virtual std::string **name** () const =0
- virtual bool **includeSuccessfulResults** () const =0
- virtual bool **shouldDebugBreak** () const =0
- virtual bool **warnAboutMissingAssertions** () const =0
- virtual bool **warnAboutNoTests** () const =0
- virtual int **abortAfter** () const =0
- virtual bool **showInvisibles** () const =0
- virtual ShowDurations::OrNot **showDurations** () const =0
- virtual double **minDuration** () const =0
- virtual TestSpec const & **testSpec** () const =0
- virtual bool **hasTestFilters** () const =0
- virtual std::vector< std::string > const & **getTestsOrTags** () const =0
- virtual RunTests::InWhatOrder **runOrder** () const =0
- virtual unsigned int **rngSeed** () const =0
- virtual UseColour::YesOrNo **useColour** () const =0
- virtual std::vector< std::string > const & **getSectionsToRun** () const =0
- virtual Verbosity **verbosity** () const =0
- virtual bool **benchmarkNoAnalysis** () const =0
- virtual int **benchmarkSamples** () const =0
- virtual double **benchmarkConfidenceInterval** () const =0
- virtual unsigned int **benchmarkResamples** () const =0
- virtual std::chrono::milliseconds **benchmarkWarmupTime** () const =0

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.36 Catch::IContext Struct Reference

Inheritance diagram for Catch::IContext:



Public Member Functions

- `virtual IResultCapture * getResultCapture ()=0`
- `virtual IRunner * getRunner ()=0`
- `virtual IConfigPtr const & getConfig () const =0`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.37 Catch::IExceptionTranslator Struct Reference

Public Member Functions

- `virtual std::string translate (ExceptionTranslators::const_iterator it, ExceptionTranslators::const_iterator itEnd) const =0`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.38 Catch::IExceptionTranslatorRegistry Struct Reference

Public Member Functions

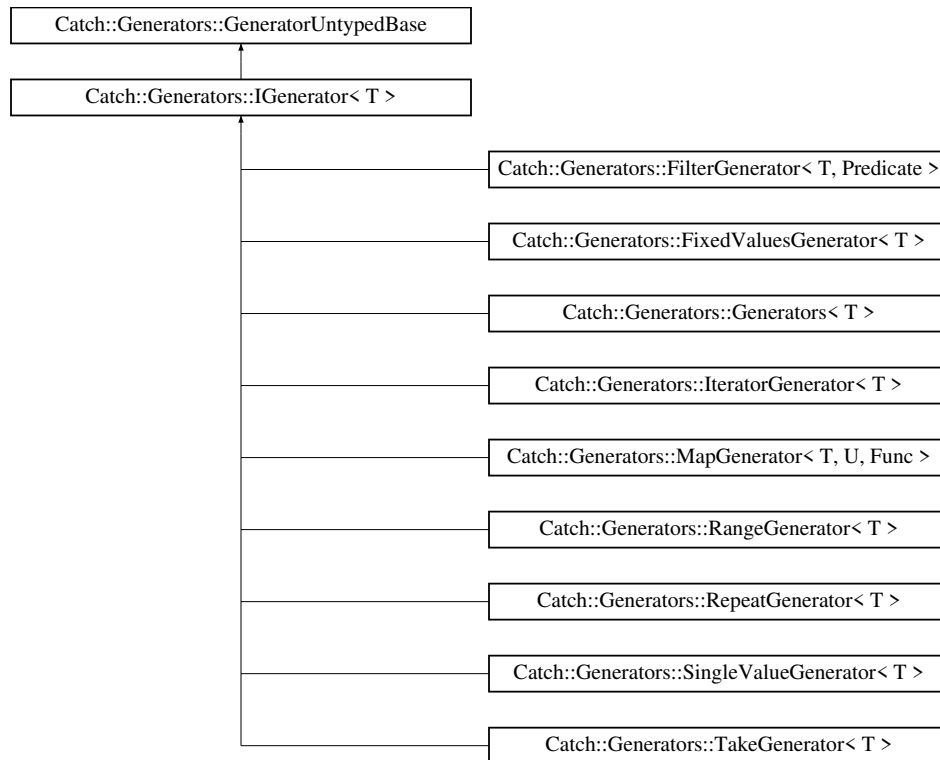
- `virtual std::string translateActiveException () const =0`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.39 Catch::Generators::IGenerator< T > Struct Template Reference

Inheritance diagram for Catch::Generators::IGenerator< T >:



Public Types

- `using type = T`

Public Member Functions

- `virtual T const & get () const =0`

Public Member Functions inherited from [Catch::Generators::GeneratorUntypedBase](#)

- `virtual bool next ()=0`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.40 Catch::IGeneratorTracker Struct Reference

Public Member Functions

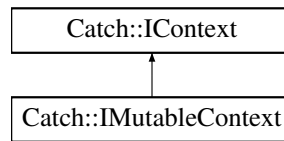
- `virtual auto hasGenerator () const -> bool=0`
- `virtual auto getGenerator () const -> Generators::GeneratorBasePtr const &=0`
- `virtual void setGenerator (Generators::GeneratorBasePtr &&generator)=0`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.41 Catch::ImmutableContext Struct Reference

Inheritance diagram for Catch::ImmutableContext:



Public Member Functions

- `virtual void setResultCapture (IResultCapture *resultCapture)=0`
- `virtual void setRunner (IRunner *runner)=0`
- `virtual void setConfig (IConfigPtr const &config)=0`

Public Member Functions inherited from Catch::IContext

- `virtual IResultCapture * getResultCapture ()=0`
- `virtual IRunner * getRunner ()=0`
- `virtual IConfigPtr const & getConfig () const =0`

Friends

- `ImmutableContext & getCurrentMutableContext ()`
- `void cleanUpContext ()`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.42 Catch::ImmutableEnumValuesRegistry Struct Reference

Public Member Functions

- `virtual Detail::EnumInfo const & registerEnum (StringRef enumName, StringRef allEnums, std::vector< int > const &values)=0`
- `template<typename E> Detail::EnumInfo const & registerEnum (StringRef enumName, StringRef allEnums, std::initializer_list< E > values)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.43 Catch::IMutableRegistryHub Struct Reference

Public Member Functions

- **virtual void registerReporter** (std::string const &name, IReporterFactoryPtr const &factory)=0
- **virtual void registerListener** (IReporterFactoryPtr const &factory)=0
- **virtual void registerTest** (TestCase const &testInfo)=0
- **virtual void registerTranslator** (const IExceptionTranslator *translator)=0
- **virtual void registerTagAlias** (std::string const &alias, std::string const &tag, SourceLineInfo const &lineInfo)=0
- **virtual void registerStartupException** () noexcept=0
- **virtual IMutableEnumValuesRegistry & getMutableEnumValuesRegistry** ()=0

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.44 Catch::IRegistryHub Struct Reference

Public Member Functions

- **virtual IReporterRegistry const & getReporterRegistry** () const =0
- **virtual ITestCaseRegistry const & getTestCaseRegistry** () const =0
- **virtual ITagAliasRegistry const & getTagAliasRegistry** () const =0
- **virtual IExceptionTranslatorRegistry const & getExceptionTranslatorRegistry** () const =0
- **virtual StartupExceptionRegistry const & getStartupExceptionRegistry** () const =0

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.45 Catch::IResultCapture Struct Reference

Public Member Functions

- **virtual bool sectionStarted** (SectionInfo const §ionInfo, Counts &assertions)=0
- **virtual void sectionEnded** (SectionEndInfo const &endInfo)=0
- **virtual void sectionEndedEarly** (SectionEndInfo const &endInfo)=0
- **virtual auto acquireGeneratorTracker** (StringRef generatorName, SourceLineInfo const &lineInfo) -> IGeneratorTracker &=0
- **virtual void pushScopedMessage** (MessageInfo const &message)=0
- **virtual void popScopedMessage** (MessageInfo const &message)=0
- **virtual void emplaceUnscopedMessage** (MessageBuilder const &builder)=0
- **virtual void handleFatalErrorCondition** (StringRef message)=0
- **virtual void handleExpr** (AssertionInfo const &info, ITransientExpression const &expr, AssertionReaction &reaction)=0
- **virtual void handleMessage** (AssertionInfo const &info, ResultWas::OfType resultType, StringRef const &message, AssertionReaction &reaction)=0

- `virtual void handleUnexpectedExceptionNotThrown (AssertionInfo const &info, AssertionReaction &reaction)=0`
- `virtual void handleUnexpectedInflightException (AssertionInfo const &info, std::string const &message, AssertionReaction &reaction)=0`
- `virtual void handleIncomplete (AssertionInfo const &info)=0`
- `virtual void handleNonExpr (AssertionInfo const &info, ResultWas::OfType resultType, AssertionReaction &reaction)=0`
- `virtual bool lastAssertionPassed ()=0`
- `virtual void assertionPassed ()=0`
- `virtual std::string getCurrentTestName () const =0`
- `virtual const AssertionResult * getLastResult () const =0`
- `virtual void exceptionEarlyReported ()=0`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.46 Catch::IRunner Struct Reference

Public Member Functions

- `virtual bool aborting () const =0`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.47 Catch::is_callable< T > Struct Template Reference

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.48 Catch::is_callable< Fun(Args...) > Struct Template Reference

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.49 Catch::is_callable_tester Struct Reference

Static Public Member Functions

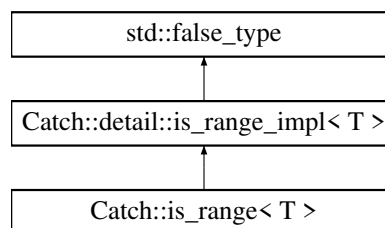
- `template<typename Fun , typename... Args>
static true_given< decltype(std::declval< Fun >()(std::declval< Args >())...)> test (int)`
- `template<typename... >
static std::false_type test (...)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.50 Catch::is_range< T > Struct Template Reference

Inheritance diagram for Catch::is_range< T >:

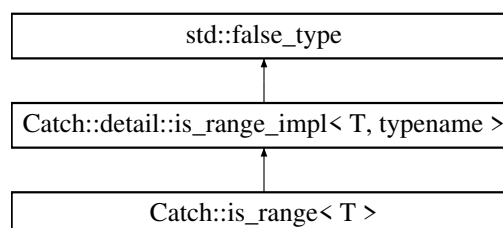


The documentation for this struct was generated from the following file:

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.51 Catch::detail::is_range_impl< T, typename > Struct Template Reference

Inheritance diagram for Catch::detail::is_range_impl< T, typename >:

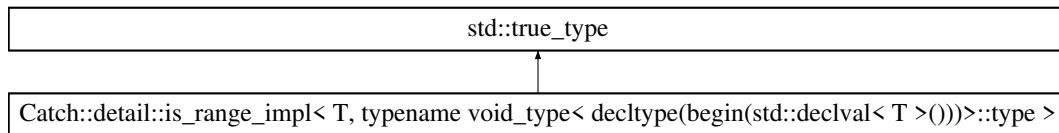


The documentation for this struct was generated from the following file:

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.52 `Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >()))>::type >` Struct Template Reference

Inheritance diagram for `Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >()))>::type >`:



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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.53 `Catch::Detail::IsStreamInsertable< T >` Class Template Reference

Static Public Attributes

- `static const bool value = decltype(test<std::ostream, const T&>(0))::value`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.54 `Catch::IStream` Struct Reference

Public Member Functions

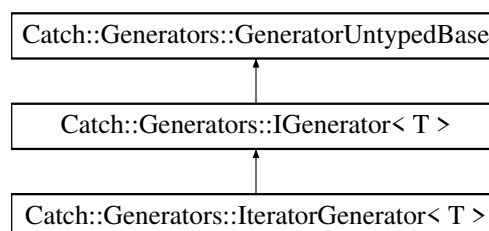
- `virtual std::ostream & stream () const =0`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.55 `Catch::Generators::IteratorGenerator< T >` Class Template Reference

Inheritance diagram for `Catch::Generators::IteratorGenerator< T >`:



Public Member Functions

- `template<typename InputIterator, typename InputSentinel > IteratorGenerator (InputIterator first, InputSentinel last)`
- `T const & get () const override`
- `bool next () override`

Additional Inherited Members

Public Types inherited from `Catch::Generators::IGenerator< T >`

- `using type = T`

5.55.1 Member Function Documentation

5.55.1.1 `get()`

```
template<typename T >
T const & Catch::Generators::IteratorGenerator< T >::get ( ) const [inline], [override], [virtual]
```

Implements `Catch::Generators::IGenerator< T >`.

5.55.1.2 `next()`

```
template<typename T >
bool Catch::Generators::IteratorGenerator< T >::next ( ) [inline], [override], [virtual]
```

Implements `Catch::Generators::GeneratorUntypedBase`.

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.56 Catch::ITestCaseRegistry Struct Reference

Public Member Functions

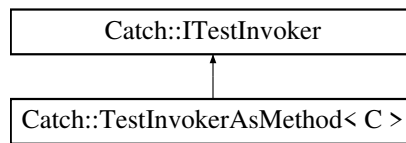
- `virtual std::vector< TestCase > const & getAllTests () const =0`
- `virtual std::vector< TestCase > const & getAllTestsSorted (IConfig const &config) const =0`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.57 Catch::ITestInvoker Struct Reference

Inheritance diagram for Catch::ITestInvoker:



Public Member Functions

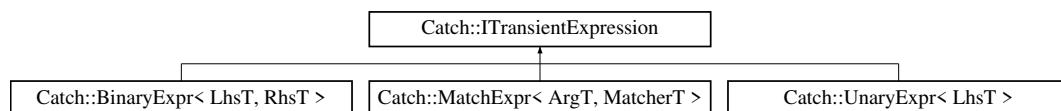
- `virtual void invoke () const =0`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.58 Catch::ITransientExpression Struct Reference

Inheritance diagram for Catch::ITransientExpression:



Public Member Functions

- `auto isBinaryExpression () const -> bool`
- `auto getResult () const -> bool`
- `virtual void streamReconstructedExpression (std::ostream &os) const =0`
- `ITransientExpression (bool isBinaryExpression, bool result)`

Public Attributes

- `bool m_isBinaryExpression`
- `bool m_result`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.59 Catch::LazyExpression Class Reference

Public Member Functions

- **LazyExpression** ([bool isNegated](#))
- **LazyExpression** ([LazyExpression const &other](#))
- [LazyExpression](#) & **operator=** ([LazyExpression const &](#))=delete
- **operator bool** () [const](#)

Friends

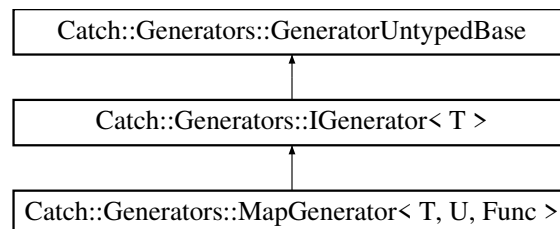
- [class](#) **AssertionHandler**
- [struct](#) **AssertionStats**
- [class](#) **RunContext**
- [auto](#) **operator<<** ([std::ostream &os](#), [LazyExpression const &lazyExpr](#)) -> [std::ostream &](#)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.60 Catch::Generators::MapGenerator< T, U, Func > Class Template Reference

Inheritance diagram for Catch::Generators::MapGenerator< T, U, Func >:



Public Member Functions

- [template<typename F2 = Func>](#)
MapGenerator ([F2 &&function](#), [GeneratorWrapper< U > &&generator](#))
- [T const &](#) **get** () [const override](#)
- [bool](#) **next** () [override](#)

Additional Inherited Members

Public Types inherited from [Catch::Generators::IGenerator< T >](#)

- [using](#) **type** = [T](#)

5.60.1 Member Function Documentation

5.60.1.1 get()

```
template<typename T , typename U , typename Func >
T const & Catch::Generators::MapGenerator< T, U, Func >::get ( ) const [inline], [override],
[virtual]
```

Implements [Catch::Generators::IGenerator< T >](#).

5.60.1.2 next()

```
template<typename T , typename U , typename Func >
bool Catch::Generators::MapGenerator< T, U, Func >::next ( ) [inline], [override], [virtual]
```

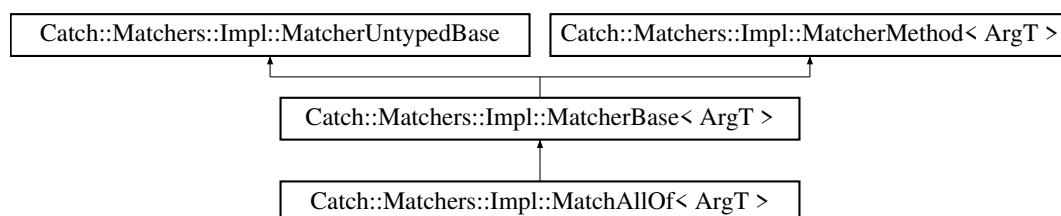
Implements [Catch::Generators::GeneratorUntypedBase](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.61 Catch::Matchers::Impl::MatchAllOf< ArgT > Struct Template Reference

Inheritance diagram for [Catch::Matchers::Impl::MatchAllOf< ArgT >](#):



Public Member Functions

- [bool match \(ArgT const &arg\) const override](#)
- [std::string describe \(\) const override](#)
- [MatchAllOf< ArgT > operator&& \(MatcherBase< ArgT > const &other\)](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< ArgT >](#)

- [MatchAllOf< ArgT > operator&& \(MatcherBase const &other\) const](#)
- [MatchAnyOf< ArgT > operator|| \(MatcherBase const &other\) const](#)
- [MatchNotOf< ArgT > operator! \(\) const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [MatcherUntypedBase](#) ([MatcherUntypedBase](#) const &)=default
- [MatcherUntypedBase](#) & **operator=** ([MatcherUntypedBase](#) const &)=delete
- std::string **toString** () const

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< ObjectT >](#)

- virtual bool **match** ([ObjectT](#) const &arg) const =0

Public Attributes

- std::vector< [MatcherBase](#)< [ArgT](#) > const * > **m_matchers**

Additional Inherited Members**Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- std::string **m_cachedToString**

5.61.1 Member Function Documentation**5.61.1.1 describe()**

```
template<typename ArgT >
std::string Catch::Matchers::Impl::MatchAllOf< ArgT >::describe ( ) const [inline], [override],
[virtual]
```

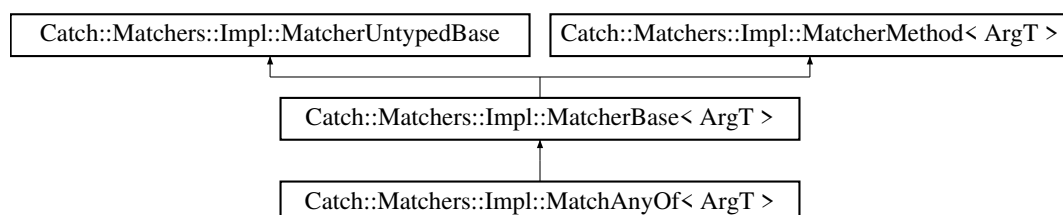
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.62 [Catch::Matchers::Impl::MatchAnyOf< ArgT >](#) Struct Template Reference

Inheritance diagram for [Catch::Matchers::Impl::MatchAnyOf< ArgT >](#):



Public Member Functions

- `bool match (ArgT const &arg) const` [override](#)
- `std::string describe ()` [const override](#)
- `MatchAnyOf< ArgT > operator|| (MatcherBase< ArgT > const &other)`

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< ArgT >](#)

- `MatchAllOf< ArgT > operator&& (MatcherBase const &other) const`
- `MatchAnyOf< ArgT > operator|| (MatcherBase const &other) const`
- `MatchNotOf< ArgT > operator! () const`

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- `MatcherUntypedBase (MatcherUntypedBase const &)=default`
- `MatcherUntypedBase & operator= (MatcherUntypedBase const &)=delete`
- `std::string toString ()` [const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< ObjectT >](#)

- `virtual bool match (ObjectT const &arg) const` `=0`

Public Attributes

- `std::vector< MatcherBase< ArgT > const * > m_matchers`

Additional Inherited Members

Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- `std::string m_cachedToString`

5.62.1 Member Function Documentation

5.62.1.1 describe()

```
template<typename ArgT >
std::string Catch::Matchers::Impl::MatchAnyOf< ArgT >::describe ( ) const [inline], [override],
[virtual]
```

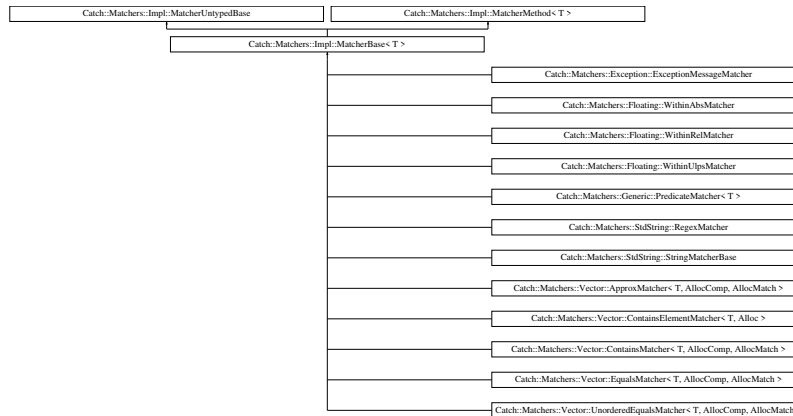
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.63 Catch::Matchers::Impl::MatcherBase< T > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatcherBase< T >:



Public Member Functions

- [MatchAllOf< T > operator&& \(MatcherBase const &other\) const](#)
- [MatchAnyOf< T > operator|| \(MatcherBase const &other\) const](#)
- [MatchNotOf< T > operator! \(\) const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [MatcherUntypedBase \(MatcherUntypedBase const &\)=default](#)
- [MatcherUntypedBase & operator= \(MatcherUntypedBase const &\)=delete](#)
- [std::string toString \(\) const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- [virtual bool match \(T const &arg\) const=0](#)

Additional Inherited Members

Protected Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [virtual std::string describe \(\) const =0](#)

Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

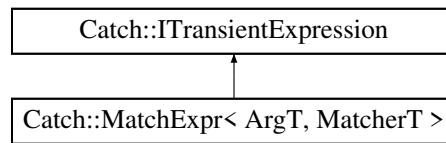
- [std::string m_cachedToString](#)

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.66 Catch::MatchExpr< ArgT, MatcherT > Class Template Reference

Inheritance diagram for Catch::MatchExpr< ArgT, MatcherT >:



Public Member Functions

- **MatchExpr** (ArgT const &arg, MatcherT const &matcher, StringRef const &matcherString)
- void streamReconstructedExpression (std::ostream &os) const override

Public Member Functions inherited from [Catch::ITransientExpression](#)

- auto isBinaryExpression () const -> bool
- auto getResult () const -> bool
- ITransientExpression (bool isBinaryExpression, bool result)

Additional Inherited Members

Public Attributes inherited from [Catch::ITransientExpression](#)

- bool m_isBinaryExpression
- bool m_result

5.66.1 Member Function Documentation

5.66.1.1 streamReconstructedExpression()

```

template<typename ArgT , typename MatcherT >
void Catch::MatchExpr< ArgT, MatcherT >::streamReconstructedExpression (
    std::ostream & os ) const [inline], [override], [virtual]
  
```

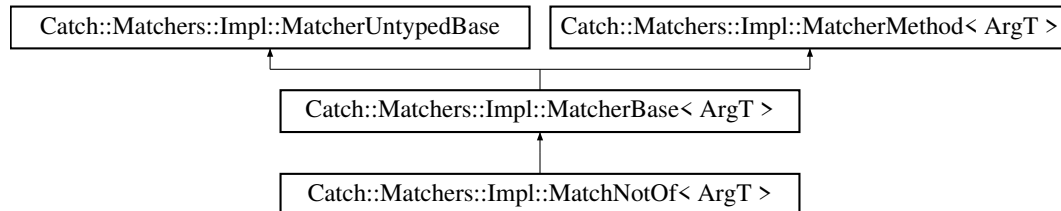
Implements [Catch::ITransientExpression](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.67 Catch::Matchers::Impl::MatchNotOf< ArgT > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatchNotOf< ArgT >:



Public Member Functions

- **MatchNotOf** ([MatcherBase< ArgT > const](#) &[underlyingMatcher](#))
- **bool match** ([ArgT const](#) &[arg](#)) [const override](#)
- [std::string describe](#) () [const override](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< ArgT >](#)

- [MatchAllOf< ArgT > operator&&](#) ([MatcherBase const](#) &[other](#)) [const](#)
- [MatchAnyOf< ArgT > operator||](#) ([MatcherBase const](#) &[other](#)) [const](#)
- [MatchNotOf< ArgT > operator!](#) () [const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** ([MatcherUntypedBase const](#) &)=[default](#)
- [MatcherUntypedBase](#) & **operator=** ([MatcherUntypedBase const](#) &)=[delete](#)
- [std::string toString](#) () [const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< ObjectT >](#)

- **virtual bool match** ([ObjectT const](#) &[arg](#)) [const](#) =0

Public Attributes

- [MatcherBase< ArgT > const](#) & [m_underlyingMatcher](#)

Additional Inherited Members

Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [std::string m_cachedToString](#)

5.67.1 Member Function Documentation

5.67.1.1 describe()

```
template<typename ArgT >
std::string Catch::Matchers::Impl::MatchNotOf< ArgT >::describe ( ) const [inline], [override],
[virtual]
```

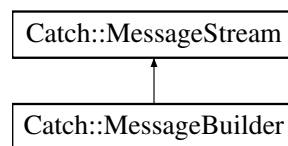
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.68 Catch::MessageBuilder Struct Reference

Inheritance diagram for `Catch::MessageBuilder`:



Public Member Functions

- **MessageBuilder** ([StringRef const](#) ¯oName, [SourceLineInfo const](#) &lineInfo, [ResultWas::OfType](#) type)
- `template<typename T >`
[MessageBuilder](#) & **operator**<< ([T const](#) &value)

Public Member Functions inherited from [Catch::MessageStream](#)

- `template<typename T >`
[MessageStream](#) & **operator**<< ([T const](#) &value)

Public Attributes

- [MessageInfo](#) m_info

Public Attributes inherited from [Catch::MessageStream](#)

- [ReusableStringStream](#) m_stream

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.69 Catch::MessageInfo Struct Reference

Public Member Functions

- **MessageInfo** ([StringRef](#) const &_macroName, [SourceLineInfo](#) const &_lineInfo, ResultWas::OfType _type)
- **bool operator==** ([MessageInfo](#) const &other) const
- **bool operator<** ([MessageInfo](#) const &other) const

Public Attributes

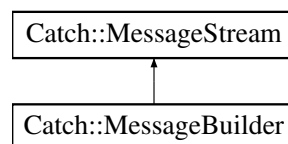
- [StringRef](#) macroName
- std::string message
- [SourceLineInfo](#) lineInfo
- ResultWas::OfType type
- [unsigned int](#) sequence

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.70 Catch::MessageStream Struct Reference

Inheritance diagram for Catch::MessageStream:



Public Member Functions

- template<typename T >
[MessageStream](#) & **operator<<** (T const &value)

Public Attributes

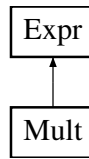
- [ReusableStringStream](#) m_stream

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.71 Mult Class Reference

Inheritance diagram for Mult:



Public Member Functions

- [Mult](#) ([Expr](#) *lhs, [Expr](#) *rhs)
- bool [equals](#) ([Expr](#) *e)
- int [interp](#) ()
- bool [has_variable](#) ()
- [Expr](#) * [subst](#) (std::string, [Expr](#) *s)
- void [print](#) (std::ostream &ot)
- void [pretty_print_dr](#) (std::ostream &ot)
- void [pretty_print](#) (std::ostream &ot, precedence_t prec)

Public Member Functions inherited from [Expr](#)

- std::string [to_string](#) ()
- std::string [to_pretty_string](#) ()

Public Attributes

- [Expr](#) * [lhs](#)
- [Expr](#) * [rhs](#)

5.71.1 Constructor & Destructor Documentation

5.71.1.1 Mult()

```

Mult::Mult (
    Expr * lhs,
    Expr * rhs )
  
```

this is the [Mult](#) function

Parameters

<i>lhs</i>	
<i>rhs</i>	

5.71.2 Member Function Documentation

5.71.2.1 equals()

```
bool Mult::equals (
    Expr * e ) [virtual]
```

Implements [Expr](#).

5.71.2.2 has_variable()

```
bool Mult::has_variable ( ) [virtual]
```

Implements [Expr](#).

5.71.2.3 interp()

```
int Mult::interp ( ) [virtual]
```

Implements [Expr](#).

5.71.2.4 pretty_print()

```
void Mult::pretty_print (
    std::ostream & ot,
    precedence_t prec ) [virtual]
```

Implements [Expr](#).

5.71.2.5 pretty_print_dr()

```
void Mult::pretty_print_dr (
    std::ostream & ot ) [virtual]
```

Implements [Expr](#).

5.71.2.6 print()

```
void Mult::print (
    std::ostream & ot ) [virtual]
```

Implements [Expr](#).

5.71.2.7 subst()

```
Expr * Mult::subst (
    std::string replace,
    Expr * s ) [virtual]
```

Implements [Expr](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/Expr.h
- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/Expr.cpp

5.72 Catch::NameAndTags Struct Reference

Public Member Functions

- **NameAndTags** ([StringRef const](#) &name_[_](#)=[StringRef\(\)](#), [StringRef const](#) &tags_[_](#)=[StringRef\(\)](#)) [noexcept](#)

Public Attributes

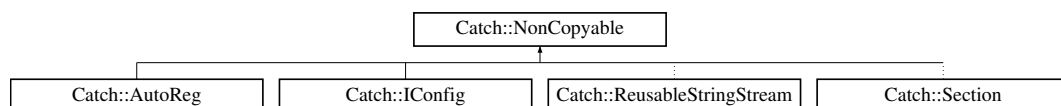
- [StringRef](#) name
- [StringRef](#) tags

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.73 Catch::NonCopyable Class Reference

Inheritance diagram for Catch::NonCopyable:

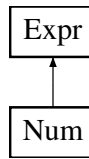


The documentation for this class was generated from the following file:

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.74 Num Class Reference

Inheritance diagram for Num:



Public Member Functions

- [Num](#) (int val)
- bool [equals](#) ([Expr](#) *e)
- int [interp](#) ()
- bool [has_variable](#) ()
- [Expr](#) * [subst](#) (std::string, [Expr](#) *s)
- void [print](#) (std::ostream &ot)
- void [pretty_print_dr](#) (std::ostream &ot)
- void [pretty_print](#) (std::ostream &ot, precedence_t prec)

Public Member Functions inherited from [Expr](#)

- std::string [to_string](#) ()
- std::string [to_pretty_string](#) ()

Public Attributes

- int [val](#)

5.74.1 Constructor & Destructor Documentation

5.74.1.1 Num()

```
Num::Num (
    int val )
```

this is the [Num](#) function

Parameters

<i>val</i>	
------------	--

5.74.2 Member Function Documentation

5.74.2.1 equals()

```
bool Num::equals (
    Expr * e ) [virtual]
```

Implements [Expr](#).

5.74.2.2 has_variable()

```
bool Num::has_variable ( ) [virtual]
```

Implements [Expr](#).

5.74.2.3 interp()

```
int Num::interp ( ) [virtual]
```

Implements [Expr](#).

5.74.2.4 pretty_print()

```
void Num::pretty_print (
    std::ostream & ot,
    precedence_t prec ) [virtual]
```

Implements [Expr](#).

5.74.2.5 pretty_print_dr()

```
void Num::pretty_print_dr (
    std::ostream & ot ) [virtual]
```

Implements [Expr](#).

5.74.2.6 print()

```
void Num::print (
    std::ostream & ot ) [virtual]
```

Implements [Expr](#).

5.74.2.7 subst()

```
Expr * Num::subst (
    std::string replace,
    Expr * s ) [virtual]
```

Implements [Expr](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/Expr.h
- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/Expr.cpp

5.75 Catch::Option< T > Class Template Reference

Public Member Functions

- [Option](#) ([T const](#) &[_value](#))
- [Option](#) ([Option const](#) &[_other](#))
- [Option](#) & [operator=](#) ([Option const](#) &[_other](#))
- [Option](#) & [operator=](#) ([T const](#) &[_value](#))
- [void reset](#) ()
- [T](#) & [operator*](#) ()
- [T const](#) & [operator*](#) () [const](#)
- [T](#) * [operator->](#) ()
- [const T](#) * [operator->](#) () [const](#)
- [T valueOr](#) ([T const](#) &[defaultValue](#)) [const](#)
- [bool some](#) () [const](#)
- [bool none](#) () [const](#)
- [bool operator!](#) () [const](#)
- [operator bool](#) () [const](#)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.76 Catch::pluralise Struct Reference

Public Member Functions

- [pluralise](#) ([std::size_t count](#), [std::string const](#) &[label](#))

Public Attributes

- [std::size_t m_count](#)
- [std::string m_label](#)

Friends

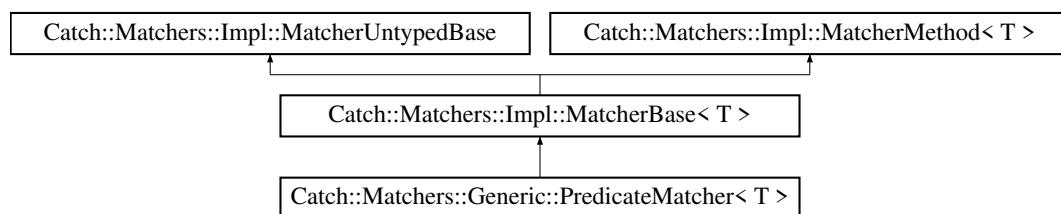
- `std::ostream & operator<< (std::ostream &os, pluralise const &pluraliser)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.77 Catch::Matchers::Generic::PredicateMatcher< T > Class Template Reference

Inheritance diagram for Catch::Matchers::Generic::PredicateMatcher< T >:

**Public Member Functions**

- **PredicateMatcher** (`std::function< bool(T const &)>` const &elem, `std::string` const &descr)
- `bool match (T const &item) const` override
- `std::string describe ()` const override

Public Member Functions inherited from `Catch::Matchers::Impl::MatcherBase< T >`

- `MatchAllOf< T > operator&& (MatcherBase const &other) const`
- `MatchAnyOf< T > operator|| (MatcherBase const &other) const`
- `MatchNotOf< T > operator! () const`

Public Member Functions inherited from `Catch::Matchers::Impl::MatcherUntypedBase`

- **MatcherUntypedBase** (`MatcherUntypedBase` const &)=default
- `MatcherUntypedBase & operator= (MatcherUntypedBase const &)=delete`
- `std::string toString ()` const

Additional Inherited Members

Protected Attributes inherited from `Catch::Matchers::Impl::MatcherUntypedBase`

- `std::string m_cachedToString`

5.77.1 Member Function Documentation

5.77.1.1 describe()

```
template<typename T >
std::string Catch::Matchers::Generic::PredicateMatcher< T >::describe ( ) const [inline],
[override], [virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

5.77.1.2 match()

```
template<typename T >
bool Catch::Matchers::Generic::PredicateMatcher< T >::match (
    T const & item ) const [inline], [override], [virtual]
```

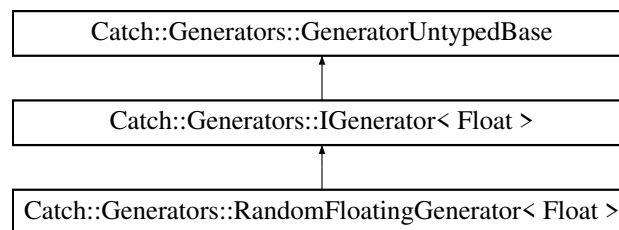
Implements [Catch::Matchers::Impl::MatcherMethod< T >](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.78 Catch::Generators::RandomFloatingGenerator< Float > Class Template Reference

Inheritance diagram for [Catch::Generators::RandomFloatingGenerator< Float >](#):



Public Member Functions

- **RandomFloatingGenerator** ([Float a](#), [Float b](#))
- [Float const & get \(\)](#) *const override*
- [bool next \(\)](#) *override*

Additional Inherited Members

Public Types inherited from [Catch::Generators::IGenerator< Float >](#)

- [using type](#)

5.78.1 Member Function Documentation

5.78.1.1 get()

```
template<typename Float >
Float const & Catch::Generators::RandomFloatingGenerator< Float >::get ( ) const [inline],
[override], [virtual]
```

Implements [Catch::Generators::IGenerator< Float >](#).

5.78.1.2 next()

```
template<typename Float >
bool Catch::Generators::RandomFloatingGenerator< Float >::next ( ) [inline], [override],
[virtual]
```

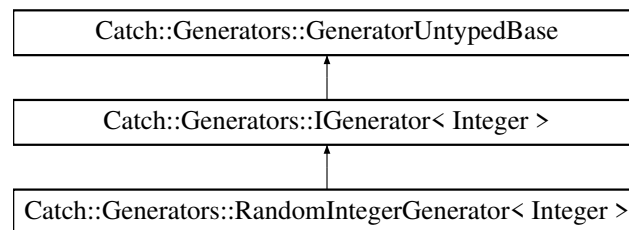
Implements [Catch::Generators::GeneratorUntypedBase](#).

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.79 Catch::Generators::RandomIntegerGenerator< Integer > Class Template Reference

Inheritance diagram for [Catch::Generators::RandomIntegerGenerator< Integer >](#):



Public Member Functions

- [RandomIntegerGenerator \(Integer a, Integer b\)](#)
- [Integer const & get \(\) const override](#)
- [bool next \(\) override](#)

Additional Inherited Members

Public Types inherited from [Catch::Generators::IGenerator< Integer >](#)

- [using type](#)

5.79.1 Member Function Documentation

5.79.1.1 get()

```
template<typename Integer >
Integer const & Catch::Generators::RandomIntegerGenerator< Integer >::get ( ) const [inline],
[override], [virtual]
```

Implements [Catch::Generators::IGenerator< Integer >](#).

5.79.1.2 next()

```
template<typename Integer >
bool Catch::Generators::RandomIntegerGenerator< Integer >::next ( ) [inline], [override],
[virtual]
```

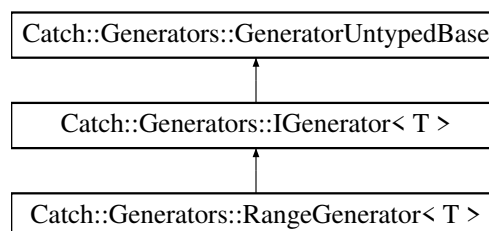
Implements [Catch::Generators::GeneratorUntypedBase](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.80 Catch::Generators::RangeGenerator< T > Class Template Reference

Inheritance diagram for [Catch::Generators::RangeGenerator< T >](#):



Public Member Functions

- **RangeGenerator** ([T const](#) &start, [T const](#) &end, [T const](#) &step)
- **RangeGenerator** ([T const](#) &start, [T const](#) &end)
- [T const](#) & [get \(\)](#) [const override](#)
- [bool](#) [next \(\)](#) [override](#)

Additional Inherited Members

Public Types inherited from [Catch::Generators::IGenerator< T >](#)

- [using](#) [type](#) = [T](#)

5.80.1 Member Function Documentation

5.80.1.1 get()

```
template<typename T >
T const & Catch::Generators::RangeGenerator< T >::get ( ) const [inline], [override], [virtual]
```

Implements [Catch::Generators::IGenerator< T >](#).

5.80.1.2 next()

```
template<typename T >
bool Catch::Generators::RangeGenerator< T >::next ( ) [inline], [override], [virtual]
```

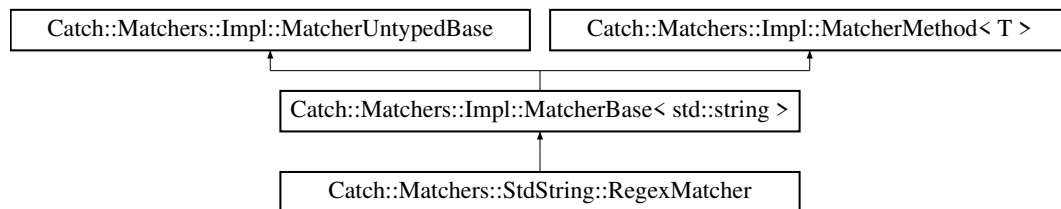
Implements [Catch::Generators::GeneratorUntypedBase](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.81 Catch::Matchers::StdString::RegexMatcher Struct Reference

Inheritance diagram for [Catch::Matchers::StdString::RegexMatcher](#):



Public Member Functions

- **RegexMatcher** (std::string [regex](#), CaseSensitive::Choice [caseSensitivity](#))
- **bool match** (std::string [const &matchee](#)) [const override](#)
- std::string **describe** () [const override](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- **MatchAllOf**< T > **operator&&** ([MatcherBase const &other](#)) [const](#)
- **MatchAnyOf**< T > **operator||** ([MatcherBase const &other](#)) [const](#)
- **MatchNotOf**< T > **operator!** () [const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** ([MatcherUntypedBase const &](#))=default
- **MatcherUntypedBase & operator=** ([MatcherUntypedBase const &](#))=delete
- std::string **toString** () [const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- [virtual bool match](#) ([T const](#) &arg) const=0

Additional Inherited Members

Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- `std::string m_cachedToString`

5.81.1 Member Function Documentation

5.81.1.1 `describe()`

```
std::string Catch::Matchers::StdString::RegexMatcher::describe ( ) const [override], [virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.82 [Catch::RegistrarForTagAliases](#) Struct Reference

Public Member Functions

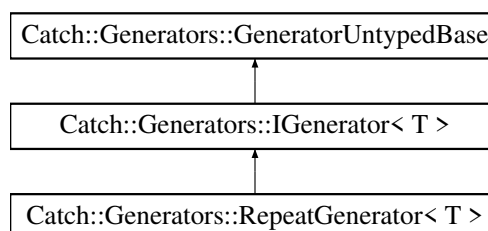
- **`RegistrarForTagAliases`** ([char const](#) *alias, [char const](#) *tag, [SourceLineInfo const](#) &lineInfo)

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.83 [Catch::Generators::RepeatGenerator< T >](#) Class Template Reference

Inheritance diagram for [Catch::Generators::RepeatGenerator< T >](#):



Public Member Functions

- **RepeatGenerator** ([size_t repeats](#), [GeneratorWrapper< T > &&generator](#))
- [T const & get \(\)](#) [const override](#)
- [bool next \(\)](#) [override](#)

Additional Inherited Members

Public Types inherited from [Catch::Generators::IGenerator< T >](#)

- [using type = T](#)

5.83.1 Member Function Documentation

5.83.1.1 get()

```
template<typename T >
T const & Catch::Generators::RepeatGenerator< T >::get ( ) const [inline], [override], [virtual]
```

Implements [Catch::Generators::IGenerator< T >](#).

5.83.1.2 next()

```
template<typename T >
bool Catch::Generators::RepeatGenerator< T >::next ( ) [inline], [override], [virtual]
```

Implements [Catch::Generators::GeneratorUntypedBase](#).

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.84 Catch::ResultDisposition Struct Reference

Public Types

- enum **Flags** { **Normal** = 0x01 , **ContinueOnFailure** = 0x02 , **FalseTest** = 0x04 , **SuppressFail** = 0x08 }

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.85 Catch::ResultWas Struct Reference

Public Types

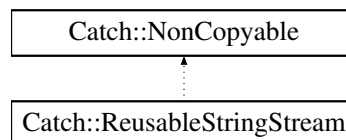
- enum **OfType** {
Unknown = -1 , **Ok** = 0 , **Info** = 1 , **Warning** = 2 ,
FailureBit = 0x10 , **ExpressionFailed** = FailureBit | 1 , **ExplicitFailure** = FailureBit | 2 , **Exception** = 0x100
| FailureBit ,
ThrowException = Exception | 1 , **DidntThrowException** = Exception | 2 , **FatalErrorCondition** = 0x200 |
FailureBit }

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.86 Catch::ReusableStringStream Class Reference

Inheritance diagram for Catch::ReusableStringStream:



Public Member Functions

- auto** **str** () **const** -> std::string
- template<typename T >
auto **operator**<< (T **const** &value) -> **ReusableStringStream** &
- auto** **get** () -> std::ostream &

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.87 Catch::RunTests Struct Reference

Public Types

- enum **InWhatOrder** { **InDeclarationOrder** , **InLexicographicalOrder** , **InRandomOrder** }

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.88 Catch::ScopedMessage Class Reference

Public Member Functions

- **ScopedMessage** ([MessageBuilder const](#) &builder)
- **ScopedMessage** ([ScopedMessage](#) &duplicate)=delete
- **ScopedMessage** ([ScopedMessage](#) &&old)

Public Attributes

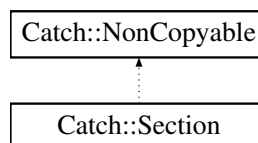
- [MessageInfo](#) m_info
- [bool](#) m_moved

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.89 Catch::Section Class Reference

Inheritance diagram for Catch::Section:



Public Member Functions

- **Section** ([SectionInfo const](#) &info)
- **operator bool** () [const](#)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.90 Catch::SectionEndInfo Struct Reference

Public Attributes

- [SectionInfo](#) sectionInfo
- [Counts](#) prevAssertions
- [double](#) durationInSeconds

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.91 Catch::SectionInfo Struct Reference

Public Member Functions

- **SectionInfo** ([SourceLineInfo](#) const &_lineInfo, std::string const &_name)
- **SectionInfo** ([SourceLineInfo](#) const &_lineInfo, std::string const &_name, std::string const &)

Public Attributes

- std::string **name**
- std::string **description**
- [SourceLineInfo](#) **lineInfo**

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.92 Catch::ShowDurations Struct Reference

Public Types

- enum **OrNot** { **DefaultForReporter** , **Always** , **Never** }

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.93 Catch::SimplePcg32 Class Reference

Public Types

- [using](#) **result_type** = std::uint32_t

Public Member Functions

- **SimplePcg32** (result_type [seed](#)_)
- [void](#) **seed** (result_type [seed](#)_)
- [void](#) **discard** ([uint64_t](#) skip)
- result_type **operator()** ()

Static Public Member Functions

- [static constexpr](#) result_type **min** ()
- [static constexpr](#) result_type **max** ()

Friends

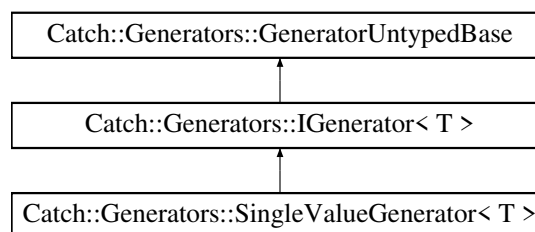
- `bool operator==` ([SimplePcg32 const](#) &lhs, [SimplePcg32 const](#) &rhs)
- `bool operator!=` ([SimplePcg32 const](#) &lhs, [SimplePcg32 const](#) &rhs)

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.94 Catch::Generators::SingleValueGenerator< T > Class Template Reference

Inheritance diagram for `Catch::Generators::SingleValueGenerator< T >`:

**Public Member Functions**

- `SingleValueGenerator` ([T](#) &&value)
- `T const` & `get` () `const override`
- `bool next` () `override`

Additional Inherited Members

Public Types inherited from [Catch::Generators::IGenerator< T >](#)

- `using type = T`

5.94.1 Member Function Documentation

5.94.1.1 `get()`

```

template<typename T >
T const & Catch::Generators::SingleValueGenerator< T >::get ( ) const [inline], [override],
[virtual]

```

Implements [Catch::Generators::IGenerator< T >](#).

5.94.1.2 next()

```
template<typename T >
bool Catch::Generators::SingleValueGenerator< T >::next ( ) [inline], [override], [virtual]
```

Implements [Catch::Generators::GeneratorUntypedBase](#).

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.95 Catch::SourceLineInfo Struct Reference

Public Member Functions

- **SourceLineInfo** (`char const * _file`, `std::size_t _line`) `noexcept`
- **SourceLineInfo** (`SourceLineInfo const &other`)=`default`
- **SourceLineInfo & operator=** (`SourceLineInfo const &`)=`default`
- **SourceLineInfo** (`SourceLineInfo &&`) `noexcept=default`
- **SourceLineInfo & operator=** (`SourceLineInfo &&`) `noexcept=default`
- **bool empty** () `const noexcept`
- **bool operator==** (`SourceLineInfo const &other`) `const noexcept`
- **bool operator<** (`SourceLineInfo const &other`) `const noexcept`

Public Attributes

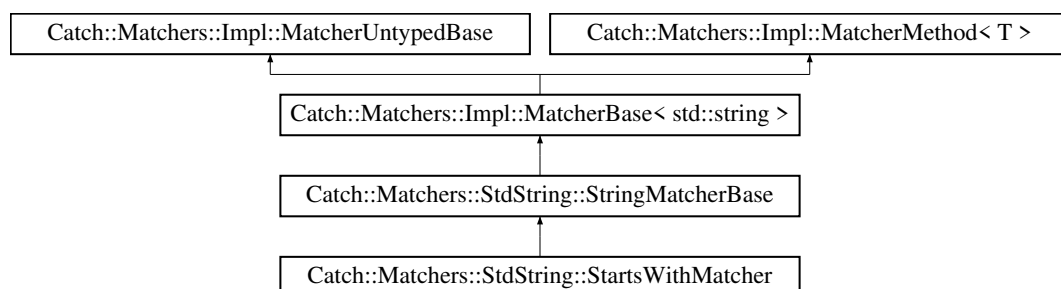
- `char const * file`
- `std::size_t line`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.96 Catch::Matchers::StdString::StartsWithMatcher Struct Reference

Inheritance diagram for `Catch::Matchers::StdString::StartsWithMatcher`:



Public Member Functions

- **StartsWithMatcher** ([CasedString](#) const &comparator)
- **bool match** (std::string const &source) const override

Public Member Functions inherited from [Catch::Matchers::StdString::StringMatcherBase](#)

- **StringMatcherBase** (std::string const &operation, [CasedString](#) const &comparator)
- std::string **describe** () const override

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- **MatchAllOf**< T > **operator&&** ([MatcherBase](#) const &other) const
- **MatchAnyOf**< T > **operator||** ([MatcherBase](#) const &other) const
- **MatchNotOf**< T > **operator!** () const

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** ([MatcherUntypedBase](#) const &)=default
- **MatcherUntypedBase** & **operator=** ([MatcherUntypedBase](#) const &)=delete
- std::string **toString** () const

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- **virtual bool match** (T const &arg) const=0

Additional Inherited Members

Public Attributes inherited from [Catch::Matchers::StdString::StringMatcherBase](#)

- [CasedString](#) m_comparator
- std::string m_operation

Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- std::string m_cachedToString

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.97 Catch::StreamEndStop Struct Reference

Public Member Functions

- `std::string operator+ () const`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.98 Catch::StringMaker< T, typename > Struct Template Reference

Static Public Member Functions

- `template<typename Fake = T>
static std::enable_if<::Catch::Detail::IsStreamInsertable< Fake >::value, std::string >::type convert (const Fake &value)`
- `template<typename Fake = T>
static std::enable_if<!::Catch::Detail::IsStreamInsertable< Fake >::value, std::string >::type convert (const Fake &value)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.99 Catch::StringMaker< bool > Struct Reference

Static Public Member Functions

- `static std::string convert (bool b)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.100 Catch::StringMaker< Catch::Detail::Approx > Struct Reference

Static Public Member Functions

- `static std::string convert (Catch::Detail::Approx const &value)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.101 Catch::StringMaker< char * > Struct Reference

Static Public Member Functions

- `static std::string convert (char *str)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.102 Catch::StringMaker< char > Struct Reference

Static Public Member Functions

- `static std::string convert (char c)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.103 Catch::StringMaker< char const * > Struct Reference

Static Public Member Functions

- `static std::string convert (char const *str)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.104 Catch::StringMaker< char[SZ]> Struct Template Reference

Static Public Member Functions

- `static std::string convert (char const *str)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.105 Catch::StringMaker< double > Struct Reference

Static Public Member Functions

- [static](#) std::string **convert** ([double](#) value)

Static Public Attributes

- [static int](#) **precision**

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.106 Catch::StringMaker< float > Struct Reference

Static Public Member Functions

- [static](#) std::string **convert** ([float](#) value)

Static Public Attributes

- [static int](#) **precision**

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.107 Catch::StringMaker< int > Struct Reference

Static Public Member Functions

- [static](#) std::string **convert** ([int](#) value)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.108 Catch::StringMaker< long > Struct Reference

Static Public Member Functions

- [static](#) std::string **convert** ([long](#) value)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.109 Catch::StringMaker< long long > Struct Reference

Static Public Member Functions

- `static std::string convert (long long value)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.110 Catch::StringMaker< R C::* > Struct Template Reference

Static Public Member Functions

- `static std::string convert (R C::*p)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.111 Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!Catch::Detail::IsStreamInsertable< R >::value >::type > Struct Template Reference

Static Public Member Functions

- `static std::string convert (R const &range)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.112 Catch::StringMaker< signed char > Struct Reference

Static Public Member Functions

- `static std::string convert (signed char c)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.113 Catch::StringMaker< signed char[SZ]> Struct Template Reference

Static Public Member Functions

- [static](#) std::string **convert** ([signed char const](#) *str)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.114 Catch::StringMaker< std::nullptr_t > Struct Reference

Static Public Member Functions

- [static](#) std::string **convert** (std::nullptr_t)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.115 Catch::StringMaker< std::string > Struct Reference

Static Public Member Functions

- [static](#) std::string **convert** ([const](#) std::string &str)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.116 Catch::StringMaker< std::wstring > Struct Reference

Static Public Member Functions

- [static](#) std::string **convert** ([const](#) std::wstring &[wstr](#))

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.117 Catch::StringMaker< T * > Struct Template Reference

Static Public Member Functions

- `template<typename U >
static std::string convert (U *p)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.118 Catch::StringMaker< T[SZ]> Struct Template Reference

Static Public Member Functions

- `static std::string convert (T const(&arr)[SZ])`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.119 Catch::StringMaker< unsigned char > Struct Reference

Static Public Member Functions

- `static std::string convert (unsigned char c)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.120 Catch::StringMaker< unsigned char[SZ]> Struct Template Reference

Static Public Member Functions

- `static std::string convert (unsigned char const *str)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.121 Catch::StringMaker< unsigned int > Struct Reference

Static Public Member Functions

- [static](#) std::string **convert** ([unsigned int](#) value)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.122 Catch::StringMaker< unsigned long > Struct Reference

Static Public Member Functions

- [static](#) std::string **convert** ([unsigned long](#) value)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.123 Catch::StringMaker< unsigned long long > Struct Reference

Static Public Member Functions

- [static](#) std::string **convert** ([unsigned long long](#) value)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.124 Catch::StringMaker< wchar_t * > Struct Reference

Static Public Member Functions

- [static](#) std::string **convert** ([wchar_t](#) *str)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.125 Catch::StringMaker< wchar_t const * > Struct Reference

Static Public Member Functions

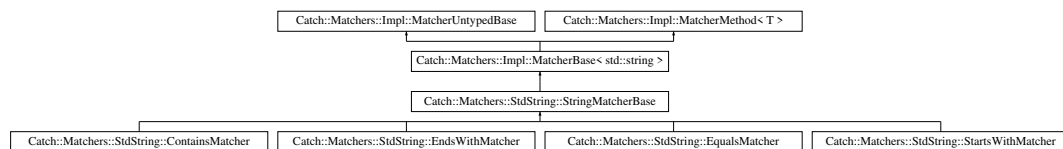
- [static](#) `std::string convert (wchar_t const *str)`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.126 Catch::Matchers::StdString::StringMatcherBase Struct Reference

Inheritance diagram for Catch::Matchers::StdString::StringMatcherBase:



Public Member Functions

- **StringMatcherBase** (`std::string const &operation`, `CasedString const &comparator`)
- `std::string describe () const override`

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- [MatchAllOf< T > operator&& \(MatcherBase const &other\) const](#)
- [MatchAnyOf< T > operator|| \(MatcherBase const &other\) const](#)
- [MatchNotOf< T > operator! \(\) const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** (`MatcherUntypedBase const &`)=default
- [MatcherUntypedBase & operator= \(MatcherUntypedBase const &\)=delete](#)
- `std::string toString () const`

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- `virtual bool match (T const &arg) const=0`

Public Attributes

- [CasedString](#) `m_comparator`
- `std::string m_operation`

Additional Inherited Members

Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- `std::string m_cachedToString`

5.126.1 Member Function Documentation

5.126.1.1 `describe()`

```
std::string Catch::Matchers::StdString::StringMatcherBase::describe ( ) const [override],
[virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.127 `Catch::StringRef` Class Reference

```
#include <catch.hpp>
```

Public Types

- `using size_type = std::size_t`
- `using const_iterator = const char*`

Public Member Functions

- `StringRef (char const *rawChars) noexcept`
- `constexpr StringRef (char const *rawChars, size_type size) noexcept`
- `StringRef (std::string const &stdString) noexcept`
- `operator std::string () const`
- `auto operator== (StringRef const &other) const noexcept -> bool`
- `auto operator!= (StringRef const &other) const noexcept -> bool`
- `auto operator[] (size_type index) const noexcept -> char`
- `constexpr auto empty () const noexcept -> bool`
- `constexpr auto size () const noexcept -> size_type`
- `auto c_str () const -> char const *`
- `auto substr (size_type start, size_type length) const noexcept -> StringRef`
- `auto data () const noexcept -> char const *`
- `constexpr auto isNullTerminated () const noexcept -> bool`
- `constexpr const_iterator begin () const`
- `constexpr const_iterator end () const`

5.127.1 Detailed Description

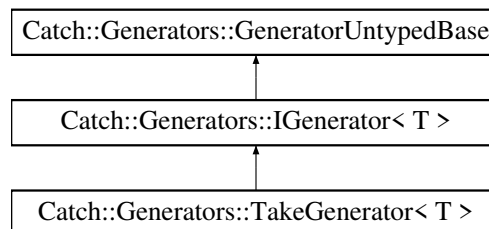
A non-owning string class (similar to the forthcoming `std::string_view`) Note that, because a [StringRef](#) may be a substring of another string, it may not be null terminated.

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.128 Catch::Generators::TakeGenerator< T > Class Template Reference

Inheritance diagram for `Catch::Generators::TakeGenerator< T >`:



Public Member Functions

- **TakeGenerator** (`size_t` target, `GeneratorWrapper< T >` &&generator)
- `T` const & `get ()` const override
- `bool` `next ()` override

Additional Inherited Members

Public Types inherited from `Catch::Generators::IGenerator< T >`

- `using` `type` = `T`

5.128.1 Member Function Documentation

5.128.1.1 `get()`

```

template<typename T>
T const & Catch::Generators::TakeGenerator< T >::get () const [inline], [override], [virtual]

```

Implements `Catch::Generators::IGenerator< T >`.

5.128.1.2 next()

```
template<typename T >
bool Catch::Generators::TakeGenerator< T >::next ( ) [inline], [override], [virtual]
```

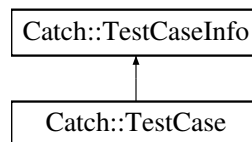
Implements [Catch::Generators::GeneratorUntypedBase](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.129 Catch::TestCase Class Reference

Inheritance diagram for [Catch::TestCase](#):



Public Member Functions

- [TestCase](#) ([ITestInvoker](#) *[testCase](#), [TestCaseInfo](#) &&[info](#))
- [TestCase](#) [WithName](#) ([std::string](#) [const](#) &[_newName](#)) [const](#)
- [void](#) [invoke](#) () [const](#)
- [TestCaseInfo](#) [const](#) & [getTestCaseInfo](#) () [const](#)
- [bool](#) [operator==](#) ([TestCase](#) [const](#) &[other](#)) [const](#)
- [bool](#) [operator<](#) ([TestCase](#) [const](#) &[other](#)) [const](#)

Public Member Functions inherited from [Catch::TestCaseInfo](#)

- [TestCaseInfo](#) ([std::string](#) [const](#) &[_name](#), [std::string](#) [const](#) &[_className](#), [std::string](#) [const](#) &[_description](#), [std::vector](#)< [std::string](#) > [const](#) &[_tags](#), [SourceLineInfo](#) [const](#) &[_lineInfo](#))
- [bool](#) [isHidden](#) () [const](#)
- [bool](#) [throws](#) () [const](#)
- [bool](#) [okToFail](#) () [const](#)
- [bool](#) [expectedToFail](#) () [const](#)
- [std::string](#) [tagsAsString](#) () [const](#)

Additional Inherited Members

Public Types inherited from [Catch::TestCaseInfo](#)

- enum [SpecialProperties](#) {
None = 0 , **IsHidden** = 1 << 1 , **ShouldFail** = 1 << 2 , **MayFail** = 1 << 3 ,
Throws = 1 << 4 , **NonPortable** = 1 << 5 , **Benchmark** = 1 << 6 }

Public Attributes inherited from [Catch::TestCaseInfo](#)

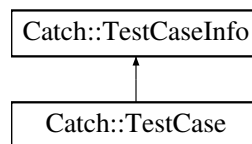
- std::string **name**
- std::string **className**
- std::string **description**
- std::vector< std::string > **tags**
- std::vector< std::string > **lcaseTags**
- [SourceLineInfo](#) **lineInfo**
- SpecialProperties **properties**

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.130 Catch::TestCaseInfo Struct Reference

Inheritance diagram for Catch::TestCaseInfo:



Public Types

- enum **SpecialProperties** {
None = 0 , **IsHidden** = 1 << 1 , **ShouldFail** = 1 << 2 , **MayFail** = 1 << 3 ,
Throws = 1 << 4 , **NonPortable** = 1 << 5 , **Benchmark** = 1 << 6 }

Public Member Functions

- **TestCaseInfo** (std::string [const](#) &_name, std::string [const](#) &_className, std::string [const](#) &_description, std::vector< std::string > [const](#) &_tags, [SourceLineInfo](#) [const](#) &_lineInfo)
- [bool](#) **isHidden** () [const](#)
- [bool](#) **throws** () [const](#)
- [bool](#) **okToFail** () [const](#)
- [bool](#) **expectedToFail** () [const](#)
- std::string **tagsAsString** () [const](#)

Public Attributes

- std::string **name**
- std::string **className**
- std::string **description**
- std::vector< std::string > **tags**
- std::vector< std::string > **lcaseTags**
- [SourceLineInfo](#) **lineInfo**
- SpecialProperties **properties**

Friends

- **void setTags** ([TestCaseInfo](#) &[testCaseInfo](#), std::vector< std::string > tags)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

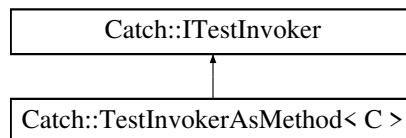
5.131 Catch::TestFailureException Struct Reference

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.132 Catch::TestInvokerAsMethod< C > Class Template Reference

Inheritance diagram for Catch::TestInvokerAsMethod< C >:

**Public Member Functions**

- **TestInvokerAsMethod** ([void](#)(C::*[testAsMethod](#)()) **noexcept**)
- [void invoke](#) () **const override**

5.132.1 Member Function Documentation**5.132.1.1 invoke()**

```

template<typename C >
void Catch::TestInvokerAsMethod< C >::invoke ( ) const [inline], [override], [virtual]

```

Implements [Catch::ITestInvoker](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.133 Catch::Timer Class Reference

Public Member Functions

- `void start ()`
- `auto getElapsedNanoseconds () const -> uint64_t`
- `auto getElapsedMicroseconds () const -> uint64_t`
- `auto getElapsedMilliseconds () const -> unsigned int`
- `auto getElapsedSeconds () const -> double`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.134 Catch::Totals Struct Reference

Public Member Functions

- `Totals operator- (Totals const &other) const`
- `Totals & operator+= (Totals const &other)`
- `Totals delta (Totals const &prevTotals) const`

Public Attributes

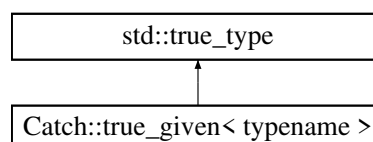
- `int error = 0`
- `Counts assertions`
- `Counts testCases`

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.135 Catch::true_given< typename > Struct Template Reference

Inheritance diagram for Catch::true_given< typename >:

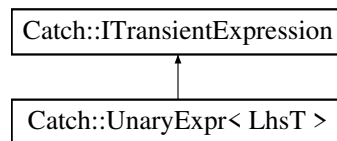


The documentation for this struct was generated from the following file:

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.136 Catch::UnaryExpr< LhsT > Class Template Reference

Inheritance diagram for Catch::UnaryExpr< LhsT >:



Public Member Functions

- **UnaryExpr** ([LhsT lhs](#))

Public Member Functions inherited from [Catch::ITransientExpression](#)

- **auto isBinaryExpression** () **const** -> [bool](#)
- **auto getResult** () **const** -> [bool](#)
- **ITransientExpression** ([bool isBinaryExpression](#), [bool result](#))

Additional Inherited Members

Public Attributes inherited from [Catch::ITransientExpression](#)

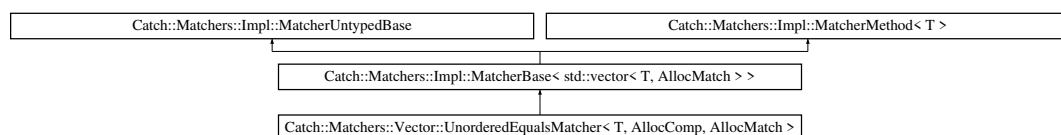
- [bool m_isBinaryExpression](#)
- [bool m_result](#)

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.137 Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch > Struct Template Reference

Inheritance diagram for Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >:



Public Member Functions

- **UnorderedEqualsMatcher** (std::vector< [T](#), [AllocComp](#) > **const** &target)
- **bool match** (std::vector< [T](#), [AllocMatch](#) > **const** &vec) **const** override
- std::string **describe** () **const** override

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- [MatchAllOf< T > operator&& \(MatcherBase const &other\) const](#)
- [MatchAnyOf< T > operator|| \(MatcherBase const &other\) const](#)
- [MatchNotOf< T > operator! \(\) const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [MatcherUntypedBase \(MatcherUntypedBase const &\)=default](#)
- [MatcherUntypedBase & operator= \(MatcherUntypedBase const &\)=delete](#)
- [std::string toString \(\) const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- [virtual bool match \(T const &arg\) const=0](#)

Additional Inherited Members**Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- [std::string m_cachedToString](#)

5.137.1 Member Function Documentation**5.137.1.1 describe()**

```
template<typename T , typename AllocComp , typename AllocMatch >
std::string Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >↔
::describe ( ) const [inline], [override], [virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.138 Catch::UseColour Struct Reference**Public Types**

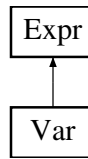
- enum [YesOrNo { Auto , Yes , No }](#)

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

- [/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp](#)

5.139 Var Class Reference

Inheritance diagram for Var:



Public Member Functions

- [Var](#) (std::string val)
- bool [equals](#) ([Expr](#) *e)
- int [interp](#) ()
- bool [has_variable](#) ()
- [Expr](#) * [subst](#) (std::string, [Expr](#) *s)
- void [print](#) (std::ostream &ot)
- void [pretty_print_dr](#) (std::ostream &ot)
- void [pretty_print](#) (std::ostream &ot, precedence_t prec)

Public Member Functions inherited from [Expr](#)

- std::string [to_string](#) ()
- std::string [to_pretty_string](#) ()

Public Attributes

- std::string [val](#)

5.139.1 Constructor & Destructor Documentation

5.139.1.1 Var()

```
Var::Var (
    std::string val )
```

this is the value function

Parameters

<i>val</i>	
------------	--

5.139.2 Member Function Documentation

5.139.2.1 equals()

```
bool Var::equals (
    Expr * e ) [virtual]
```

Implements [Expr](#).

5.139.2.2 has_variable()

```
bool Var::has_variable ( ) [virtual]
```

Implements [Expr](#).

5.139.2.3 interp()

```
int Var::interp ( ) [virtual]
```

Implements [Expr](#).

5.139.2.4 pretty_print()

```
void Var::pretty_print (
    std::ostream & ot,
    precedence_t prec ) [virtual]
```

Implements [Expr](#).

5.139.2.5 pretty_print_dr()

```
void Var::pretty_print_dr (
    std::ostream & ot ) [virtual]
```

Implements [Expr](#).

5.139.2.6 print()

```
void Var::print (
    std::ostream & ot ) [virtual]
```

Implements [Expr](#).

5.139.2.7 subst()

```
Expr * Var::subst (
    std::string replace,
    Expr * s ) [virtual]
```

Implements [Expr](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/Expr.h
- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/Expr.cpp

5.140 Catch::detail::void_type<... > Struct Template Reference

Public Types

- [using type](#) = [void](#)

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.141 Catch::WaitForKeypress Struct Reference

Public Types

- enum **When** { **Never** , **BeforeStart** = 1 , **BeforeExit** = 2 , **BeforeStartAndExit** = BeforeStart | BeforeExit }

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.142 Catch::WarnAbout Struct Reference

Public Types

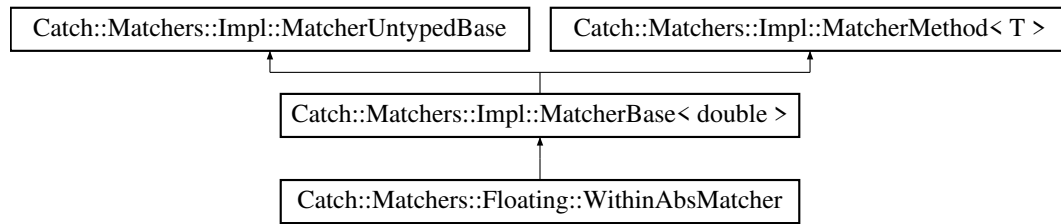
- enum **What** { **Nothing** = 0x00 , **NoAssertions** = 0x01 , **NoTests** = 0x02 }

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.143 Catch::Matchers::Floating::WithinAbsMatcher Struct Reference

Inheritance diagram for Catch::Matchers::Floating::WithinAbsMatcher:



Public Member Functions

- **WithinAbsMatcher** ([double target](#), [double margin](#))
- **bool match** ([double const &matchee](#)) [const override](#)
- **std::string describe** () [const override](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- **MatchAllOf**< T > **operator&&** ([MatcherBase const &other](#)) [const](#)
- **MatchAnyOf**< T > **operator||** ([MatcherBase const &other](#)) [const](#)
- **MatchNotOf**< T > **operator!** () [const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** ([MatcherUntypedBase const &](#))=default
- **MatcherUntypedBase & operator=** ([MatcherUntypedBase const &](#))=delete
- **std::string toString** () [const](#)

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- **virtual bool match** ([T const &arg](#)) [const=0](#)

Additional Inherited Members

Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **std::string m_cachedToString**

5.143.1 Member Function Documentation

5.143.1.1 describe()

```
std::string Catch::Matchers::Floating::WithinAbsMatcher::describe ( ) const [override], [virtual]
```

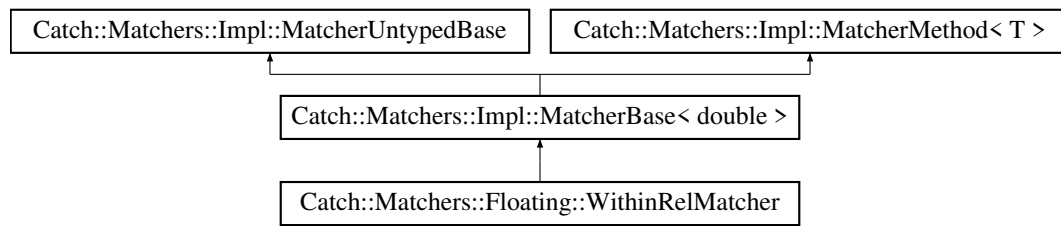
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- `/Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp`

5.144 Catch::Matchers::Floating::WithinRelMatcher Struct Reference

Inheritance diagram for Catch::Matchers::Floating::WithinRelMatcher:



Public Member Functions

- **WithinRelMatcher** (double target, double epsilon)
- **bool match** (double const &matchee) const override
- std::string **describe** () const override

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- **MatchAllOf**< T > **operator&&** (MatcherBase const &other) const
- **MatchAnyOf**< T > **operator||** (MatcherBase const &other) const
- **MatchNotOf**< T > **operator!** () const

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- **MatcherUntypedBase & operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- **virtual bool match** (T const &arg) const=0

Additional Inherited Members

Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- std::string **m_cachedToString**

5.144.1 Member Function Documentation

5.144.1.1 describe()

```
std::string Catch::Matchers::Floating::WithinRelMatcher::describe ( ) const [override], [virtual]
```

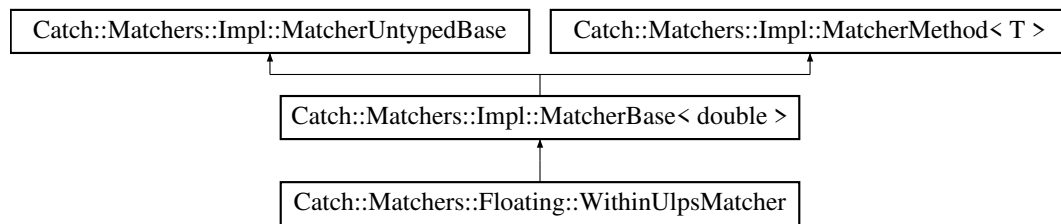
Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

5.145 Catch::Matchers::Floating::WithinUlpMatcher Struct Reference

Inheritance diagram for Catch::Matchers::Floating::WithinUlpMatcher:



Public Member Functions

- **WithinUlpMatcher** (double target, uint64_t ulps, FloatingPointKind baseType)
- **bool match** (double const &matchee) const override
- **std::string describe** () const override

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherBase< T >](#)

- **MatchAllOf**< T > **operator&&** (MatcherBase const &other) const
- **MatchAnyOf**< T > **operator||** (MatcherBase const &other) const
- **MatchNotOf**< T > **operator!** () const

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- **MatcherUntypedBase & operator=** (MatcherUntypedBase const &)=delete
- **std::string toString** () const

Public Member Functions inherited from [Catch::Matchers::Impl::MatcherMethod< T >](#)

- **virtual bool match** (T const &arg) const=0

Additional Inherited Members

Protected Attributes inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- **std::string m_cachedToString**

5.145.1 Member Function Documentation

5.145.1.1 describe()

```
std::string Catch::Matchers::Floating::WithinUlpMatcher::describe ( ) const [override],
[virtual]
```

Implements [Catch::Matchers::Impl::MatcherUntypedBase](#).

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

- /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

Chapter 6

File Documentation

6.1 /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/catch.hpp

```
00001 /*
00002  * Catch v2.13.7
00003  * Generated: 2021-07-28 20:29:27.753164
00004  * -----
00005  * This file has been merged from multiple headers. Please don't edit it directly
00006  * Copyright (c) 2021 Two Blue Cubes Ltd. All rights reserved.
00007  *
00008  * Distributed under the Boost Software License, Version 1.0. (See accompanying
00009  * file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
00010  */
00011 #ifndef TWOBLUECUBES_SINGLE_INCLUDE_CATCH_HPP_INCLUDED
00012 #define TWOBLUECUBES_SINGLE_INCLUDE_CATCH_HPP_INCLUDED
00013 // start catch.hpp
00014
00015
00016 #define CATCH_VERSION_MAJOR 2
00017 #define CATCH_VERSION_MINOR 13
00018 #define CATCH_VERSION_PATCH 7
00019
00020 #ifdef __clang__
00021 # pragma clang system_header
00022 #elif defined __GNUC__
00023 # pragma GCC system_header
00024 #endif
00025
00026 // start catch_suppress_warnings.h
00027
00028 #ifdef __clang__
00029 # if __ICC // icpc defines the __clang__ macro
00030 # pragma warning(push)
00031 # pragma warning(disable: 161 1682)
00032 # else // __ICC
00033 # pragma clang diagnostic push
00034 # pragma clang diagnostic ignored "-Wpadded"
00035 # pragma clang diagnostic ignored "-Wswitch-enum"
00036 # pragma clang diagnostic ignored "-Wcovered-switch-default"
00037 # endif
00038 #elif defined __GNUC__
00039 // Because REQUIREs trigger GCC's -Wparentheses, and because still
00040 // supported version of g++ have only buggy support for _Pragmas,
00041 // Wparentheses have to be suppressed globally.
00042 # pragma GCC diagnostic ignored "-Wparentheses" // See #674 for details
00043
00044 # pragma GCC diagnostic push
00045 # pragma GCC diagnostic ignored "-Wunused-variable"
00046 # pragma GCC diagnostic ignored "-Wpadded"
00047 #endif
00048 // end catch_suppress_warnings.h
00049 #if defined(CATCH_CONFIG_MAIN) || defined(CATCH_CONFIG_RUNNER)
00050 # define CATCH_IMPL
00051 # define CATCH_CONFIG_ALL_PARTS
00052 #endif
00053
00054 // In the impl file, we want to have access to all parts of the headers
00055 // Can also be used to sanely support PCHs
```

```

00056 #if defined(CATCH_CONFIG_ALL_PARTS)
00057 #   define CATCH_CONFIG_EXTERNAL_INTERFACES
00058 #   if defined(CATCH_CONFIG_DISABLE_MATCHERS)
00059 #       undef CATCH_CONFIG_DISABLE_MATCHERS
00060 #   endif
00061 #   if !defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
00062 #       define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
00063 #   endif
00064 #endif
00065
00066 #if !defined(CATCH_CONFIG_IMPL_ONLY)
00067 // start catch_platform.h
00068
00069 // See e.g.:
00070 // https://opensource.apple.com/source/CarbonHeaders/CarbonHeaders-18.1/TargetConditionals.h.auto.html
00071 #ifdef __APPLE__
00072 #   include <TargetConditionals.h>
00073 #   if (defined(TARGET_OS_OSX) && TARGET_OS_OSX == 1) || \
00074       (defined(TARGET_OS_MAC) && TARGET_OS_MAC == 1)
00075 #       define CATCH_PLATFORM_MAC
00076 #   elif (defined(TARGET_OS_IPHONE) && TARGET_OS_IPHONE == 1)
00077 #       define CATCH_PLATFORM_IPHONE
00078 #   endif
00079
00080 #elif defined(linux) || defined(__linux) || defined(__linux__)
00081 #   define CATCH_PLATFORM_LINUX
00082
00083 #elif defined(WIN32) || defined(__WIN32__) || defined(_WIN32) || defined(_MSC_VER) || \
00084       defined(__MINGW32__)
00085 #   define CATCH_PLATFORM_WINDOWS
00086 #endif
00087 // end catch_platform.h
00088
00089 #ifdef CATCH_IMPL
00090 #   ifndef CLARA_CONFIG_MAIN
00091 #       define CLARA_CONFIG_MAIN_NOT_DEFINED
00092 #       define CLARA_CONFIG_MAIN
00093 #   endif
00094 #endif
00095
00096 // start catch_user_interfaces.h
00097
00098 namespace Catch {
00099     unsigned int rngSeed();
00100 }
00101
00102 // end catch_user_interfaces.h
00103 // start catch_tag_alias_autoregistrar.h
00104
00105 // start catch_common.h
00106
00107 // start catch_compiler_capabilities.h
00108
00109 // Detect a number of compiler features - by compiler
00110 // The following features are defined:
00111 //
00112 // CATCH_CONFIG_COUNTER : is the __COUNTER__ macro supported?
00113 // CATCH_CONFIG_WINDOWS_SEH : is Windows SEH supported?
00114 // CATCH_CONFIG_POSIX_SIGNALS : are POSIX signals supported?
00115 // CATCH_CONFIG_DISABLE_EXCEPTIONS : Are exceptions enabled?
00116 // *****
00117 // Note to maintainers: if new toggles are added please document them
00118 // in configuration.md, too
00119 // *****
00120
00121 // In general each macro has a _NO_<feature name> form
00122 // (e.g. CATCH_CONFIG_NO_POSIX_SIGNALS) which disables the feature.
00123 // Many features, at point of detection, define an _INTERNAL_ macro, so they
00124 // can be combined, en-mass, with the _NO_ forms later.
00125
00126 #ifdef __cplusplus
00127
00128 #   if (__cplusplus >= 201402L) || (defined(_MSVC_LANG) && _MSVC_LANG >= 201402L)
00129 #       define CATCH_CPP14_OR_GREATER
00130 #   endif
00131
00132 #   if (__cplusplus >= 201703L) || (defined(_MSVC_LANG) && _MSVC_LANG >= 201703L)
00133 #       define CATCH_CPP17_OR_GREATER
00134 #   endif
00135 #endif
00136 #endif
00137
00138 // Only GCC compiler should be used in this block, so other compilers trying to
00139 // mask themselves as GCC should be ignored.
00140 #if defined(__GNUC__) && !defined(__clang__) && !defined(__ICC) && !defined(__CUDACC__) && \
    !defined(__LCC__)

```

```

00141 #   define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION _Pragma( "GCC diagnostic push" )
00142 #   define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION  _Pragma( "GCC diagnostic pop" )
00143
00144 #   define CATCH_INTERNAL_IGNORE_BUT_WARN(...) (void)__builtin_constant_p(__VA_ARGS__)
00145
00146 #endif
00147
00148 #if defined(__clang__)
00149
00150 #   define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION _Pragma( "clang diagnostic push" )
00151 #   define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION  _Pragma( "clang diagnostic pop" )
00152
00153 // As of this writing, IBM XL's implementation of __builtin_constant_p has a bug
00154 // which results in calls to destructors being emitted for each temporary,
00155 // without a matching initialization. In practice, this can result in something
00156 // like `std::string::~~string` being called on an uninitialized value.
00157 //
00158 // For example, this code will likely segfault under IBM XL:
00159 // ```
00160 // REQUIRE(std::string("12") + "34" == "1234")
00161 // ```
00162 //
00163 // Therefore, `CATCH_INTERNAL_IGNORE_BUT_WARN` is not implemented.
00164 #   if !defined(__ibmxl__) && !defined(__CUDACC__)
00165 #       define CATCH_INTERNAL_IGNORE_BUT_WARN(...) (void)__builtin_constant_p(__VA_ARGS__) /*
00166 #           NOLINT(cppcoreguidelines-pro-type-vararg, hicpp-vararg) */
00167 #   endif
00168
00169 #   define CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
00170     _Pragma( "clang diagnostic ignored \"-Wexit-time-destructors\"" ) \
00171     _Pragma( "clang diagnostic ignored \"-Wglobal-constructors\"" )
00172
00173 #   define CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS \
00174     _Pragma( "clang diagnostic ignored \"-Wparentheses\"" )
00175
00176 #   define CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
00177     _Pragma( "clang diagnostic ignored \"-Wunused-variable\"" )
00178
00179 #   define CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
00180     _Pragma( "clang diagnostic ignored \"-Wgnu-zero-variadic-macro-arguments\"" )
00181
00182 #   define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
00183     _Pragma( "clang diagnostic ignored \"-Wunused-template\"" )
00184
00185 #endif // __clang__
00186
00187 // Assume that non-Windows platforms support posix signals by default
00188 #if !defined(CATCH_PLATFORM_WINDOWS)
00189     #define CATCH_INTERNAL_CONFIG_POSIX_SIGNALS
00190 #endif
00191
00192 // We know some environments not to support full POSIX signals
00193 #if defined(__CYGWIN__) || defined(__QNX__) || defined(__EMSCRIPTEN__) || defined(__DJGPP__)
00194     #define CATCH_INTERNAL_CONFIG_NO_POSIX_SIGNALS
00195 #endif
00196
00197 #ifdef __OS400__
00198     #define CATCH_INTERNAL_CONFIG_NO_POSIX_SIGNALS
00199     #define CATCH_CONFIG_COLOUR_NONE
00200 #endif
00201
00202 // Android somehow still does not support std::to_string
00203 #if defined(__ANDROID__)
00204     #define CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING
00205     #define CATCH_INTERNAL_CONFIG_ANDROID_LOGWRITE
00206 #endif
00207
00208 // Not all Windows environments support SEH properly
00209 #if defined(__MINGW32__)
00210     #define CATCH_INTERNAL_CONFIG_NO_WINDOWS_SEH
00211 #endif
00212
00213 // PS4
00214 #if defined(__ORBIS__)
00215     #define CATCH_INTERNAL_CONFIG_NO_NEW_CAPTURE
00216 #endif
00217
00218 // Cygwin
00219 #ifdef __CYGWIN__
00220     // Required for some versions of Cygwin to declare gettimeofday
00221     // see: http://stackoverflow.com/questions/36901803/gettimeofday-not-declared-in-this-scope-cygwin
00222     #define CATCH_INTERNAL_CONFIG_BSD_SOURCE
00223     // some versions of cygwin (most) do not support std::to_string. Use the libstd check.
00224     // https://gcc.gnu.org/onlinedocs/gcc-4.8.2/libstdc++/api/a01053_source.html line 2812-2813
00225     #if !((__cplusplus >= 201103L) && defined(_GLIBCXX_USE_C99) \
00226         && !defined(_GLIBCXX_HAVE_BROKEN_VSWPRINTF))
00227

```

```

00233
00234 #   define CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING
00235
00236 # endif
00237 #endif // __CYGWIN__
00238
00240 // Visual C++
00241 #if defined(_MSC_VER)
00242
00243 #   define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION __pragma( warning(push) )
00244 #   define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION __pragma( warning(pop) )
00245
00246 // Universal Windows platform does not support SEH
00247 // Or console colours (or console at all...)
00248 #   if defined(WINAPI_FAMILY) && (WINAPI_FAMILY == WINAPI_FAMILY_APP)
00249 #       define CATCH_CONFIG_COLOUR_NONE
00250 #   else
00251 #       define CATCH_INTERNAL_CONFIG_WINDOWS_SEH
00252 #   endif
00253
00254 // MSVC traditional preprocessor needs some workaround for __VA_ARGS__
00255 // _MSVC_TRADITIONAL == 0 means new conformant preprocessor
00256 // _MSVC_TRADITIONAL == 1 means old traditional non-conformant preprocessor
00257 #   if !defined(__clang__) // Handle Clang masquerading for msvc
00258 #       if !defined(_MSVC_TRADITIONAL) || (defined(_MSVC_TRADITIONAL) && _MSVC_TRADITIONAL)
00259 #           define CATCH_INTERNAL_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00260 #       endif // _MSVC_TRADITIONAL
00261 #   endif // __clang__
00262
00263 #endif // _MSC_VER
00264
00265 #if defined(_REENTRANT) || defined(_MSC_VER)
00266 // Enable async processing, as -pthread is specified or no additional linking is required
00267 #   define CATCH_INTERNAL_CONFIG_USE_ASYNC
00268 #endif // _MSC_VER
00269
00271 // Check if we are compiled with -fno-exceptions or equivalent
00272 #if defined(__EXCEPTIONS) || defined(__cpp_exceptions) || defined(_CPPUNWIND)
00273 #   define CATCH_INTERNAL_CONFIG_EXCEPTIONS_ENABLED
00274 #endif
00275
00277 // DJGPP
00278 #ifdef __DJGPP__
00279 #   define CATCH_INTERNAL_CONFIG_NO_WCHAR
00280 #endif // __DJGPP__
00281
00283 // Embarcadero C++Build
00284 #if defined(__BORLANDC__)
00285     #define CATCH_INTERNAL_CONFIG_POLYFILL_ISNAN
00286 #endif
00287
00289
00290 // Use of __COUNTER__ is suppressed during code analysis in
00291 // CLion/AppCode 2017.2.x and former, because __COUNTER__ is not properly
00292 // handled by it.
00293 // Otherwise all supported compilers support COUNTER macro,
00294 // but user still might want to turn it off
00295 #if ( !defined(__JETBRAINS_IDE__) || __JETBRAINS_IDE__ >= 20170300L )
00296     #define CATCH_INTERNAL_CONFIG_COUNTER
00297 #endif
00298
00300
00301 // RTX is a special version of Windows that is real time.
00302 // This means that it is detected as Windows, but does not provide
00303 // the same set of capabilities as real Windows does.
00304 #if defined(UNDER_RTSS) || defined(RTX64_BUILD)
00305     #define CATCH_INTERNAL_CONFIG_NO_WINDOWS_SEH
00306     #define CATCH_INTERNAL_CONFIG_NO_ASYNC
00307     #define CATCH_CONFIG_COLOUR_NONE
00308 #endif
00309
00310 #if !defined(_GLIBCXX_USE_C99_MATH_TR1)
00311 #define CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER
00312 #endif
00313
00314 // Various stdlib support checks that require __has_include
00315 #if defined(__has_include)
00316     // Check if string_view is available and usable
00317     #if __has_include(<string_view>) && defined(CATCH_CPP17_OR_GREATER)
00318         #define CATCH_INTERNAL_CONFIG_CPP17_STRING_VIEW
00319     #endif
00320
00321     // Check if optional is available and usable
00322     #if __has_include(<optional>) && defined(CATCH_CPP17_OR_GREATER)
00323         #define CATCH_INTERNAL_CONFIG_CPP17_OPTIONAL
00324     #endif // __has_include(<optional>) && defined(CATCH_CPP17_OR_GREATER)
00325

```



```

00326 // Check if byte is available and usable
00327 # if __has_include(<cstdint>) && defined(CATCH_CPP17_OR_GREATER)
00328 #   include <cstdint>
00329 #   if defined(__cpp_lib_byte) && (__cpp_lib_byte > 0)
00330 #       define CATCH_INTERNAL_CONFIG_CPP17_BYTE
00331 #   endif
00332 # endif // __has_include(<cstdint>) && defined(CATCH_CPP17_OR_GREATER)
00333
00334 // Check if variant is available and usable
00335 # if __has_include(<variant>) && defined(CATCH_CPP17_OR_GREATER)
00336 #   if defined(__clang__) && (__clang_major__ < 8)
00337 // work around clang bug with libstdc++ https://bugs.llvm.org/show_bug.cgi?id=31852
00338 // fix should be in clang 8, workaround in libstdc++ 8.2
00339 #   include <ciso646>
00340 #   if defined(__GLIBCXX__) && defined(_GLIBCXX_RELEASE) && (_GLIBCXX_RELEASE < 9)
00341 #       define CATCH_CONFIG_NO_CPP17_VARIANT
00342 #   else
00343 #       define CATCH_INTERNAL_CONFIG_CPP17_VARIANT
00344 #   endif // defined(__GLIBCXX__) && defined(_GLIBCXX_RELEASE) && (_GLIBCXX_RELEASE < 9)
00345 #   else
00346 #       define CATCH_INTERNAL_CONFIG_CPP17_VARIANT
00347 #   endif // defined(__clang__) && (__clang_major__ < 8)
00348 # endif // __has_include(<variant>) && defined(CATCH_CPP17_OR_GREATER)
00349 #endif // defined(__has_include)
00350
00351 #if defined(CATCH_INTERNAL_CONFIG_COUNTER) && !defined(CATCH_CONFIG_NO_COUNTER) &&
!defined(CATCH_CONFIG_COUNTER)
00352 #   define CATCH_CONFIG_COUNTER
00353 #endif
00354 #if defined(CATCH_INTERNAL_CONFIG_WINDOWS_SEH) && !defined(CATCH_CONFIG_NO_WINDOWS_SEH) &&
!defined(CATCH_CONFIG_WINDOWS_SEH) && !defined(CATCH_INTERNAL_CONFIG_NO_WINDOWS_SEH)
00355 #   define CATCH_CONFIG_WINDOWS_SEH
00356 #endif
00357 // This is set by default, because we assume that unix compilers are posix-signal-compatible by
default.
00358 #if defined(CATCH_INTERNAL_CONFIG_POSIX_SIGNALS) && !defined(CATCH_INTERNAL_CONFIG_NO_POSIX_SIGNALS)
&& !defined(CATCH_CONFIG_NO_POSIX_SIGNALS) && !defined(CATCH_CONFIG_POSIX_SIGNALS)
00359 #   define CATCH_CONFIG_POSIX_SIGNALS
00360 #endif
00361 // This is set by default, because we assume that compilers with no wchar_t support are just rare
exceptions.
00362 #if !defined(CATCH_INTERNAL_CONFIG_NO_WCHAR) && !defined(CATCH_CONFIG_NO_WCHAR) &&
!defined(CATCH_CONFIG_WCHAR)
00363 #   define CATCH_CONFIG_WCHAR
00364 #endif
00365
00366 #if !defined(CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING) && !defined(CATCH_CONFIG_NO_CPP11_TO_STRING) &&
!defined(CATCH_CONFIG_CPP11_TO_STRING)
00367 #   define CATCH_CONFIG_CPP11_TO_STRING
00368 #endif
00369
00370 #if defined(CATCH_INTERNAL_CONFIG_CPP17_OPTIONAL) && !defined(CATCH_CONFIG_NO_CPP17_OPTIONAL) &&
!defined(CATCH_CONFIG_CPP17_OPTIONAL)
00371 #   define CATCH_CONFIG_CPP17_OPTIONAL
00372 #endif
00373
00374 #if defined(CATCH_INTERNAL_CONFIG_CPP17_STRING_VIEW) && !defined(CATCH_CONFIG_NO_CPP17_STRING_VIEW) &&
!defined(CATCH_CONFIG_CPP17_STRING_VIEW)
00375 #   define CATCH_CONFIG_CPP17_STRING_VIEW
00376 #endif
00377
00378 #if defined(CATCH_INTERNAL_CONFIG_CPP17_VARIANT) && !defined(CATCH_CONFIG_NO_CPP17_VARIANT) &&
!defined(CATCH_CONFIG_CPP17_VARIANT)
00379 #   define CATCH_CONFIG_CPP17_VARIANT
00380 #endif
00381
00382 #if defined(CATCH_INTERNAL_CONFIG_CPP17_BYTE) && !defined(CATCH_CONFIG_NO_CPP17_BYTE) &&
!defined(CATCH_CONFIG_CPP17_BYTE)
00383 #   define CATCH_CONFIG_CPP17_BYTE
00384 #endif
00385
00386 #if defined(CATCH_CONFIG_EXPERIMENTAL_REDIRECT)
00387 #   define CATCH_INTERNAL_CONFIG_NEW_CAPTURE
00388 #endif
00389
00390 #if defined(CATCH_INTERNAL_CONFIG_NEW_CAPTURE) && !defined(CATCH_INTERNAL_CONFIG_NO_NEW_CAPTURE) &&
!defined(CATCH_CONFIG_NO_NEW_CAPTURE) && !defined(CATCH_CONFIG_NEW_CAPTURE)
00391 #   define CATCH_CONFIG_NEW_CAPTURE
00392 #endif
00393
00394 #if !defined(CATCH_INTERNAL_CONFIG_EXCEPTIONS_ENABLED) && !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
00395 #   define CATCH_CONFIG_DISABLE_EXCEPTIONS
00396 #endif
00397
00398 #if defined(CATCH_INTERNAL_CONFIG_POLYFILL_ISNAN) && !defined(CATCH_CONFIG_NO_POLYFILL_ISNAN) &&
!defined(CATCH_CONFIG_POLYFILL_ISNAN)
00399 #   define CATCH_CONFIG_POLYFILL_ISNAN

```

```

00400 #endif
00401
00402 #if defined(CATCH_INTERNAL_CONFIG_USE_ASYNC) && !defined(CATCH_INTERNAL_CONFIG_NO_ASYNC) &&
!defined(CATCH_CONFIG_NO_USE_ASYNC) && !defined(CATCH_CONFIG_USE_ASYNC)
00403 # define CATCH_CONFIG_USE_ASYNC
00404 #endif
00405
00406 #if defined(CATCH_INTERNAL_CONFIG_ANDROID_LOGWRITE) && !defined(CATCH_CONFIG_NO_ANDROID_LOGWRITE) &&
!defined(CATCH_CONFIG_ANDROID_LOGWRITE)
00407 # define CATCH_CONFIG_ANDROID_LOGWRITE
00408 #endif
00409
00410 #if defined(CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER) && !defined(CATCH_CONFIG_NO_GLOBAL_NEXTAFTER) &&
!defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
00411 # define CATCH_CONFIG_GLOBAL_NEXTAFTER
00412 #endif
00413
00414 // Even if we do not think the compiler has that warning, we still have
00415 // to provide a macro that can be used by the code.
00416 #if !defined(CATCH_INTERNAL_START_WARNINGS_SUPPRESSION)
00417 # define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION
00418 #endif
00419 #if !defined(CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION)
00420 # define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
00421 #endif
00422 #if !defined(CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS)
00423 # define CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS
00424 #endif
00425 #if !defined(CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS)
00426 # define CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS
00427 #endif
00428 #if !defined(CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS)
00429 # define CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS
00430 #endif
00431 #if !defined(CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS)
00432 # define CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS
00433 #endif
00434
00435 // The goal of this macro is to avoid evaluation of the arguments, but
00436 // still have the compiler warn on problems inside...
00437 #if !defined(CATCH_INTERNAL_IGNORE_BUT_WARN)
00438 # define CATCH_INTERNAL_IGNORE_BUT_WARN(...)
00439 #endif
00440
00441 #if defined(__APPLE__) && defined(__apple_build_version__) && (__clang_major__ < 10)
00442 # undef CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
00443 #elif defined(__clang__) && (__clang_major__ < 5)
00444 # undef CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
00445 #endif
00446
00447 #if !defined(CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS)
00448 # define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
00449 #endif
00450
00451 #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
00452 #define CATCH_TRY if ((true))
00453 #define CATCH_CATCH_ALL if ((false))
00454 #define CATCH_CATCH_ANON(type) if ((false))
00455 #else
00456 #define CATCH_TRY try
00457 #define CATCH_CATCH_ALL catch (...)
00458 #define CATCH_CATCH_ANON(type) catch (type)
00459 #endif
00460
00461 #if defined(CATCH_INTERNAL_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR) &&
!defined(CATCH_CONFIG_NO_TRADITIONAL_MSVC_PREPROCESSOR) &&
!defined(CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR)
00462 #define CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00463 #endif
00464
00465 // end catch_compiler_capabilities.h
00466 #define INTERNAL_CATCH_UNIQUE_NAME_LINE2( name, line ) name##line
00467 #define INTERNAL_CATCH_UNIQUE_NAME_LINE( name, line ) INTERNAL_CATCH_UNIQUE_NAME_LINE2( name, line )
00468 #ifdef CATCH_CONFIG_COUNTER
00469 # define INTERNAL_CATCH_UNIQUE_NAME( name ) INTERNAL_CATCH_UNIQUE_NAME_LINE( name, __COUNTER__ )
00470 #else
00471 # define INTERNAL_CATCH_UNIQUE_NAME( name ) INTERNAL_CATCH_UNIQUE_NAME_LINE( name, __LINE__ )
00472 #endif
00473
00474 #include <iosfwd>
00475 #include <string>
00476 #include <cstdint>
00477
00478 // We need a dummy global operator« so we can bring it into Catch namespace later
00479 struct Catch_global_namespace_dummy {};
00480 std::ostream& operator«(std::ostream&, Catch_global_namespace_dummy);
00481

```

```

00482 namespace Catch {
00483
00484     struct CaseSensitive { enum Choice {
00485         Yes,
00486         No
00487     }; };
00488
00489     class NonCopyable {
00490     public:
00491         NonCopyable( NonCopyable const& )           = delete;
00492         NonCopyable( NonCopyable && )               = delete;
00493         NonCopyable& operator = ( NonCopyable const& ) = delete;
00494         NonCopyable& operator = ( NonCopyable && )     = delete;
00495
00496     protected:
00497         NonCopyable();
00498         virtual ~NonCopyable();
00499     };
00500
00501     struct SourceLineInfo {
00502     public:
00503         SourceLineInfo() = delete;
00504         SourceLineInfo( char const* _file, std::size_t _line ) noexcept
00505             : file( _file ),
00506               line( _line )
00507         {}
00508
00509         SourceLineInfo( SourceLineInfo const& other )           = default;
00510         SourceLineInfo& operator = ( SourceLineInfo const& )   = default;
00511         SourceLineInfo( SourceLineInfo&& )                     noexcept = default;
00512         SourceLineInfo& operator = ( SourceLineInfo&& )         noexcept = default;
00513
00514         bool empty() const noexcept { return file[0] == '\0'; }
00515         bool operator == ( SourceLineInfo const& other ) const noexcept;
00516         bool operator < ( SourceLineInfo const& other ) const noexcept;
00517
00518         char const* file;
00519         std::size_t line;
00520     };
00521
00522     std::ostream& operator << ( std::ostream& os, SourceLineInfo const& info );
00523
00524     // Bring in operator<< from global namespace into Catch namespace
00525     // This is necessary because the overload of operator<< above makes
00526     // lookup stop at namespace Catch
00527     using ::operator<<;
00528
00529     // Use this in variadic streaming macros to allow
00530     //   » +StreamEndStop
00531     // as well as
00532     //   » stuff +StreamEndStop
00533     struct StreamEndStop {
00534     public:
00535         std::string operator+() const;
00536     };
00537
00538     template<typename T>
00539     T const& operator + ( T const& value, StreamEndStop ) {
00540         return value;
00541     }
00542
00543 #define CATCH_INTERNAL_LINEINFO \
00544     ::Catch::SourceLineInfo( __FILE__, static_cast<std::size_t>( __LINE__ ) )
00545
00546 // end catch_common.h
00547 namespace Catch {
00548
00549     struct RegistrarForTagAliases {
00550     public:
00551         RegistrarForTagAliases( char const* alias, char const* tag, SourceLineInfo const& lineInfo );
00552     };
00553
00554 #define CATCH_REGISTER_TAG_ALIAS( alias, spec ) \
00555     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
00556     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
00557     namespace{ Catch::RegistrarForTagAliases INTERNAL_CATCH_UNIQUE_NAME( AutoRegisterTagAlias ) ( \
00558         alias, spec, CATCH_INTERNAL_LINEINFO ); } \
00559     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
00560
00561 // end catch_tag_alias_autoregistrar.h
00562 // start catch_test_registry.h
00563
00564 #include <vector>
00565
00566 namespace Catch {
00567

```

```

00568     class TestSpec;
00569
00570     struct ITestInvoker {
00571         virtual void invoke () const = 0;
00572         virtual ~ITestInvoker();
00573     };
00574
00575     class TestCase;
00576     struct IConfig;
00577
00578     struct ITestCaseRegistry {
00579         virtual ~ITestCaseRegistry();
00580         virtual std::vector<TestCase> const& getAllTests() const = 0;
00581         virtual std::vector<TestCase> const& getAllTestsSorted( IConfig const& config ) const = 0;
00582     };
00583
00584     bool isThrowSafe( TestCase const& testCase, IConfig const& config );
00585     bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config );
00586     std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
testSpec, IConfig const& config );
00587     std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config );
00588 }
00589
00590
00591 // end catch_interfaces_testcase.h
00592 // start catch_stringref.h
00593
00594 #include <cstdint>
00595 #include <string>
00596 #include <iosfwd>
00597 #include <cassert>
00598
00599 namespace Catch {
00600
00601     class StringRef {
00602     public:
00603         using size_type = std::size_t;
00604         using const_iterator = const char*;
00605
00606     private:
00607         static constexpr char const* const s_empty = "";
00608
00609         char const* m_start = s_empty;
00610         size_type m_size = 0;
00611
00612     public: // construction
00613         constexpr StringRef() noexcept = default;
00614
00615         StringRef( char const* rawChars ) noexcept;
00616
00617         constexpr StringRef( char const* rawChars, size_type size ) noexcept
00618         :   m_start( rawChars ),
00619             m_size( size )
00619         {}
00620
00621         StringRef( std::string const& stdString ) noexcept
00622         :   m_start( stdString.c_str() ),
00623             m_size( stdString.size() )
00624         {}
00625
00626         explicit operator std::string() const {
00627             return std::string(m_start, m_size);
00628         }
00629
00630     public: // operators
00631         auto operator == ( StringRef const& other ) const noexcept -> bool;
00632         auto operator != ( StringRef const& other ) const noexcept -> bool {
00633             return !(*this == other);
00634         }
00635
00636         auto operator[] ( size_type index ) const noexcept -> char {
00637             assert(index < m_size);
00638             return m_start[index];
00639         }
00640
00641     public: // named queries
00642         constexpr auto empty() const noexcept -> bool {
00643             return m_size == 0;
00644         }
00645
00646         constexpr auto size() const noexcept -> size_type {
00647             return m_size;
00648         }
00649
00650         // Returns the current start pointer. If the StringRef is not
00651         // null-terminated, throws std::domain_exception
00652         auto c_str() const -> char const*;
00653
00654
00655
00656

```

```

00657     public: // substrings and searches
00658         // Returns a substring of [start, start + length).
00659         // If start + length > size(), then the substring is [start, size()).
00660         // If start > size(), then the substring is empty.
00661         auto substr( size_type start, size_type length ) const noexcept -> StringRef;
00662
00663         // Returns the current start pointer. May not be null-terminated.
00664         auto data() const noexcept -> char const*;
00665
00666         constexpr auto isNullTerminated() const noexcept -> bool {
00667             return m_start[m_size] == '\0';
00668         }
00669
00670     public: // iterators
00671         constexpr const_iterator begin() const { return m_start; }
00672         constexpr const_iterator end() const { return m_start + m_size; }
00673     };
00674
00675     auto operator += ( std::string& lhs, StringRef const& sr ) -> std::string&;
00676     auto operator << ( std::ostream& os, StringRef const& sr ) -> std::ostream&;
00677
00678     constexpr auto operator "" _sr( char const* rawChars, std::size_t size ) noexcept -> StringRef {
00679         return StringRef( rawChars, size );
00680     }
00681 } // namespace Catch
00682
00683 constexpr auto operator "" _catch_sr( char const* rawChars, std::size_t size ) noexcept ->
Catch::StringRef {
00684     return Catch::StringRef( rawChars, size );
00685 }
00686
00687 // end catch_stringref.h
00688 // start catch_preprocessor.hpp
00689
00690 #define CATCH_RECURSION_LEVEL0(...) __VA_ARGS__
00691 #define CATCH_RECURSION_LEVEL1(...)
00692     CATCH_RECURSION_LEVEL0(CATCH_RECURSION_LEVEL0(CATCH_RECURSION_LEVEL0(__VA_ARGS__)))
00693 #define CATCH_RECURSION_LEVEL2(...)
00694     CATCH_RECURSION_LEVEL1(CATCH_RECURSION_LEVEL1(CATCH_RECURSION_LEVEL1(__VA_ARGS__)))
00695 #define CATCH_RECURSION_LEVEL3(...)
00696     CATCH_RECURSION_LEVEL2(CATCH_RECURSION_LEVEL2(CATCH_RECURSION_LEVEL2(__VA_ARGS__)))
00697 #define CATCH_RECURSION_LEVEL4(...)
00698     CATCH_RECURSION_LEVEL3(CATCH_RECURSION_LEVEL3(CATCH_RECURSION_LEVEL3(__VA_ARGS__)))
00699 #define CATCH_RECURSION_LEVEL5(...)
00700     CATCH_RECURSION_LEVEL4(CATCH_RECURSION_LEVEL4(CATCH_RECURSION_LEVEL4(__VA_ARGS__)))
00701
00702 #ifdef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00703 #define INTERNAL_CATCH_EXPAND_VARGS(...) __VA_ARGS__
00704 // MSVC needs more evaluations
00705 #define CATCH_RECURSION_LEVEL6(...)
00706     CATCH_RECURSION_LEVEL5(CATCH_RECURSION_LEVEL5(CATCH_RECURSION_LEVEL5(__VA_ARGS__)))
00707 #define CATCH_RECURSE(...) CATCH_RECURSION_LEVEL6(CATCH_RECURSION_LEVEL6(__VA_ARGS__))
00708 #else
00709 #define CATCH_RECURSE(...) CATCH_RECURSION_LEVEL5(__VA_ARGS__)
00710 #endif
00711
00712 #define CATCH_REC_END(...)
00713 #define CATCH_REC_OUT
00714
00715 #define CATCH_EMPTY()
00716 #define CATCH_DEFER(id) id CATCH_EMPTY()
00717
00718 #define CATCH_REC_GET_END2() 0, CATCH_REC_END
00719 #define CATCH_REC_GET_END1(...) CATCH_REC_GET_END2
00720 #define CATCH_REC_GET_END(...) CATCH_REC_GET_END1
00721 #define CATCH_REC_NEXT0(test, next, ...) next CATCH_REC_OUT
00722 #define CATCH_REC_NEXT1(test, next) CATCH_DEFER ( CATCH_REC_NEXT0 ) ( test, next, 0)
00723 #define CATCH_REC_NEXT(test, next) CATCH_REC_NEXT1(CATCH_REC_GET_END test, next)
00724
00725 #define CATCH_REC_LIST0(f, x, peek, ...) , f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1) )
00726     ( f, peek, __VA_ARGS__ )
00727 #define CATCH_REC_LIST1(f, x, peek, ...) , f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST0) )
00728     ( f, peek, __VA_ARGS__ )
00729 #define CATCH_REC_LIST2(f, x, peek, ...) f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1) )
00730     ( f, peek, __VA_ARGS__ )
00731
00732 #define CATCH_REC_LIST0_UD(f, userdata, x, peek, ...) , f(userdata, x) CATCH_DEFER (
00733     CATCH_REC_NEXT(peek, CATCH_REC_LIST1_UD) ) ( f, userdata, peek, __VA_ARGS__ )
00734 #define CATCH_REC_LIST1_UD(f, userdata, x, peek, ...) , f(userdata, x) CATCH_DEFER (
00735     CATCH_REC_NEXT(peek, CATCH_REC_LIST0_UD) ) ( f, userdata, peek, __VA_ARGS__ )
00736 #define CATCH_REC_LIST2_UD(f, userdata, x, peek, ...) f(userdata, x) CATCH_DEFER (
00737     CATCH_REC_NEXT(peek, CATCH_REC_LIST1_UD) ) ( f, userdata, peek, __VA_ARGS__ )
00738
00739 // Applies the function macro `f` to each of the remaining parameters, inserts commas between the
00740 results,
00741 // and passes userdata as the first parameter to each invocation,

```



```

00796     template<typename T> \
00797     struct append<T> { using type = T; }; \
00798     template< template<typename...> class L1, typename...E1, template<typename...> class L2,
typename...E2, typename...Rest> \
00799     struct append<L1<E1...>, L2<E2...>, Rest...> { using type = typename append<L1<E1...,E2...>,
Rest...>::type; }; \
00800     template< template<typename...> class L1, typename...E1, typename...Rest> \
00801     struct append<L1<E1...>, TypeList<mpl_::na>, Rest...> { using type = L1<E1...>; }; \
00802     \
00803     template< template<typename...> class Container, template<typename...> class List,
typename...elems> \
00804     struct rewrap<TemplateTypeList<Container>, List<elems...> { using type =
TypeList<Container<elems...>; }; \
00805     template< template<typename...> class Container, template<typename...> class List, class...Elements,
typename...Elements> \
00806     struct rewrap<TemplateTypeList<Container>, List<Elements...>, Elements...> { using type = typename
append<TypeList<Container<Elements...>, typename rewrap<TemplateTypeList<Container>,
Elements...>::type::type; }; \
00807     \
00808     template<template <typename...> class Final, template< typename...> class...Containers,
typename...Types> \
00809     struct create<Final, TemplateTypeList<Containers...>, TypeList<Types...> { using type = typename
append<Final<>, typename rewrap<TemplateTypeList<Containers>, Types...>::type...>::type; }; \
00810     template<template <typename...> class Final, template <typename...> class List, typename...Ts> \
00811     struct convert<Final, List<Ts...> { using type = typename append<Final<>,TypeList<Ts>...>::type;
};
00812
00813 #define INTERNAL_CATCH_NTTP_1(signature, ...) \
00814     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> struct Nttp{}; \
00815     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> \
00816     constexpr auto get_wrapper() noexcept -> Nttp<__VA_ARGS__> { return {}; } \
00817     template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class...> struct
NttpTemplateTypeList{}; \
00818     template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class...Cs> \
00819     constexpr auto get_wrapper() noexcept -> NttpTemplateTypeList<Cs...> { return {}; } \
00820     \
00821     template< template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class Container,
template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class List,
INTERNAL_CATCH_REMOVE_PARENS(signature)> \
00822     struct rewrap<NttpTemplateTypeList<Container>, List<__VA_ARGS__> { using type =
TypeList<Container<__VA_ARGS__>; }; \
00823     template< template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class Container,
template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class List, INTERNAL_CATCH_REMOVE_PARENS(signature),
typename...Elements> \
00824     struct rewrap<NttpTemplateTypeList<Container>, List<__VA_ARGS__>, Elements...> { using type =
typename append<TypeList<Container<__VA_ARGS__>, typename rewrap<NttpTemplateTypeList<Container>,
Elements...>::type::type; }; \
00825     template<template <typename...> class Final, template<INTERNAL_CATCH_REMOVE_PARENS(signature)>
class...Containers, typename...Types> \
00826     struct create<Final, NttpTemplateTypeList<Containers...>, TypeList<Types...> { using type =
typename append<Final<>, typename rewrap<NttpTemplateTypeList<Containers>, Types...>::type...>::type;
};
00827
00828 #define INTERNAL_CATCH_DECLARE_SIG_TEST0(TestName)
00829 #define INTERNAL_CATCH_DECLARE_SIG_TEST1(TestName, signature) \
00830     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> \
00831     static void TestName()
00832 #define INTERNAL_CATCH_DECLARE_SIG_TEST_X(TestName, signature, ...) \
00833     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> \
00834     static void TestName()
00835
00836 #define INTERNAL_CATCH_DEFINE_SIG_TEST0(TestName)
00837 #define INTERNAL_CATCH_DEFINE_SIG_TEST1(TestName, signature) \
00838     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> \
00839     static void TestName()
00840 #define INTERNAL_CATCH_DEFINE_SIG_TEST_X(TestName, signature,...) \
00841     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> \
00842     static void TestName()
00843
00844 #define INTERNAL_CATCH_NTTP_REGISTER0(TestFunc, signature) \
00845     template<typename Type> \
00846     void reg_test(TypeList<Type>, Catch::NameAndTags nameAndTags) \
00847     { \
00848         Catch::AutoReg( Catch::makeTestInvoker(&TestFunc<Type>), CATCH_INTERNAL_LINEINFO,
Catch::StringRef(), nameAndTags); \
00849     }
00850
00851 #define INTERNAL_CATCH_NTTP_REGISTER(TestFunc, signature, ...) \
00852     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> \
00853     void reg_test(Nttp<__VA_ARGS__>, Catch::NameAndTags nameAndTags) \
00854     { \
00855         Catch::AutoReg( Catch::makeTestInvoker(&TestFunc<__VA_ARGS__>), CATCH_INTERNAL_LINEINFO,
Catch::StringRef(), nameAndTags); \
00856     }
00857
00858 #define INTERNAL_CATCH_NTTP_REGISTER_METHOD0(TestName, signature, ...) \
00859     template<typename Type> \

```



```

00860     void reg_test(TypeList<Type>, Catch::StringRef className, Catch::NameAndTags nameAndTags)\
00861     {\
00862         Catch::AutoReg( Catch::makeTestInvoker(&TestName<Type>::test), CATCH_INTERNAL_LINEINFO,
        className, nameAndTags);\
00863     }
00864
00865 #define INTERNAL_CATCH_NTTP_REGISTER_METHOD(TestName, signature, ...)\
00866     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00867     void reg_test(Nttp<__VA_ARGS__>, Catch::StringRef className, Catch::NameAndTags nameAndTags)\
00868     {\
00869         Catch::AutoReg( Catch::makeTestInvoker(&TestName<__VA_ARGS__>::test), CATCH_INTERNAL_LINEINFO,
        className, nameAndTags);\
00870     }
00871
00872 #define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0(TestName, ClassName)
00873 #define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1(TestName, ClassName, signature)\
00874     template<typename TestType> \
00875     struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName)<TestType> { \
00876         void test();\
00877     }
00878
00879 #define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X(TestName, ClassName, signature, ...)\
00880     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> \
00881     struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName)<__VA_ARGS__> { \
00882         void test();\
00883     }
00884
00885 #define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0(TestName)
00886 #define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1(TestName, signature)\
00887     template<typename TestType> \
00888     void INTERNAL_CATCH_MAKE_NAMESPACED(TestName)::TestName<TestType>::test()
00889 #define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X(TestName, signature, ...)\
00890     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> \
00891     void INTERNAL_CATCH_MAKE_NAMESPACED(TestName)::TestName<__VA_ARGS__>::test()
00892
00893 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00894 #define INTERNAL_CATCH_NTTP_0
00895 #define INTERNAL_CATCH_NTTP_GEN(...) INTERNAL_CATCH_VA_NARGS_IMPL(__VA_ARGS__,
        INTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_1(__VA_ARGS__),
        INTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_1(__VA_ARGS__),
        INTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_1(
        __VA_ARGS__), INTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_1(
        __VA_ARGS__),INTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_0)
00896 #define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, ...) INTERNAL_CATCH_VA_NARGS_IMPL( "dummy",
        __VA_ARGS__, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
        INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
        INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0)(TestName, __VA_ARGS__)
00897 #define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD(TestName, ClassName, ...) INTERNAL_CATCH_VA_NARGS_IMPL(
        "dummy", __VA_ARGS__,
        INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
        INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
        INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0)(TestName, ClassName,
        __VA_ARGS__)
00898 #define INTERNAL_CATCH_NTTP_REG_METHOD_GEN(TestName, ...) INTERNAL_CATCH_VA_NARGS_IMPL( "dummy",
        __VA_ARGS__, INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
        INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,INTERNAL_CATCH_NTTP_REGISTER_METHOD,INTERNAL_CATCH_NTTP_REGISTER_METHOD,INTERNAL_CATCH_NTTP_REGISTER_METHOD,INTERNAL_CATCH_NTTP_REGISTER_METHOD,INTERNAL_CATCH_NTTP_REGISTER_METHOD,INTERNAL_CATCH_NTTP_REGISTER_METHOD,INTERNAL_CATCH_NTTP_REGISTER_METHOD0)(TestName, __VA_ARGS__)
00899 #define INTERNAL_CATCH_NTTP_REG_GEN(TestFunc, ...) INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
        INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,INTERNAL_CATCH_NTTP_REGISTER,INTERNAL_CATCH_NTTP_REGISTER,INTERNAL_CATCH_NTTP_REGISTER,INTERNAL_CATCH_NTTP_REGISTER,INTERNAL_CATCH_NTTP_REGISTER,INTERNAL_CATCH_NTTP_REGISTER,INTERNAL_CATCH_NTTP_REGISTER,INTERNAL_CATCH_NTTP_REGISTER,INTERNAL_CATCH_NTTP_REGISTER0)(TestFunc, __VA_ARGS__)
00900 #define INTERNAL_CATCH_DEFINE_SIG_TEST(TestName, ...) INTERNAL_CATCH_VA_NARGS_IMPL( "dummy",
        __VA_ARGS__, INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST1,INTERNAL_CATCH_DEFINE_SIG_TEST0)(TestName, __VA_ARGS__)
00901 #define INTERNAL_CATCH_DECLARE_SIG_TEST(TestName, ...) INTERNAL_CATCH_VA_NARGS_IMPL( "dummy",
        __VA_ARGS__, INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST1,INTERNAL_CATCH_DECLARE_SIG_TEST0)(TestName, __VA_ARGS__)
00902 #define INTERNAL_CATCH_REMOVE_PARENS_GEN(...) INTERNAL_CATCH_VA_NARGS_IMPL(__VA_ARGS__,
        INTERNAL_CATCH_REMOVE_PARENS_11_ARG,INTERNAL_CATCH_REMOVE_PARENS_10_ARG,INTERNAL_CATCH_REMOVE_PARENS_9_ARG,INTERNAL_CATCH_REMOVE_PARENS_8_ARG,INTERNAL_CATCH_REMOVE_PARENS_7_ARG,INTERNAL_CATCH_REMOVE_PARENS_6_ARG,INTERNAL_CATCH_REMOVE_PARENS_5_ARG,INTERNAL_CATCH_REMOVE_PARENS_4_ARG,INTERNAL_CATCH_REMOVE_PARENS_3_ARG,INTERNAL_CATCH_REMOVE_PARENS_2_ARG,INTERNAL_CATCH_REMOVE_PARENS_1_ARG)
00903 #else
00904 #define INTERNAL_CATCH_NTTP_0(signature)
00905 #define INTERNAL_CATCH_NTTP_GEN(...)
        INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL(__VA_ARGS__, INTERNAL_CATCH_NTTP_1,

```



```

INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1,
INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1,
INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_0) ( __VA_ARGS__ )
00906 #define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, ...)
INTERNAL_CATCH_EXPAND_VARGS (INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0) (TestName,
__VA_ARGS__))
00907 #define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD(TestName, ClassName, ...)
INTERNAL_CATCH_EXPAND_VARGS (INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0) (TestName, ClassName,
__VA_ARGS__))
00908 #define INTERNAL_CATCH_NTTP_REG_METHOD_GEN(TestName, ...)
INTERNAL_CATCH_EXPAND_VARGS (INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD0,
INTERNAL_CATCH_NTTP_REGISTER_METHOD0) (TestName, __VA_ARGS__))
00909 #define INTERNAL_CATCH_NTTP_REG_GEN(TestFunc, ...)
INTERNAL_CATCH_EXPAND_VARGS (INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,
INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,
INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,
INTERNAL_CATCH_NTTP_REGISTER0, INTERNAL_CATCH_NTTP_REGISTER0) (TestFunc, __VA_ARGS__))
00910 #define INTERNAL_CATCH_DEFINE_SIG_TEST(TestName, ...)
INTERNAL_CATCH_EXPAND_VARGS (INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST1,
INTERNAL_CATCH_DEFINE_SIG_TEST0) (TestName, __VA_ARGS__))
00911 #define INTERNAL_CATCH_DECLARE_SIG_TEST(TestName, ...)
INTERNAL_CATCH_EXPAND_VARGS (INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST1, INTERNAL_CATCH_DECLARE_SIG_TEST0) (TestName, __VA_ARGS__))
00912 #define INTERNAL_CATCH_REMOVE_PARENS_GEN(...)
INTERNAL_CATCH_EXPAND_VARGS (INTERNAL_CATCH_VA_NARGS_IMPL(__VA_ARGS__,
INTERNAL_CATCH_REMOVE_PARENS_11_ARG, INTERNAL_CATCH_REMOVE_PARENS_10_ARG, INTERNAL_CATCH_REMOVE_PARENS_9_ARG, INTERNAL_CATCH_REMOVE_PARENS_8_ARG,
INTERNAL_CATCH_REMOVE_PARENS_7_ARG, INTERNAL_CATCH_REMOVE_PARENS_6_ARG, INTERNAL_CATCH_REMOVE_PARENS_5_ARG, INTERNAL_CATCH_REMOVE_PARENS_4_ARG,
INTERNAL_CATCH_REMOVE_PARENS_3_ARG, INTERNAL_CATCH_REMOVE_PARENS_2_ARG, INTERNAL_CATCH_REMOVE_PARENS_1_ARG) ...)
00913 #endif
00914
00915 // end catch_preprocessor.hpp
00916 // start catch_meta.hpp
00917
00918
00919 #include <type_traits>
00920
00921 namespace Catch {
00922     template<typename T>
00923     struct always_false : std::false_type {};
00924
00925     template<typename> struct true_given : std::true_type {};
00926     struct is_callable_tester {
00927         template<typename Fun, typename... Args>
00928         true_given<decltype(std::declval<Fun>() (std::declval<Args>()...))> static test(int);
00929         template<typename...>
00930         std::false_type static test(...);
00931     };
00932
00933     template<typename T>
00934     struct is_callable;
00935
00936     template<typename Fun, typename... Args>
00937     struct is_callable<Fun(Args...)> : decltype(is_callable_tester::test<Fun, Args...> (0)) {};
00938
00939 #if defined(__cpp_lib_is_invocable) && __cpp_lib_is_invocable >= 201703
00940     // std::result_of is deprecated in C++17 and removed in C++20. Hence, it is
00941     // replaced with std::invoke_result here.
00942     template<typename Func, typename... U>
00943     using FunctionReturnType = std::remove_reference_t<std::remove_cv_t<std::invoke_result_t<Func,
U...>>>;
00944 #else
00945     // Keep ::type here because we still support C++11
00946     template<typename Func, typename... U>
00947     using FunctionReturnType = typename std::remove_reference<typename std::remove_cv<typename
std::result_of<Func(U...)>::type>::type>::type;

```



```

01022     #else
01023         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(Name, Tags, Signature, ...) \
01024             INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION_2(
INTERNAL_CATCH_UNIQUE_NAME( ____C_A_T_C_H____T_E_M_P_L_A_T_E____T_E_S_T____ ),
INTERNAL_CATCH_UNIQUE_NAME( ____C_A_T_C_H____T_E_M_P_L_A_T_E____T_E_S_T____F_U_N_C____ ), Name, Tags,
Signature, __VA_ARGS__ ) )
01025     #endif
01026
01027     #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01028         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION( ClassName, Name, Tags,... )
\
01029             INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME(
____C_A_T_C_H____T_E_M_P_L_A_T_E____T_E_S_T____C_L_A_S_S____ ), INTERNAL_CATCH_UNIQUE_NAME(
____C_A_T_C_H____T_E_M_P_L_A_T_E____T_E_S_T____ ), ClassName, Name, Tags, typename T, __VA_ARGS__ )
01030     #else
01031         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION( ClassName, Name, Tags,... )
\
01032             INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2(
INTERNAL_CATCH_UNIQUE_NAME( ____C_A_T_C_H____T_E_M_P_L_A_T_E____T_E_S_T____C_L_A_S_S____ ),
INTERNAL_CATCH_UNIQUE_NAME( ____C_A_T_C_H____T_E_M_P_L_A_T_E____T_E_S_T____ ), ClassName, Name, Tags,
typename T, __VA_ARGS__ ) )
01033     #endif
01034
01035     #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01036         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION( ClassName, Name, Tags,
Signature, ... ) \
01037             INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME(
____C_A_T_C_H____T_E_M_P_L_A_T_E____T_E_S_T____C_L_A_S_S____ ), INTERNAL_CATCH_UNIQUE_NAME(
____C_A_T_C_H____T_E_M_P_L_A_T_E____T_E_S_T____ ), ClassName, Name, Tags, Signature, __VA_ARGS__ )
01038     #else
01039         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION( ClassName, Name, Tags,
Signature, ... ) \
01040             INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2(
INTERNAL_CATCH_UNIQUE_NAME( ____C_A_T_C_H____T_E_M_P_L_A_T_E____T_E_S_T____C_L_A_S_S____ ),
INTERNAL_CATCH_UNIQUE_NAME( ____C_A_T_C_H____T_E_M_P_L_A_T_E____T_E_S_T____ ), ClassName, Name, Tags,
Signature, __VA_ARGS__ ) )
01041     #endif
01042 #endif
01043
01044     #define INTERNAL_CATCH_TESTCASE2( TestName, ... ) \
01045         static void TestName(); \
01046         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01047         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01048         namespace{ Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker(
&TestName ), CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ __VA_ARGS__ } ); } /*
NOLINT */ \
01049         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01050         static void TestName()
01051     #define INTERNAL_CATCH_TESTCASE( ... ) \
01052         INTERNAL_CATCH_TESTCASE2( INTERNAL_CATCH_UNIQUE_NAME( ____C_A_T_C_H____T_E_S_T____ ),
__VA_ARGS__ )
01053
01054     #define INTERNAL_CATCH_METHOD_AS_TEST_CASE( QualifiedMethod, ... ) \
01055         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01056         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01057         namespace{ Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker(
&QualifiedMethod ), CATCH_INTERNAL_LINEINFO, "&" #QualifiedMethod, Catch::NameAndTags{ __VA_ARGS__ }
); } /* NOLINT */ \
01058         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
01059
01060     #define INTERNAL_CATCH_TEST_CASE_METHOD2( TestName, ClassName, ... ) \
01061         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01062         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01063         namespace{ \
01064             struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName) { \
01065                 void test(); \
01066             }; \
01067             Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker(
&TestName::test ), CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ __VA_ARGS__ } ); } /* NOLINT
*/ \
01068         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01069         void TestName::test()
01070     #define INTERNAL_CATCH_TEST_CASE_METHOD( ClassName, ... ) \
01071         INTERNAL_CATCH_TEST_CASE_METHOD2( INTERNAL_CATCH_UNIQUE_NAME( ____C_A_T_C_H____T_E_S_T____ ),
ClassName, __VA_ARGS__ )
01072
01073     #define INTERNAL_CATCH_REGISTER_TESTCASE( Function, ... ) \
01074         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01075         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01076         Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker( Function
), CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ __VA_ARGS__ } ); } /* NOLINT */ \
01077         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
01078
01079     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( TestName, TestFunc, Name, Tags, Signature, ... ) \
01080         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01081         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \

```

```

01088     CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
01089     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01090     INTERNAL_CATCH_DECLARE_SIG_TEST(TestFunc, INTERNAL_CATCH_REMOVE_PARENS(Signature));\
01091     namespace {\
01092     namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName){\
01093         INTERNAL_CATCH_TYPE_GEN\
01094         INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS(Signature))\
01095         INTERNAL_CATCH_NTTP_REG_GEN(TestFunc, INTERNAL_CATCH_REMOVE_PARENS(Signature))\
01096         template<typename...Types> \
01097         struct TestName{\
01098             TestName(){\
01099                 int index = 0; \
01100                 constexpr char const* tmpl_types[] = \
01101 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, __VA_ARGS__)};\
01102                 using expander = int[];\
01103                 (void)expander{(reg_test(Types{}, Catch::NameAndTags{ Name " - " + \
01104 std::string(tmpl_types[index]), Tags } ), index++)... };/* NOLINT */ \
01105             }\
01106             static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
01107                 TestName<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(__VA_ARGS__)>{};\
01108                 return 0;\
01109             }();\
01110         }\
01111         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01112         INTERNAL_CATCH_DEFINE_SIG_TEST(TestFunc, INTERNAL_CATCH_REMOVE_PARENS(Signature))
01113
01114 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01115     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE(Name, Tags, ...) \
01116     INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME( \
01117     __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( \
01118     __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__F_U_N_C_ ), Name, Tags, typename TestType, \
01119     __VA_ARGS__ )
01120 #else
01121     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE(Name, Tags, ...) \
01122     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME( \
01123     __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( \
01124     __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__F_U_N_C_ ), Name, Tags, typename TestType, \
01125     __VA_ARGS__ ) )
01126 #endif
01127
01128 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01129     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG(Name, Tags, Signature, ...) \
01130     INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME( \
01131     __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( \
01132     __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__F_U_N_C_ ), Name, Tags, Signature, __VA_ARGS__ )
01133 #else
01134     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG(Name, Tags, Signature, ...) \
01135     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME( \
01136     __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( \
01137     __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__F_U_N_C_ ), Name, Tags, Signature, __VA_ARGS__ ) )
01138 #endif
01139
01140 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(TestName, TestFuncName, Name, Tags, Signature, \
01141 TmplTypes, TypesList) \
01142     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01143     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01144     CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
01145     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01146     template<typename TestType> static void TestFuncName(); \
01147     namespace {\
01148     namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName) { \
01149         INTERNAL_CATCH_TYPE_GEN \
01150         INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS(Signature)) \
01151         template<typename... Types> \
01152         struct TestName { \
01153             void reg_tests() { \
01154                 int index = 0; \
01155                 using expander = int[]; \
01156                 constexpr char const* tmpl_types[] = \
01157 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TmplTypes))};\
01158                 constexpr char const* types_list[] = \
01159 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TypesList))};\
01160                 constexpr auto num_types = sizeof(types_list) / sizeof(types_list[0]);\
01161                 (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestFuncName<Types> ), \
01162 CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ Name " - " + \
01163 std::string(tmpl_types[index / num_types]) + "<" + std::string(types_list[index % num_types]) + ">", \
01164 Tags } ), index++)... };/* NOLINT */\
01165             } \
01166         }; \
01167         static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){ \
01168             using TestInit = typename create<TestName, \
01169 decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS(TmplTypes)>())>, \
01170 TypeList<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(INTERNAL_CATCH_REMOVE_PARENS(TypesList))>::type; \
01171             TestInit t; \
01172             t.reg_tests(); \
01173         }(); \
01174     } \
01175 }

```

```

01155         return 0;
01156     }();
01157 }
01158 }
01159 CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
01160 template<typename TestType>
01161 static void TestFuncName()
01162
01163 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01164 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE(Name, Tags, ...) \
01165     INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(INTERNAL_CATCH_UNIQUE_NAME(
01166         __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__ ), INTERNAL_CATCH_UNIQUE_NAME(
01167         __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__F_U_N_C__ ), Name, Tags, typename T, __VA_ARGS__ )
01168 #else
01169 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE(Name, Tags, ...) \
01170     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(
01171     INTERNAL_CATCH_UNIQUE_NAME( __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__ ),
01172     INTERNAL_CATCH_UNIQUE_NAME( __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__F_U_N_C__ ), Name, Tags,
01173     typename T, __VA_ARGS__ ) )
01174 #endif
01175
01176 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01177 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(Name, Tags, Signature, ...) \
01178     INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(INTERNAL_CATCH_UNIQUE_NAME(
01179         __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__ ), INTERNAL_CATCH_UNIQUE_NAME(
01180         __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__F_U_N_C__ ), Name, Tags, Signature, __VA_ARGS__ )
01181 #else
01182 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(Name, Tags, Signature, ...) \
01183     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(
01184     INTERNAL_CATCH_UNIQUE_NAME( __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__ ),
01185     INTERNAL_CATCH_UNIQUE_NAME( __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__F_U_N_C__ ), Name, Tags,
01186     Signature, __VA_ARGS__ ) )
01187 #endif
01188
01189 #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_2( TestName, TestFunc, Name, Tags, TmplList ) \
01190     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01191     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01192     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01193     template<typename TestType> static void TestFunc(); \
01194     namespace { \
01195     namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestName ) { \
01196     INTERNAL_CATCH_TYPE_GEN \
01197     template<typename... Types> \
01198     struct TestName { \
01199         void reg_tests() { \
01200             int index = 0; \
01201             using expander = int[]; \
01202             (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestFunc<Types> ), \
01203             CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ Name " - " + \
01204             std::string(INTERNAL_CATCH_STRINGIZE(TmplList)) + " - " + std::to_string(index), Tags } ), index++)... \
01205             };/* NOLINT */ \
01206         } \
01207     }; \
01208     static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = []() { \
01209         using TestInit = typename convert<TestName, TmplList>::type; \
01210         TestInit t; \
01211         t.reg_tests(); \
01212         return 0; \
01213     }(); \
01214     } \
01215     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01216     template<typename TestType> \
01217     static void TestFunc()
01218
01219 #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE(Name, Tags, TmplList) \
01220     INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME(
01221     __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__ ), INTERNAL_CATCH_UNIQUE_NAME(
01222     __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__F_U_N_C__ ), Name, Tags, TmplList )
01223
01224 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2( TestNameClass, TestName, ClassName, Name, \
01225     Tags, Signature, ... ) \
01226     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01227     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01228     CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
01229     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01230     namespace { \
01231     namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestName ) { \
01232     INTERNAL_CATCH_TYPE_GEN \
01233     INTERNAL_CATCH_NTTP_GEN( INTERNAL_CATCH_REMOVE_PARENS( Signature ) ) \
01234     INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD( TestName, ClassName, \
01235     INTERNAL_CATCH_REMOVE_PARENS( Signature ) ); \
01236     INTERNAL_CATCH_NTTP_REG_METHOD_GEN( TestName, INTERNAL_CATCH_REMOVE_PARENS( Signature ) ) \
01237     template<typename... Types> \
01238     struct TestNameClass { \
01239         TestNameClass() { \
01240             int index = 0; \
01241             constexpr char const* tmpl_types[] =

```

```

{CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, __VA_ARGS__)};\
01225     using expander = int[];\
01226     (void)expander{(reg_test(Types{}, #ClassName, Catch::NameAndTags{ Name " - " +
std::string(tmpl_types[index]), Tags } ), index++)... };/* NOLINT */ \
01227     }\
01228     };\
01229     static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
01230         TestNameClass<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(__VA_ARGS__)>();\
01231         return 0;\
01232     }();\
01233     }\
01234     }\
01235     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01236     INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD( TestName, INTERNAL_CATCH_REMOVE_PARENS( Signature ) )
01237
01238 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01239     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( ClassName, Name, Tags,... ) \
01240     INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME(
__C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__C_L_A_S_S__ ), INTERNAL_CATCH_UNIQUE_NAME(
__C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__ ), ClassName, Name, Tags, typename T, __VA_ARGS__ )
01241 #else
01242     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( ClassName, Name, Tags,... ) \
01243     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2(
INTERNAL_CATCH_UNIQUE_NAME( __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__C_L_A_S_S__ ),
INTERNAL_CATCH_UNIQUE_NAME( __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__ ), ClassName, Name, Tags,
typename T, __VA_ARGS__ ) )
01244 #endif
01245
01246 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01247     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature, ... ) \
01248     INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME(
__C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__C_L_A_S_S__ ), INTERNAL_CATCH_UNIQUE_NAME(
__C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__ ), ClassName, Name, Tags, Signature, __VA_ARGS__ )
01249 #else
01250     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature, ... ) \
01251     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2(
INTERNAL_CATCH_UNIQUE_NAME( __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__C_L_A_S_S__ ),
INTERNAL_CATCH_UNIQUE_NAME( __C_A_T_C_H__T_E_M_P_L_A_T_E__T_E_S_T__ ), ClassName, Name, Tags,
Signature, __VA_ARGS__ ) )
01252 #endif
01253
01254     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2( TestNameClass, TestName, ClassName,
Name, Tags, Signature, TmplTypes, TypesList )\
01255     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01256     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01257     CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
01258     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01259     template<typename TestType> \
01260     struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName <TestType>) { \
01261         void test();\
01262     };\
01263     namespace {\
01264     namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestNameClass ) {\
01265         INTERNAL_CATCH_TYPE_GEN \
01266         INTERNAL_CATCH_NTTP_GEN( INTERNAL_CATCH_REMOVE_PARENS( Signature ) )\
01267         template<typename... Types>\
01268         struct TestNameClass{\
01269             void reg_tests(){\
01270                 int index = 0;\
01271                 using expander = int[];\
01272                 constexpr char const* tmpl_types[] =
{CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TmplTypes))};\
01273                 constexpr char const* types_list[] =
{CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TypesList))};\
01274                 constexpr auto num_types = sizeof(types_list) / sizeof(types_list[0]);\
01275                 (void)expander{ Catch::AutoReg( Catch::makeTestInvoker( &TestName<Types>::test ),
CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ Name " - " + std::string(tmpl_types[index % num_types]) + ">", Tags } ), index++)... };/*
NOLINT */ \
01276                 }\
01277             };\
01278             static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
01279                 using TestInit = typename create<TestNameClass,
decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS(TmplTypes)>()),
TypeList<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(INTERNAL_CATCH_REMOVE_PARENS(TypesList))>::type;\
01280                 TestInit t;\
01281                 t.reg_tests();\
01282                 return 0;\
01283             }(); \
01284         }\
01285     }\
01286     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01287     template<typename TestType> \
01288     void TestName<TestType>::test()
01289
01290 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01291     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( ClassName, Name, Tags, ... )\

```

```

01292     INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME(
01293         _____C_A_T_C_H_____T_E_M_P_L_A_T_E_____T_E_S_T_____ ), INTERNAL_CATCH_UNIQUE_NAME(
01294         _____C_A_T_C_H_____T_E_M_P_L_A_T_E_____T_E_S_T_____F_U_N_C_____ ), ClassName, Name, Tags, typename T,
01295         __VA_ARGS__ )
01293 #else
01294     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( ClassName, Name, Tags, ... )\
01295     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2(
01296         INTERNAL_CATCH_UNIQUE_NAME( _____C_A_T_C_H_____T_E_M_P_L_A_T_E_____T_E_S_T_____ ),
01297         INTERNAL_CATCH_UNIQUE_NAME( _____C_A_T_C_H_____T_E_M_P_L_A_T_E_____T_E_S_T_____F_U_N_C_____ ), ClassName,
01298         Name, Tags, typename T, __VA_ARGS__ ) )
01296 #endif
01297
01298 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01299     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature,
01300     ... )\
01301     INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME(
01302         _____C_A_T_C_H_____T_E_M_P_L_A_T_E_____T_E_S_T_____ ), INTERNAL_CATCH_UNIQUE_NAME(
01303         _____C_A_T_C_H_____T_E_M_P_L_A_T_E_____T_E_S_T_____F_U_N_C_____ ), ClassName, Name, Tags, Signature,
01304         __VA_ARGS__ )
01301 #else
01302     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature,
01303     ... )\
01304     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2(
01305         INTERNAL_CATCH_UNIQUE_NAME( _____C_A_T_C_H_____T_E_M_P_L_A_T_E_____T_E_S_T_____ ),
01306         INTERNAL_CATCH_UNIQUE_NAME( _____C_A_T_C_H_____T_E_M_P_L_A_T_E_____T_E_S_T_____F_U_N_C_____ ), ClassName,
01307         Name, Tags, Signature, __VA_ARGS__ ) )
01304 #endif
01305
01306     #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_2( TestNameClass, TestName, ClassName, Name,
01307     Tags, TmplList) \
01308     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01309     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01310     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01311     template<typename TestType> \
01312     struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName <TestType>) { \
01313         void test(); \
01314     }; \
01315     namespace { \
01316         namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName){ \
01317             INTERNAL_CATCH_TYPE_GEN \
01318             template<typename... Types> \
01319             struct TestNameClass{ \
01320                 void reg_tests(){ \
01321                     int index = 0; \
01322                     using expander = int[]; \
01323                     (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestName<Types>::test ),
01324                     CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ Name " - " +
01325                     std::string(INTERNAL_CATCH_STRINGIZE(TmplList)) + " - " + std::to_string(index), Tags } ), index++)...
01326                     }; /* NOLINT */ \
01327                 } \
01328             }; \
01329             static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){ \
01330                 using TestInit = typename convert<TestNameClass, TmplList>::type; \
01331                 TestInit t; \
01332                 t.reg_tests(); \
01333                 return 0; \
01334             }(); \
01335         } \
01336     } \
01337     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01338     template<typename TestType> \
01339     void TestName<TestType>::test()
01336 #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD(ClassName, Name, Tags, TmplList) \
01337     INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME(
01338         _____C_A_T_C_H_____T_E_M_P_L_A_T_E_____T_E_S_T_____ ), INTERNAL_CATCH_UNIQUE_NAME(
01339         _____C_A_T_C_H_____T_E_M_P_L_A_T_E_____T_E_S_T_____F_U_N_C_____ ), ClassName, Name, Tags, TmplList )
01338
01339 // end catch_test_registry.h
01340 // start catch_capture.hpp
01341
01342 // start catch_assertionhandler.h
01343
01344 // start catch_assertioninfo.h
01345
01346 // start catch_result_type.h
01347
01348 namespace Catch {
01349
01350     // ResultWas::OfType enum
01351     struct ResultWas { enum OfType {
01352         Unknown = -1,
01353         Ok = 0,
01354         Info = 1,
01355         Warning = 2,
01356
01357         FailureBit = 0x10,
01358     };

```



```

01359         ExpressionFailed = FailureBit | 1,
01360         ExplicitFailure = FailureBit | 2,
01361
01362         Exception = 0x100 | FailureBit,
01363
01364         ThrewException = Exception | 1,
01365         DidntThrowException = Exception | 2,
01366
01367         FatalErrorCondition = 0x200 | FailureBit
01368     }; };
01369
01370     bool isOk( ResultWas::OfType resultType );
01371     bool isJustInfo( int flags );
01372
01373     // ResultDisposition::Flags enum
01374     struct ResultDisposition { enum Flags {
01375         Normal = 0x01,
01376
01377         ContinueOnFailure = 0x02,    // Failures fail test, but execution continues
01378         FalseTest = 0x04,           // Prefix expression with !
01379         SuppressFail = 0x08         // Failures are reported but do not fail the test
01380     }; };
01381
01382     ResultDisposition::Flags operator | ( ResultDisposition::Flags lhs, ResultDisposition::Flags rhs
01383 );
01384
01385     bool shouldContinueOnFailure( int flags );
01386     inline bool isFalseTest( int flags ) { return ( flags & ResultDisposition::FalseTest ) != 0; }
01387     bool shouldSuppressFailure( int flags );
01388
01389 } // end namespace Catch
01390
01391 // end catch_result_type.h
01392 namespace Catch {
01393
01394     struct AssertionInfo
01395     {
01396         StringRef macroName;
01397         SourceLineInfo lineInfo;
01398         StringRef capturedExpression;
01399         ResultDisposition::Flags resultDisposition;
01400
01401         // We want to delete this constructor but a compiler bug in 4.8 means
01402         // the struct is then treated as non-aggregate
01403         //AssertionInfo() = delete;
01404     };
01405
01406 } // end namespace Catch
01407
01408 // end catch_assertioninfo.h
01409 // start catch_decomposer.h
01410
01411 // start catch_tostring.h
01412
01413 #include <vector>
01414 #include <cstdint>
01415 #include <type_traits>
01416 #include <string>
01417 // start catch_stream.h
01418
01419 #include <iosfwd>
01420 #include <cstdint>
01421 #include <ostream>
01422
01423 namespace Catch {
01424
01425     std::ostream& cout();
01426     std::ostream& cerr();
01427     std::ostream& clog();
01428
01429     class StringRef;
01430
01431     struct IStream {
01432         virtual ~IStream();
01433         virtual std::ostream& stream() const = 0;
01434     };
01435
01436     auto makeStream( StringRef const &filename ) -> IStream const*;
01437
01438     class ReusableStringStream : NonCopyable {
01439     public:
01440         std::size_t m_index;
01441         std::ostream* m_oss;
01442         ReusableStringStream();
01443         ~ReusableStringStream();
01444

```



```

01445         auto str() const -> std::string;
01446
01447         template<typename T>
01448         auto operator « ( T const& value ) -> ReusableStringStream& {
01449             *m_oss « value;
01450             return *this;
01451         }
01452         auto get() -> std::ostream& { return *m_oss; }
01453     };
01454 }
01455
01456 // end catch_stream.h
01457 // start catch_interfaces_enum_values_registry.h
01458
01459 #include <vector>
01460
01461 namespace Catch {
01462
01463     namespace Detail {
01464         struct EnumInfo {
01465             StringRef m_name;
01466             std::vector<std::pair<int, StringRef>> m_values;
01467
01468             ~EnumInfo();
01469
01470             StringRef lookup( int value ) const;
01471         };
01472     } // namespace Detail
01473
01474     struct IMutableEnumValuesRegistry {
01475         virtual ~IMutableEnumValuesRegistry();
01476
01477         virtual Detail::EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums,
01478             std::vector<int> const& values ) = 0;
01479
01480         template<typename E>
01481         Detail::EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums,
01482             std::initializer_list<E> values ) {
01483             static_assert(sizeof(int) >= sizeof(E), "Cannot serialize enum to int");
01484             std::vector<int> intValues;
01485             intValues.reserve( values.size() );
01486             for( auto enumValue : values )
01487                 intValues.push_back( static_cast<int>( enumValue ) );
01488             return registerEnum( enumName, allEnums, intValues );
01489         }
01490     };
01491
01492 // end catch_interfaces_enum_values_registry.h
01493
01494 #ifdef CATCH_CONFIG_CPP17_STRING_VIEW
01495 #include <string_view>
01496 #endif
01497
01498 #ifdef __OBJC__
01499 // start catch_objc_arc.hpp
01500
01501 #import <Foundation/Foundation.h>
01502
01503 #ifdef __has_feature
01504 #define CATCH_ARC_ENABLED __has_feature(objc_arc)
01505 #else
01506 #define CATCH_ARC_ENABLED 0
01507 #endif
01508
01509 void arcSafeRelease( NSObject* obj );
01510 id performOptionalSelector( id obj, SEL sel );
01511
01512 #if !CATCH_ARC_ENABLED
01513 inline void arcSafeRelease( NSObject* obj ) {
01514     [obj release];
01515 }
01516 inline id performOptionalSelector( id obj, SEL sel ) {
01517     if( [obj respondsToSelector: sel] )
01518         return [obj performSelector: sel];
01519     return nil;
01520 }
01521 #define CATCH_UNSAFE_UNRETAINED
01522 #define CATCH_ARC_STRONG
01523 #else
01524 inline void arcSafeRelease( NSObject* ){}
01525 inline id performOptionalSelector( id obj, SEL sel ) {
01526     #ifdef __clang__
01527     #pragma clang diagnostic push
01528     #pragma clang diagnostic ignored "-Warc-performSelector-leaks"
01529     #endif

```

```

01530     if( [obj respondsToSelector: sel] )
01531         return [obj performSelector: sel];
01532 #ifdef __clang__
01533 #pragma clang diagnostic pop
01534 #endif
01535     return nil;
01536 }
01537 #define CATCH_UNSAFE_UNRETAINED __unsafe_unretained
01538 #define CATCH_ARC_STRONG __strong
01539 #endif
01540
01541 // end catch_objc_arc.hpp
01542 #endif
01543
01544 #ifdef _MSC_VER
01545 #pragma warning(push)
01546 #pragma warning(disable:4180) // We attempt to stream a function (address) by const&, which MSVC
                                complains about but is harmless
01547 #endif
01548
01549 namespace Catch {
01550     namespace Detail {
01551
01552         extern const std::string unprintableString;
01553
01554         std::string rawMemoryToString( const void *object, std::size_t size );
01555
01556         template<typename T>
01557         std::string rawMemoryToString( const T& object ) {
01558             return rawMemoryToString( &object, sizeof(object) );
01559         }
01560
01561         template<typename T>
01562         class IsStreamInsertable {
01563             template<typename Stream, typename U>
01564             static auto test(int)
01565                 -> decltype(std::declval<Stream&>() << std::declval<U>(), std::true_type());
01566
01567             template<typename, typename>
01568             static auto test(...) -> std::false_type;
01569
01570         public:
01571             static const bool value = decltype(test<std::ostream, const T&>(0))::value;
01572         };
01573
01574         template<typename E>
01575         std::string convertUnknownEnumToString( E e );
01576
01577         template<typename T>
01578         typename std::enable_if<
01579             !std::is_enum<T>::value && !std::is_base_of<std::exception, T>::value,
01580             std::string>::type convertUnstreamable( T const& ) {
01581             return Detail::unprintableString;
01582         }
01583
01584         template<typename T>
01585         typename std::enable_if<
01586             !std::is_enum<T>::value && std::is_base_of<std::exception, T>::value,
01587             std::string>::type convertUnstreamable( T const& ex ) {
01588             return ex.what();
01589         }
01590
01591         template<typename T>
01592         typename std::enable_if<
01593             std::is_enum<T>::value
01594             , std::string>::type convertUnstreamable( T const& value ) {
01595             return convertUnknownEnumToString( value );
01596         }
01597
01598 #if defined(_MANAGED)
01599         template<typename T>
01600         std::string clrReferenceToString( T^ ref ) {
01601             if (ref == nullptr)
01602                 return std::string("null");
01603             auto bytes = System::Text::Encoding::UTF8->GetBytes(ref->ToString());
01604             cli::pin_ptr<System::Byte> p = &bytes[0];
01605             return std::string(reinterpret_cast<char const*>(p), bytes->Length);
01606         }
01607 #endif
01608     } // namespace Detail
01609 } // namespace Catch
01610
01611 // If we decide for C++14, change these to enable_if_ts
01612 template <typename T, typename = void>
01613 struct StringMaker {
01614     template <typename Fake = T>
01615     static
01616     typename std::enable_if<:Catch::Detail::IsStreamInsertable<Fake>::value, std::string>::type

```

```

01617         convert(const Fake& value) {
01618             ReusableStringStream rss;
01619             // NB: call using the function-like syntax to avoid ambiguity with
01620             // user-defined templated operator« under clang.
01621             rss.operator«(value);
01622             return rss.str();
01623         }
01624
01625         template <typename Fake = T>
01626         static
01627         typename std::enable_if<!::Catch::Detail::IsStreamInsertable<Fake>::value, std::string>::type
01628         convert(const Fake& value) {
01629             #if !defined(CATCH_CONFIG_FALLBACK_STRINGIFIER)
01630                 return Detail::convertUnstreamable(value);
01631             #else
01632                 return CATCH_CONFIG_FALLBACK_STRINGIFIER(value);
01633             #endif
01634         }
01635     };
01636
01637     namespace Detail {
01638
01639         // This function dispatches all stringification requests inside of Catch.
01640         // Should be preferably called fully qualified, like ::Catch::Detail::stringify
01641         template <typename T>
01642         std::string stringify(const T& e) {
01643             return ::Catch::StringMaker<typename std::remove_cv<typename
std::remove_reference<T>::type>::type>::convert(e);
01644         }
01645
01646         template<typename E>
01647         std::string convertUnknownEnumToString( E e ) {
01648             return ::Catch::Detail::stringify(static_cast<typename std::underlying_type<E>::type>(e));
01649         }
01650
01651         #if defined(_MANAGED)
01652         template <typename T>
01653         std::string stringify( T^ e ) {
01654             return ::Catch::StringMaker<T^>::convert(e);
01655         }
01656         #endif
01657
01658     } // namespace Detail
01659
01660     // Some predefined specializations
01661
01662     template<>
01663     struct StringMaker<std::string> {
01664         static std::string convert(const std::string& str);
01665     };
01666
01667     #ifdef CATCH_CONFIG_CPP17_STRING_VIEW
01668     template<>
01669     struct StringMaker<std::string_view> {
01670         static std::string convert(std::string_view str);
01671     };
01672     #endif
01673
01674     template<>
01675     struct StringMaker<char const*> {
01676         static std::string convert(char const* str);
01677     };
01678     template<>
01679     struct StringMaker<char*> {
01680         static std::string convert(char* str);
01681     };
01682
01683     #ifdef CATCH_CONFIG_WCHAR
01684     template<>
01685     struct StringMaker<std::wstring> {
01686         static std::string convert(const std::wstring& wstr);
01687     };
01688
01689     #ifdef CATCH_CONFIG_CPP17_STRING_VIEW
01690     template<>
01691     struct StringMaker<std::wstring_view> {
01692         static std::string convert(std::wstring_view str);
01693     };
01694     #endif
01695
01696     template<>
01697     struct StringMaker<wchar_t const*> {
01698         static std::string convert(wchar_t const* str);
01699     };
01700     template<>
01701     struct StringMaker<wchar_t*> {
01702         static std::string convert(wchar_t* str);

```

```

01703     };
01704 #endif
01705
01706     // TBD: Should we use `strlen` to ensure that we don't go out of the buffer,
01707     //       while keeping string semantics?
01708     template<int SZ>
01709     struct StringMaker<char[SZ]> {
01710         static std::string convert(char const* str) {
01711             return ::Catch::Detail::stringify(std::string{ str });
01712         }
01713     };
01714     template<int SZ>
01715     struct StringMaker<signed char[SZ]> {
01716         static std::string convert(signed char const* str) {
01717             return ::Catch::Detail::stringify(std::string{ reinterpret_cast<char const*>(str) });
01718         }
01719     };
01720     template<int SZ>
01721     struct StringMaker<unsigned char[SZ]> {
01722         static std::string convert(unsigned char const* str) {
01723             return ::Catch::Detail::stringify(std::string{ reinterpret_cast<char const*>(str) });
01724         }
01725     };
01726
01727 #if defined(CATCH_CONFIG_CPP17_BYTE)
01728     template<>
01729     struct StringMaker<std::byte> {
01730         static std::string convert(std::byte value);
01731     };
01732 #endif // defined(CATCH_CONFIG_CPP17_BYTE)
01733     template<>
01734     struct StringMaker<int> {
01735         static std::string convert(int value);
01736     };
01737     template<>
01738     struct StringMaker<long> {
01739         static std::string convert(long value);
01740     };
01741     template<>
01742     struct StringMaker<long long> {
01743         static std::string convert(long long value);
01744     };
01745     template<>
01746     struct StringMaker<unsigned int> {
01747         static std::string convert(unsigned int value);
01748     };
01749     template<>
01750     struct StringMaker<unsigned long> {
01751         static std::string convert(unsigned long value);
01752     };
01753     template<>
01754     struct StringMaker<unsigned long long> {
01755         static std::string convert(unsigned long long value);
01756     };
01757
01758     template<>
01759     struct StringMaker<bool> {
01760         static std::string convert(bool b);
01761     };
01762
01763     template<>
01764     struct StringMaker<char> {
01765         static std::string convert(char c);
01766     };
01767     template<>
01768     struct StringMaker<signed char> {
01769         static std::string convert(signed char c);
01770     };
01771     template<>
01772     struct StringMaker<unsigned char> {
01773         static std::string convert(unsigned char c);
01774     };
01775
01776     template<>
01777     struct StringMaker<std::nullptr_t> {
01778         static std::string convert(std::nullptr_t);
01779     };
01780
01781     template<>
01782     struct StringMaker<float> {
01783         static std::string convert(float value);
01784         static int precision;
01785     };
01786
01787     template<>
01788     struct StringMaker<double> {
01789         static std::string convert(double value);

```

```

01790     static int precision;
01791 };
01792
01793 template <typename T>
01794 struct StringMaker<T*> {
01795     template <typename U>
01796     static std::string convert(U* p) {
01797         if (p) {
01798             return ::Catch::Detail::rawMemoryToString(p);
01799         } else {
01800             return "nullptr";
01801         }
01802     }
01803 };
01804
01805 template <typename R, typename C>
01806 struct StringMaker<R C::*> {
01807     static std::string convert(R C::* p) {
01808         if (p) {
01809             return ::Catch::Detail::rawMemoryToString(p);
01810         } else {
01811             return "nullptr";
01812         }
01813     }
01814 };
01815
01816 #if defined(_MANAGED)
01817 template <typename T>
01818 struct StringMaker<T^> {
01819     static std::string convert( T^ ref ) {
01820         return ::Catch::Detail::clrReferenceToString(ref);
01821     }
01822 };
01823 #endif
01824
01825 namespace Detail {
01826     template<typename InputIterator, typename Sentinel = InputIterator>
01827     std::string rangeToString(InputIterator first, Sentinel last) {
01828         ReusableStringStream rss;
01829         rss << "{ ";
01830         if (first != last) {
01831             rss << ::Catch::Detail::stringify(*first);
01832             for (++first; first != last; ++first)
01833                 rss << ", " << ::Catch::Detail::stringify(*first);
01834         }
01835         rss << " ";
01836         return rss.str();
01837     }
01838 }
01839
01840 #ifdef __OBJC__
01841 template<>
01842 struct StringMaker<NSString*> {
01843     static std::string convert(NSString * nsstring) {
01844         if (!nsstring)
01845             return "nil";
01846         return std::string(@"") + [nsstring UTF8String];
01847     }
01848 };
01849 template<>
01850 struct StringMaker<NSObject*> {
01851     static std::string convert(NSObject* nsObject) {
01852         return ::Catch::Detail::stringify([nsObject description]);
01853     }
01854 };
01855
01856 namespace Detail {
01857     inline std::string stringify( NSString* nsstring ) {
01858         return StringMaker<NSString*>::convert( nsstring );
01859     }
01860 }
01861 } // namespace Detail
01862 #endif // __OBJC__
01863
01864 } // namespace Catch
01865
01866 // Separate std-lib types stringification, so it can be selectively enabled
01867 // This means that we do not bring in
01868
01869 #if defined(CATCH_CONFIG_ENABLE_ALL_STRINGMAKERS)
01870 # define CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER
01871 # define CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER
01872 # define CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER
01873 # define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
01874 # define CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER
01875 #endif
01876 #endif
01877

```

```

01878 // Separate std::pair specialization
01879 #if defined(CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER)
01880 #include <utility>
01881 namespace Catch {
01882     template<typename T1, typename T2>
01883     struct StringMaker<std::pair<T1, T2> > {
01884         static std::string convert(const std::pair<T1, T2>& pair) {
01885             ReusableStringStream rss;
01886             rss << "{ "
01887                 << ::Catch::Detail::stringify(pair.first)
01888                 << ", "
01889                 << ::Catch::Detail::stringify(pair.second)
01890                 << " }";
01891             return rss.str();
01892         }
01893     };
01894 }
01895 #endif // CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER
01896
01897 #if defined(CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER) && defined(CATCH_CONFIG_CPP17_OPTIONAL)
01898 #include <optional>
01899 namespace Catch {
01900     template<typename T>
01901     struct StringMaker<std::optional<T> > {
01902         static std::string convert(const std::optional<T>& optional) {
01903             ReusableStringStream rss;
01904             if (optional.has_value()) {
01905                 rss << ::Catch::Detail::stringify(*optional);
01906             } else {
01907                 rss << "{ }";
01908             }
01909             return rss.str();
01910         }
01911     };
01912 }
01913 #endif // CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER
01914
01915 // Separate std::tuple specialization
01916 #if defined(CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER)
01917 #include <tuple>
01918 namespace Catch {
01919     namespace Detail {
01920         template<
01921             typename Tuple,
01922             std::size_t N = 0,
01923             bool = (N < std::tuple_size<Tuple>::value)
01924         >
01925         struct TupleElementPrinter {
01926             static void print(const Tuple& tuple, std::ostream& os) {
01927                 os << (N ? ", " : " ")
01928                     << ::Catch::Detail::stringify(std::get<N>(tuple));
01929                 TupleElementPrinter<Tuple, N + 1>::print(tuple, os);
01930             }
01931         };
01932
01933         template<
01934             typename Tuple,
01935             std::size_t N
01936         >
01937         struct TupleElementPrinter<Tuple, N, false> {
01938             static void print(const Tuple&, std::ostream&) {}
01939         };
01940
01941     }
01942
01943     template<typename ...Types>
01944     struct StringMaker<std::tuple<Types...> > {
01945         static std::string convert(const std::tuple<Types...>& tuple) {
01946             ReusableStringStream rss;
01947             rss << '{';
01948             Detail::TupleElementPrinter<std::tuple<Types...>::print(tuple, rss.get());
01949             rss << " }";
01950             return rss.str();
01951         }
01952     };
01953 }
01954 #endif // CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER
01955
01956 #if defined(CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER) && defined(CATCH_CONFIG_CPP17_VARIANT)
01957 #include <variant>
01958 namespace Catch {
01959     template<>
01960     struct StringMaker<std::monostate> {
01961         static std::string convert(const std::monostate&) {
01962             return "{ }";
01963         }
01964     };

```

```

01965
01966     template<typename... Elements>
01967     struct StringMaker<std::variant<Elements...> {
01968         static std::string convert(const std::variant<Elements...>& variant) {
01969             if (variant.valueless_by_exception()) {
01970                 return "{valueless variant}";
01971             } else {
01972                 return std::visit(
01973                     [](const auto& value) {
01974                         return ::Catch::Detail::stringify(value);
01975                     },
01976                     variant
01977                 );
01978             }
01979         };
01980     };
01981 }
01982 #endif // CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER
01983
01984 namespace Catch {
01985     // Import begin/ end from std here
01986     using std::begin;
01987     using std::end;
01988
01989     namespace detail {
01990         template <typename...>
01991         struct void_type {
01992             using type = void;
01993         };
01994
01995         template <typename T, typename = void>
01996         struct is_range_impl : std::false_type {
01997         };
01998
01999         template <typename T>
02000         struct is_range_impl<T, typename void_type<decltype(begin(std::declval<T>()))>::type> :
02001             std::true_type {
02002         };
02003     } // namespace detail
02004
02005     template <typename T>
02006     struct is_range : detail::is_range_impl<T> {
02007     };
02008
02009     #if defined(_MANAGED) // Managed types are never ranges
02010     template <typename T>
02011     struct is_range<T^> {
02012         static const bool value = false;
02013     };
02014 #endif
02015
02016     template<typename Range>
02017     std::string rangeToString( Range const& range ) {
02018         return ::Catch::Detail::rangeToString( begin( range ), end( range ) );
02019     }
02020
02021     // Handle vector<bool> specially
02022     template<typename Allocator>
02023     std::string rangeToString( std::vector<bool, Allocator> const& v ) {
02024         ReusableStringStream rss;
02025         rss << "{ ";
02026         bool first = true;
02027         for( bool b : v ) {
02028             if( first )
02029                 first = false;
02030             else
02031                 rss << ", ";
02032             rss << ::Catch::Detail::stringify( b );
02033         }
02034         rss << " }";
02035         return rss.str();
02036     }
02037
02038     template<typename R>
02039     struct StringMaker<R, typename std::enable_if<is_range<R>::value &&
02040         !::Catch::Detail::IsStreamInsertable<R>::value>::type> {
02041         static std::string convert( R const& range ) {
02042             return rangeToString( range );
02043         };
02044     };
02045
02046     template <typename T, int SZ>
02047     struct StringMaker<T[SZ]> {
02048         static std::string convert( T const(&arr)[SZ] ) {
02049             return rangeToString(arr);
02050         };
02051     };

```

```

02050
02051 } // namespace Catch
02052
02053 // Separate std::chrono::duration specialization
02054 #if defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
02055 #include <ctime>
02056 #include <ratio>
02057 #include <chrono>
02058
02059 namespace Catch {
02060
02061 template <class Ratio>
02062 struct ratio_string {
02063     static std::string symbol();
02064 };
02065
02066 template <class Ratio>
02067 std::string ratio_string<Ratio>::symbol() {
02068     Catch::ReusableStringStream rss;
02069     rss << '[' << Ratio::num << '/'
02070         << Ratio::den << ']';
02071     return rss.str();
02072 }
02073 template <>
02074 struct ratio_string<std::atto> {
02075     static std::string symbol();
02076 };
02077 template <>
02078 struct ratio_string<std::femto> {
02079     static std::string symbol();
02080 };
02081 template <>
02082 struct ratio_string<std::pico> {
02083     static std::string symbol();
02084 };
02085 template <>
02086 struct ratio_string<std::nano> {
02087     static std::string symbol();
02088 };
02089 template <>
02090 struct ratio_string<std::micro> {
02091     static std::string symbol();
02092 };
02093 template <>
02094 struct ratio_string<std::milli> {
02095     static std::string symbol();
02096 };
02097
02098 // std::chrono::duration specializations
02099 template<typename Value, typename Ratio>
02100 struct StringMaker<std::chrono::duration<Value, Ratio> {
02101     static std::string convert(std::chrono::duration<Value, Ratio> const& duration) {
02102         ReusableStringStream rss;
02103         rss << duration.count() << ' ' << ratio_string<Ratio>::symbol() << 's';
02104         return rss.str();
02105     }
02106 };
02107
02108 template<typename Value>
02109 struct StringMaker<std::chrono::duration<Value, std::ratio<1>> {
02110     static std::string convert(std::chrono::duration<Value, std::ratio<1> const& duration) {
02111         ReusableStringStream rss;
02112         rss << duration.count() << " s";
02113         return rss.str();
02114     }
02115 };
02116
02117 template<typename Value>
02118 struct StringMaker<std::chrono::duration<Value, std::ratio<60>> {
02119     static std::string convert(std::chrono::duration<Value, std::ratio<60> const& duration) {
02120         ReusableStringStream rss;
02121         rss << duration.count() << " m";
02122         return rss.str();
02123     }
02124 };
02125
02126 template<typename Value>
02127 struct StringMaker<std::chrono::duration<Value, std::ratio<3600>> {
02128     static std::string convert(std::chrono::duration<Value, std::ratio<3600> const& duration) {
02129         ReusableStringStream rss;
02130         rss << duration.count() << " h";
02131         return rss.str();
02132     }
02133 };
02134
02135 // std::chrono::time_point specialization
02136 // Generic time_point cannot be specialized, only std::chrono::time_point<system_clock>
02137 template<typename Clock, typename Duration>
02138 struct StringMaker<std::chrono::time_point<Clock, Duration> {
02139     static std::string convert(std::chrono::time_point<Clock, Duration> const& time_point) {

```



```

02139         return ::Catch::Detail::stringify(time_point.time_since_epoch()) + " since epoch";
02140     }
02141 };
02142 // std::chrono::time_point<system_clock> specialization
02143 template<typename Duration>
02144 struct StringMaker<std::chrono::time_point<std::chrono::system_clock, Duration> {
02145     static std::string convert(std::chrono::time_point<std::chrono::system_clock, Duration> const&
02146 time_point) {
02147         auto converted = std::chrono::system_clock::to_time_t(time_point);
02148 #ifdef _MSC_VER
02149         std::tm timeInfo = {};
02150         gmtime_s(&timeInfo, &converted);
02151 #else
02152         std::tm* timeInfo = std::gmtime(&converted);
02153 #endif
02154
02155         auto const timeStampSize = sizeof("2017-01-16T17:06:45Z");
02156         char timeStamp[timeStampSize];
02157         const char * const fmt = "%Y-%m-%dT%H:%M:%SZ";
02158
02159 #ifdef _MSC_VER
02160         std::strftime(timeStamp, timeStampSize, fmt, &timeInfo);
02161 #else
02162         std::strftime(timeStamp, timeStampSize, fmt, timeInfo);
02163 #endif
02164         return std::string(timeStamp);
02165     }
02166 };
02167 }
02168 #endif // CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
02169
02170 #define INTERNAL_CATCH_REGISTER_ENUM( enumName, ... ) \
02171 namespace Catch { \
02172     template<> struct StringMaker<enumName> { \
02173         static std::string convert( enumName value ) { \
02174             static const auto& enumInfo = \
02175 ::Catch::getMutableRegistryHub().getMutableEnumValuesRegistry().registerEnum( #enumName, #__VA_ARGS__, \
02176 { __VA_ARGS__ } ); \
02177             return static_cast<std::string>(enumInfo.lookup( static_cast<int>( value ) )); \
02178         } \
02179     }; \
02180 #define CATCH_REGISTER_ENUM( enumName, ... ) INTERNAL_CATCH_REGISTER_ENUM( enumName, __VA_ARGS__ )
02181
02182 #ifdef _MSC_VER
02183 #pragma warning(pop)
02184 #endif
02185
02186 // end catch_tostring.h
02187 #include <iosfwd>
02188
02189 #ifdef _MSC_VER
02190 #pragma warning(push)
02191 #pragma warning(disable:4389) // '==' : signed/unsigned mismatch
02192 #pragma warning(disable:4018) // more "signed/unsigned mismatch"
02193 #pragma warning(disable:4312) // Converting int to T* using reinterpret_cast (issue on x64 platform)
02194 #pragma warning(disable:4180) // qualifier applied to function type has no meaning
02195 #pragma warning(disable:4800) // Forcing result to true or false
02196 #endif
02197
02198 namespace Catch {
02199
02200     struct ITransientExpression {
02201         auto isBinaryExpression() const -> bool { return m_isBinaryExpression; }
02202         auto getResult() const -> bool { return m_result; }
02203         virtual void streamReconstructedExpression( std::ostream &os ) const = 0;
02204
02205         ITransientExpression( bool isBinaryExpression, bool result )
02206             : m_isBinaryExpression( isBinaryExpression ),
02207               m_result( result )
02208         {}
02209
02210         // We don't actually need a virtual destructor, but many static analysers
02211         // complain if it's not here :-|
02212         virtual ~ITransientExpression();
02213
02214         bool m_isBinaryExpression;
02215         bool m_result;
02216     };
02217
02218     void formatReconstructedExpression( std::ostream &os, std::string const& lhs, StringRef op,
02219 std::string const& rhs );
02220
02221     template<typename LhsT, typename RhsT>

```

```

02222     class BinaryExpr : public ITransientExpression {
02223         LhsT m_lhs;
02224         StringRef m_op;
02225         RhsT m_rhs;
02226
02227         void streamReconstructedExpression( std::ostream &os ) const override {
02228             formatReconstructedExpression
02229                 ( os, Catch::Detail::stringify( m_lhs ), m_op, Catch::Detail::stringify( m_rhs )
02230 );
02231     }
02232 public:
02233     BinaryExpr( bool comparisonResult, LhsT lhs, StringRef op, RhsT rhs )
02234         : ITransientExpression{ true, comparisonResult },
02235           m_lhs( lhs ),
02236           m_op( op ),
02237           m_rhs( rhs )
02238     {}
02239
02240     template<typename T>
02241     auto operator && ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02242         static_assert(always_false<T>::value,
02243             "chained comparisons are not supported inside assertions, "
02244             "wrap the expression inside parentheses, or decompose it");
02245     }
02246
02247     template<typename T>
02248     auto operator || ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02249         static_assert(always_false<T>::value,
02250             "chained comparisons are not supported inside assertions, "
02251             "wrap the expression inside parentheses, or decompose it");
02252     }
02253
02254     template<typename T>
02255     auto operator == ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02256         static_assert(always_false<T>::value,
02257             "chained comparisons are not supported inside assertions, "
02258             "wrap the expression inside parentheses, or decompose it");
02259     }
02260
02261     template<typename T>
02262     auto operator != ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02263         static_assert(always_false<T>::value,
02264             "chained comparisons are not supported inside assertions, "
02265             "wrap the expression inside parentheses, or decompose it");
02266     }
02267
02268     template<typename T>
02269     auto operator > ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02270         static_assert(always_false<T>::value,
02271             "chained comparisons are not supported inside assertions, "
02272             "wrap the expression inside parentheses, or decompose it");
02273     }
02274
02275     template<typename T>
02276     auto operator < ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02277         static_assert(always_false<T>::value,
02278             "chained comparisons are not supported inside assertions, "
02279             "wrap the expression inside parentheses, or decompose it");
02280     }
02281
02282     template<typename T>
02283     auto operator >= ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02284         static_assert(always_false<T>::value,
02285             "chained comparisons are not supported inside assertions, "
02286             "wrap the expression inside parentheses, or decompose it");
02287     }
02288
02289     template<typename T>
02290     auto operator <= ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02291         static_assert(always_false<T>::value,
02292             "chained comparisons are not supported inside assertions, "
02293             "wrap the expression inside parentheses, or decompose it");
02294     }
02295 };
02296
02297 template<typename LhsT>
02298 class UnaryExpr : public ITransientExpression {
02299     LhsT m_lhs;
02300
02301     void streamReconstructedExpression( std::ostream &os ) const override {
02302         os << Catch::Detail::stringify( m_lhs );
02303     }
02304 public:
02305     explicit UnaryExpr( LhsT lhs )
02306         : ITransientExpression{ false, static_cast<bool>(lhs) },
02307

```

```

02308         m_lhs( lhs )
02309     {}
02310 };
02311
02312 // Specialised comparison functions to handle equality comparisons between ints and pointers (NULL
deduces as an int)
02313 template<typename LhsT, typename RhsT>
02314 auto compareEqual( LhsT const& lhs, RhsT const& rhs ) -> bool { return static_cast<bool>(lhs ==
rhs); }
02315 template<typename T>
02316 auto compareEqual( T* const& lhs, int rhs ) -> bool { return lhs == reinterpret_cast<void const*>(
rhs ); }
02317 template<typename T>
02318 auto compareEqual( T* const& lhs, long rhs ) -> bool { return lhs == reinterpret_cast<void
const*>( rhs ); }
02319 template<typename T>
02320 auto compareEqual( int lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs )
== rhs; }
02321 template<typename T>
02322 auto compareEqual( long lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs )
== rhs; }
02323
02324 template<typename LhsT, typename RhsT>
02325 auto compareNotEqual( LhsT const& lhs, RhsT&& rhs ) -> bool { return static_cast<bool>(lhs !=
rhs); }
02326 template<typename T>
02327 auto compareNotEqual( T* const& lhs, int rhs ) -> bool { return lhs != reinterpret_cast<void
const*>( rhs ); }
02328 template<typename T>
02329 auto compareNotEqual( T* const& lhs, long rhs ) -> bool { return lhs != reinterpret_cast<void
const*>( rhs ); }
02330 template<typename T>
02331 auto compareNotEqual( int lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs
) != rhs; }
02332 template<typename T>
02333 auto compareNotEqual( long lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>(
lhs ) != rhs; }
02334
02335 template<typename LhsT>
02336 class ExprLhs {
02337     LhsT m_lhs;
02338 public:
02339     explicit ExprLhs( LhsT lhs ) : m_lhs( lhs ) {}
02340
02341     template<typename RhsT>
02342     auto operator == ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02343         return { compareEqual( m_lhs, rhs ), m_lhs, "==", rhs };
02344     }
02345     auto operator == ( bool rhs ) -> BinaryExpr<LhsT, bool> const {
02346         return { m_lhs == rhs, m_lhs, "==", rhs };
02347     }
02348
02349     template<typename RhsT>
02350     auto operator != ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02351         return { compareNotEqual( m_lhs, rhs ), m_lhs, "!=", rhs };
02352     }
02353     auto operator != ( bool rhs ) -> BinaryExpr<LhsT, bool> const {
02354         return { m_lhs != rhs, m_lhs, "!=", rhs };
02355     }
02356
02357     template<typename RhsT>
02358     auto operator > ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02359         return { static_cast<bool>(m_lhs > rhs), m_lhs, ">", rhs };
02360     }
02361     template<typename RhsT>
02362     auto operator < ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02363         return { static_cast<bool>(m_lhs < rhs), m_lhs, "<", rhs };
02364     }
02365     template<typename RhsT>
02366     auto operator >= ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02367         return { static_cast<bool>(m_lhs >= rhs), m_lhs, ">=", rhs };
02368     }
02369     template<typename RhsT>
02370     auto operator <= ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02371         return { static_cast<bool>(m_lhs <= rhs), m_lhs, "<=", rhs };
02372     }
02373     template <typename RhsT>
02374     auto operator | (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
02375         return { static_cast<bool>(m_lhs | rhs), m_lhs, "|", rhs };
02376     }
02377     template <typename RhsT>
02378     auto operator & (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
02379         return { static_cast<bool>(m_lhs & rhs), m_lhs, "&", rhs };
02380     }
02381     template <typename RhsT>
02382     auto operator ^ (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
02383         return { static_cast<bool>(m_lhs ^ rhs), m_lhs, "^", rhs };

```

```

02384     }
02385
02386     template<typename RhsT>
02387     auto operator && ( RhsT const& ) -> BinaryExpr<LhsT, RhsT const&> const {
02388         static_assert(always_false<RhsT>::value,
02389             "operator&& is not supported inside assertions, "
02390             "wrap the expression inside parentheses, or decompose it");
02391     }
02392
02393     template<typename RhsT>
02394     auto operator || ( RhsT const& ) -> BinaryExpr<LhsT, RhsT const&> const {
02395         static_assert(always_false<RhsT>::value,
02396             "operator|| is not supported inside assertions, "
02397             "wrap the expression inside parentheses, or decompose it");
02398     }
02399
02400     auto makeUnaryExpr() const -> UnaryExpr<LhsT> {
02401         return UnaryExpr<LhsT>{ m_lhs };
02402     }
02403 };
02404
02405 void handleExpression( ITransientExpression const& expr );
02406
02407 template<typename T>
02408 void handleExpression( ExprLhs<T> const& expr ) {
02409     handleExpression( expr.makeUnaryExpr() );
02410 }
02411
02412 struct Decomposer {
02413     template<typename T>
02414     auto operator <= ( T const& lhs ) -> ExprLhs<T const&> {
02415         return ExprLhs<T const&>{ lhs };
02416     }
02417
02418     auto operator <=( bool value ) -> ExprLhs<bool> {
02419         return ExprLhs<bool>{ value };
02420     }
02421 };
02422
02423 } // end namespace Catch
02424
02425 #ifdef _MSC_VER
02426 #pragma warning(pop)
02427 #endif
02428
02429 // end catch_decomposer.h
02430 // start catch_interfaces_capture.h
02431
02432 #include <string>
02433 #include <chrono>
02434
02435 namespace Catch {
02436
02437     class AssertionResult;
02438     struct AssertionInfo;
02439     struct SectionInfo;
02440     struct SectionEndInfo;
02441     struct MessageInfo;
02442     struct MessageBuilder;
02443     struct Counts;
02444     struct AssertionReaction;
02445     struct SourceLineInfo;
02446
02447     struct ITransientExpression;
02448     struct IGeneratorTracker;
02449
02450     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
02451     struct BenchmarkInfo;
02452     template <typename Duration = std::chrono::duration<double, std::nano>
02453     struct BenchmarkStats;
02454     #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
02455
02456     struct IResultCapture {
02457         virtual ~IResultCapture();
02458
02459         virtual bool sectionStarted( SectionInfo const& sectionInfo,
02460                                     Counts& assertions ) = 0;
02461         virtual void sectionEnded( SectionEndInfo const& endInfo ) = 0;
02462         virtual void sectionEndedEarly( SectionEndInfo const& endInfo ) = 0;
02463
02464         virtual auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo
02465     ) -> IGeneratorTracker& = 0;
02466
02467     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
02468         virtual void benchmarkPreparing( std::string const& name ) = 0;
02469         virtual void benchmarkStarting( BenchmarkInfo const& info ) = 0;

```

```

02470     virtual void benchmarkEnded( BenchmarkStats<> const& stats ) = 0;
02471     virtual void benchmarkFailed( std::string const& error ) = 0;
02472 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
02473
02474     virtual void pushScopedMessage( MessageInfo const& message ) = 0;
02475     virtual void popScopedMessage( MessageInfo const& message ) = 0;
02476
02477     virtual void emplaceUnscopedMessage( MessageBuilder const& builder ) = 0;
02478
02479     virtual void handleFatalErrorCondition( StringRef message ) = 0;
02480
02481     virtual void handleExpr
02482     (   AssertionInfo const& info,
02483         ITransientExpression const& expr,
02484         AssertionReaction& reaction ) = 0;
02485     virtual void handleMessage
02486     (   AssertionInfo const& info,
02487         ResultWas::OfType resultType,
02488         StringRef const& message,
02489         AssertionReaction& reaction ) = 0;
02490     virtual void handleUnexpectedExceptionNotThrown
02491     (   AssertionInfo const& info,
02492         AssertionReaction& reaction ) = 0;
02493     virtual void handleUnexpectedInflightException
02494     (   AssertionInfo const& info,
02495         std::string const& message,
02496         AssertionReaction& reaction ) = 0;
02497     virtual void handleIncomplete
02498     (   AssertionInfo const& info ) = 0;
02499     virtual void handleNonExpr
02500     (   AssertionInfo const& info,
02501         ResultWas::OfType resultType,
02502         AssertionReaction& reaction ) = 0;
02503
02504     virtual bool lastAssertionPassed() = 0;
02505     virtual void assertionPassed() = 0;
02506
02507     // Deprecated, do not use:
02508     virtual std::string getCurrentTestName() const = 0;
02509     virtual const AssertionResult* getLastResult() const = 0;
02510     virtual void exceptionEarlyReported() = 0;
02511 };
02512
02513 IResultCapture& getResultCapture();
02514 }
02515
02516 // end catch_interfaces_capture.h
02517 namespace Catch {
02518
02519     struct TestFailureException{};
02520     struct AssertionResultData;
02521     struct IResultCapture;
02522     class RunContext;
02523
02524     class LazyExpression {
02525     friend class AssertionHandler;
02526     friend struct AssertionStats;
02527     friend class RunContext;
02528
02529     ITransientExpression const* m_transientExpression = nullptr;
02530     bool m_isNegated;
02531     public:
02532     LazyExpression( bool isNegated );
02533     LazyExpression( LazyExpression const& other );
02534     LazyExpression& operator = ( LazyExpression const& ) = delete;
02535
02536     explicit operator bool() const;
02537
02538     friend auto operator << ( std::ostream& os, LazyExpression const& lazyExpr ) -> std::ostream&;
02539 };
02540
02541 struct AssertionReaction {
02542     bool shouldDebugBreak = false;
02543     bool shouldThrow = false;
02544 };
02545
02546 class AssertionHandler {
02547     AssertionInfo m_assertionInfo;
02548     AssertionReaction m_reaction;
02549     bool m_completed = false;
02550     IResultCapture& m_resultCapture;
02551     public:
02552     AssertionHandler
02553     (   StringRef const& macroName,
02554         SourceLineInfo const& lineInfo,
02555         StringRef capturedExpression,

```

```

02557         ResultDisposition::Flags resultDisposition );
02558     ~AssertionHandler() {
02559         if ( !m_completed ) {
02560             m_resultCapture.handleIncomplete( m_assertionInfo );
02561         }
02562     }
02563
02564     template<typename T>
02565     void handleExpr( ExprLhs<T> const& expr ) {
02566         handleExpr( expr.makeUnaryExpr() );
02567     }
02568     void handleExpr( ITransientExpression const& expr );
02569
02570     void handleMessage(ResultWas::OfType resultType, StringRef const& message);
02571
02572     void handleExceptionThrownAsExpected();
02573     void handleUnexpectedExceptionNotThrown();
02574     void handleExceptionNotThrownAsExpected();
02575     void handleThrowingCallSkipped();
02576     void handleUnexpectedInflightException();
02577
02578     void complete();
02579     void setCompleted();
02580
02581     // query
02582     auto allowThrows() const -> bool;
02583 };
02584
02585 void handleExceptionMatchExpr( AssertionHandler& handler, std::string const& str, StringRef const&
matcherString );
02586 } // namespace Catch
02587
02588 // end catch_assertionhandler.h
02589 // start catch_message.h
02590
02591 #include <string>
02592 #include <vector>
02593
02594 namespace Catch {
02595
02596     struct MessageInfo {
02597         MessageInfo( StringRef const& _macroName,
02598                     SourceLineInfo const& _lineInfo,
02599                     ResultWas::OfType _type );
02600
02601         StringRef macroName;
02602         std::string message;
02603         SourceLineInfo lineInfo;
02604         ResultWas::OfType type;
02605         unsigned int sequence;
02606
02607         bool operator == ( MessageInfo const& other ) const;
02608         bool operator < ( MessageInfo const& other ) const;
02609     private:
02610         static unsigned int globalCount;
02611     };
02612
02613     struct MessageStream {
02614
02615         template<typename T>
02616         MessageStream& operator << ( T const& value ) {
02617             m_stream << value;
02618             return *this;
02619         }
02620
02621         ReusableStringStream m_stream;
02622     };
02623
02624     struct MessageBuilder : MessageStream {
02625         MessageBuilder( StringRef const& macroName,
02626                        SourceLineInfo const& lineInfo,
02627                        ResultWas::OfType type );
02628
02629         template<typename T>
02630         MessageBuilder& operator << ( T const& value ) {
02631             m_stream << value;
02632             return *this;
02633         }
02634
02635         MessageInfo m_info;
02636     };
02637
02638     class ScopedMessage {
02639     public:
02640         explicit ScopedMessage( MessageBuilder const& builder );
02641         ScopedMessage( ScopedMessage& duplicate ) = delete;
02642     };

```

```

02643         ScopedMessage( ScopedMessage&& old );
02644         ~ScopedMessage();
02645
02646         MessageInfo m_info;
02647         bool m_moved;
02648     };
02649
02650     class Capturer {
02651     public:
02652         std::vector<MessageInfo> m_messages;
02653         IResultCapture& m_resultCapture = getResultCapture();
02654         size_t m_captured = 0;
02655         Capturer( StringRef macroName, SourceLineInfo const& lineInfo, ResultWas::OfType resultType,
02656                 StringRef names );
02657         ~Capturer();
02658
02659         void captureValue( size_t index, std::string const& value );
02660
02661         template<typename T>
02662         void captureValues( size_t index, T const& value ) {
02663             captureValue( index, Catch::Detail::stringify( value ) );
02664         }
02665
02666         template<typename T, typename... Ts>
02667         void captureValues( size_t index, T const& value, Ts const&... values ) {
02668             captureValue( index, Catch::Detail::stringify(value) );
02669             captureValues( index+1, values... );
02670         }
02671     };
02672 } // end namespace Catch
02673
02674 // end catch_message.h
02675 #if !defined(CATCH_CONFIG_DISABLE)
02676
02677 #if !defined(CATCH_CONFIG_DISABLE_STRINGIFICATION)
02678     #define CATCH_INTERNAL_STRINGIFY(...) #__VA_ARGS__
02679 #else
02680     #define CATCH_INTERNAL_STRINGIFY(...) "Disabled by CATCH_CONFIG_DISABLE_STRINGIFICATION"
02681 #endif
02682
02683 #if defined(CATCH_CONFIG_FAST_COMPILE) || defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
02684
02685     // Another way to speed-up compilation is to omit local try-catch for REQUIRE*
02686     // macros.
02687     #define INTERNAL_CATCH_TRY
02688     #define INTERNAL_CATCH_CATCH( handler )
02689
02690 #else // !defined(CATCH_CONFIG_FAST_COMPILE) && !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
02691     #define INTERNAL_CATCH_TRY try
02692     #define INTERNAL_CATCH_CATCH( handler ) catch(...) { handler.handleUnexpectedInflightException(); }
02693 #endif
02694
02695 #define INTERNAL_CATCH_REACT( handler ) handler.complete();
02696
02697 #define INTERNAL_CATCH_TEST( macroName, resultDisposition, ... ) \
02698     do { \
02699         CATCH_INTERNAL_IGNORE_BUT_WARN(__VA_ARGS__); \
02700         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
02701             CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition ); \
02702         INTERNAL_CATCH_TRY { \
02703             CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
02704             CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS \
02705             catchAssertionHandler.handleExpr( Catch::Decomposer() <= __VA_ARGS__ ); \
02706             CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
02707         } INTERNAL_CATCH_CATCH( catchAssertionHandler ) \
02708         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02709     } while( (void)0, (false) && static_cast<bool>( !(__VA_ARGS__) ) )
02710
02711 #define INTERNAL_CATCH_IF( macroName, resultDisposition, ... ) \
02712     INTERNAL_CATCH_TEST( macroName, resultDisposition, __VA_ARGS__ ); \
02713     if( Catch::getResultCapture().lastAssertionPassed() )
02714
02715 #define INTERNAL_CATCH_ELSE( macroName, resultDisposition, ... ) \
02716     INTERNAL_CATCH_TEST( macroName, resultDisposition, __VA_ARGS__ ); \
02717     if( !Catch::getResultCapture().lastAssertionPassed() )
02718
02719 #define INTERNAL_CATCH_NO_THROW( macroName, resultDisposition, ... ) \
02720     do { \
02721         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
02722             CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition ); \
02723         try { \
02724             static_cast<void>(__VA_ARGS__); \
02725             catchAssertionHandler.handleExceptionNotThrownAsExpected(); \
02726         } \
02727     } \

```

```

02732         catch( ... ) { \
02733             catchAssertionHandler.handleUnexpectedInflightException(); \
02734         } \
02735         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02736     } while( false )
02737
02739 #define INTERNAL_CATCH_THROWS( macroName, resultDisposition, ... ) \
02740     do { \
02741         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
02742             CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition); \
02743         if( catchAssertionHandler.allowThrows() ) \
02744             try { \
02745                 static_cast<void>(__VA_ARGS__); \
02746                 catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
02747             } \
02748             catch( ... ) { \
02749                 catchAssertionHandler.handleExceptionThrownAsExpected(); \
02750             } \
02751         else \
02752             catchAssertionHandler.handleThrowingCallSkipped(); \
02753         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02754     } while( false )
02755
02756 #define INTERNAL_CATCH_THROWS_AS( macroName, exceptionType, resultDisposition, expr ) \
02757     do { \
02758         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
02759             CATCH_INTERNAL_STRINGIFY(expr) ", " CATCH_INTERNAL_STRINGIFY(exceptionType), resultDisposition ); \
02760         if( catchAssertionHandler.allowThrows() ) \
02761             try { \
02762                 static_cast<void>(expr); \
02763                 catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
02764             } \
02765             catch( exceptionType const& ) { \
02766                 catchAssertionHandler.handleExceptionThrownAsExpected(); \
02767             } \
02768             catch( ... ) { \
02769                 catchAssertionHandler.handleUnexpectedInflightException(); \
02770             } \
02771         else \
02772             catchAssertionHandler.handleThrowingCallSkipped(); \
02773         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02774     } while( false )
02775
02776 #define INTERNAL_CATCH_MSG( macroName, messageType, resultDisposition, ... ) \
02777     do { \
02778         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
02779             Catch::StringRef(), resultDisposition ); \
02780         catchAssertionHandler.handleMessage( messageType, ( Catch::MessageStream() << __VA_ARGS__ + \
02781             ::Catch::StreamEndStop() ).m_stream.str() ); \
02782         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02783     } while( false )
02784
02785 #define INTERNAL_CATCH_CAPTURE( varName, macroName, ... ) \
02786     auto varName = Catch::Capturer( macroName, CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info, \
02787         #__VA_ARGS__ ); \
02788     varName.captureValues( 0, __VA_ARGS__ )
02789
02790 #define INTERNAL_CATCH_INFO( macroName, log ) \
02791     Catch::ScopedMessage INTERNAL_CATCH_UNIQUE_NAME( scopedMessage )( Catch::MessageBuilder( \
02792         macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info ) << log );
02793
02794 #define INTERNAL_CATCH_UNSCOPED_INFO( macroName, log ) \
02795     Catch::getResultCapture().emplaceUnscopedMessage( Catch::MessageBuilder( macroName##_catch_sr, \
02796         CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info ) << log )
02797
02798 // Although this is matcher-based, it can be used with just a string
02799 #define INTERNAL_CATCH_THROWS_STR_MATCHES( macroName, resultDisposition, matcher, ... ) \
02800     do { \
02801         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
02802             CATCH_INTERNAL_STRINGIFY(__VA_ARGS__) ", " CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
02803         if( catchAssertionHandler.allowThrows() ) \
02804             try { \
02805                 static_cast<void>(__VA_ARGS__); \
02806                 catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
02807             } \
02808             catch( ... ) { \
02809                 Catch::handleExceptionMatchExpr( catchAssertionHandler, matcher, #matcher##_catch_sr \
02810 ); \
02811             } \
02812         else \
02813             catchAssertionHandler.handleThrowingCallSkipped(); \
02814         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02815     } while( false )
02816
02817 #endif // CATCH_CONFIG_DISABLE
02818
02819 // end catch_capture.hpp

```



```

02817 // start catch_section.h
02818
02819 // start catch_section_info.h
02820
02821 // start catch_totals.h
02822
02823 #include <cstdint>
02824
02825 namespace Catch {
02826
02827     struct Counts {
02828         Counts operator - ( Counts const& other ) const;
02829         Counts& operator += ( Counts const& other );
02830
02831         std::size_t total() const;
02832         bool allPassed() const;
02833         bool allOk() const;
02834
02835         std::size_t passed = 0;
02836         std::size_t failed = 0;
02837         std::size_t failedButOk = 0;
02838     };
02839
02840     struct Totals {
02841
02842         Totals operator - ( Totals const& other ) const;
02843         Totals& operator += ( Totals const& other );
02844
02845         Totals delta( Totals const& prevTotals ) const;
02846
02847         int error = 0;
02848         Counts assertions;
02849         Counts testCases;
02850     };
02851 }
02852
02853 // end catch_totals.h
02854 #include <string>
02855
02856 namespace Catch {
02857
02858     struct SectionInfo {
02859         SectionInfo
02860             ( SourceLineInfo const& _lineInfo,
02861               std::string const& _name );
02862
02863         // Deprecated
02864         SectionInfo
02865             ( SourceLineInfo const& _lineInfo,
02866               std::string const& _name,
02867               std::string const& ) : SectionInfo( _lineInfo, _name ) {}
02868
02869         std::string name;
02870         std::string description; // !Deprecated: this will always be empty
02871         SourceLineInfo lineInfo;
02872     };
02873
02874     struct SectionEndInfo {
02875         SectionInfo sectionInfo;
02876         Counts prevAssertions;
02877         double durationInSeconds;
02878     };
02879
02880 } // end namespace Catch
02881
02882 // end catch_section_info.h
02883 // start catch_timer.h
02884
02885 #include <cstdint>
02886
02887 namespace Catch {
02888
02889     auto getCurrentNanosecondsSinceEpoch() -> uint64_t;
02890     auto getEstimatedClockResolution() -> uint64_t;
02891
02892     class Timer {
02893     public:
02894         uint64_t m_nanoseconds = 0;
02895         void start();
02896         auto getElapsedNanoseconds() const -> uint64_t;
02897         auto getElapsedMicroseconds() const -> uint64_t;
02898         auto getElapsedMilliseconds() const -> unsigned int;
02899         auto getElapsedSeconds() const -> double;
02900     };
02901
02902 } // namespace Catch
02903

```

```

02904 // end catch_timer.h
02905 #include <string>
02906
02907 namespace Catch {
02908     class Section : NonCopyable {
02909     public:
02910         Section( SectionInfo const& info );
02911         ~Section();
02912
02913         // This indicates whether the section should be executed or not
02914         explicit operator bool() const;
02915
02916     private:
02917         SectionInfo m_info;
02918
02919         std::string m_name;
02920         Counts m_assertions;
02921         bool m_sectionIncluded;
02922         Timer m_timer;
02923     };
02924 };
02925
02926 } // end namespace Catch
02927
02928 #define INTERNAL_CATCH_SECTION( ... ) \
02929     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
02930     CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
02931     if( Catch::Section const& INTERNAL_CATCH_UNIQUE_NAME( catch_internal_Section ) = \
02932         Catch::SectionInfo( CATCH_INTERNAL_LINEINFO, __VA_ARGS__ ) ) \
02933         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
02934
02935 #define INTERNAL_CATCH_DYNAMIC_SECTION( ... ) \
02936     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
02937     CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
02938     if( Catch::Section const& INTERNAL_CATCH_UNIQUE_NAME( catch_internal_Section ) = \
02939         Catch::SectionInfo( CATCH_INTERNAL_LINEINFO, (Catch::ReusableStringStream() << __VA_ARGS__).str() ) ) \
02940         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
02941
02942 // end catch_section.h
02943 // start catch_interfaces_exception.h
02944 // start catch_interfaces_registry_hub.h
02945 #include <string>
02946 #include <memory>
02947
02948 namespace Catch {
02949
02950     class TestCase;
02951     struct ITestCaseRegistry;
02952     struct IExceptionTranslatorRegistry;
02953     struct IExceptionTranslator;
02954     struct IReporterRegistry;
02955     struct IReporterFactory;
02956     struct ITagAliasRegistry;
02957     struct IMutableEnumValuesRegistry;
02958
02959     class StartupExceptionRegistry;
02960
02961     using IReporterFactoryPtr = std::shared_ptr<IReporterFactory>;
02962
02963     struct IRegistryHub {
02964     virtual ~IRegistryHub();
02965
02966     virtual IReporterRegistry const& getReporterRegistry() const = 0;
02967     virtual ITestCaseRegistry const& getTestCaseRegistry() const = 0;
02968     virtual ITagAliasRegistry const& getTagAliasRegistry() const = 0;
02969     virtual IExceptionTranslatorRegistry const& getExceptionTranslatorRegistry() const = 0;
02970     virtual StartupExceptionRegistry const& getStartupExceptionRegistry() const = 0;
02971
02972     };
02973
02974     struct IMutableRegistryHub {
02975     virtual ~IMutableRegistryHub();
02976     virtual void registerReporter( std::string const& name, IReporterFactoryPtr const& factory ) =
02977     0;
02978     virtual void registerListener( IReporterFactoryPtr const& factory ) = 0;
02979     virtual void registerTest( TestCase const& testInfo ) = 0;
02980     virtual void registerTranslator( const IExceptionTranslator* translator ) = 0;
02981     virtual void registerTagAlias( std::string const& alias, std::string const& tag, \
SourceLineInfo const& lineInfo ) = 0;
02982     virtual void registerStartupException() noexcept = 0;
02983     virtual IMutableEnumValuesRegistry& getMutableEnumValuesRegistry() = 0;
02984     };
02985
02986     IRegistryHub const& getRegistryHub();
02987     IMutableRegistryHub& getMutableRegistryHub();

```

```

02987     void cleanUp();
02988     std::string translateActiveException();
02989
02990 }
02991
02992 // end catch_interfaces_registry_hub.h
02993 #if defined(CATCH_CONFIG_DISABLE)
02994     #define INTERNAL_CATCH_TRANSLATE_EXCEPTION_NO_REG( translatorName, signature) \
02995         static std::string translatorName( signature )
02996 #endif
02997
02998 #include <exception>
02999 #include <string>
03000 #include <vector>
03001
03002 namespace Catch {
03003     using exceptionTranslateFunction = std::string(*)();
03004
03005     struct IExceptionTranslator;
03006     using ExceptionTranslators = std::vector<std::unique_ptr<IExceptionTranslator const>;
03007
03008     struct IExceptionTranslator {
03009         virtual ~IExceptionTranslator();
03010         virtual std::string translate( ExceptionTranslators::const_iterator it,
03011             ExceptionTranslators::const_iterator itEnd ) const = 0;
03012     };
03013
03014     struct IExceptionTranslatorRegistry {
03015         virtual ~IExceptionTranslatorRegistry();
03016
03017         virtual std::string translateActiveException() const = 0;
03018     };
03019
03020     class ExceptionTranslatorRegistrar {
03021     public:
03022         template<typename T>
03023         class ExceptionTranslator : public IExceptionTranslator {
03024         public:
03025             ExceptionTranslator( std::string(*translateFunction)( T& ) )
03026                 : m_translateFunction( translateFunction ) {}
03027
03028             std::string translate( ExceptionTranslators::const_iterator it,
03029                 ExceptionTranslators::const_iterator itEnd ) const override {
03030                 #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
03031                     return "";
03032                 #else
03033                     try {
03034                         if( it == itEnd )
03035                             std::rethrow_exception(std::current_exception());
03036                         else
03037                             return (*it)->translate( it+1, itEnd );
03038                     } catch( T& ex ) {
03039                         return m_translateFunction( ex );
03040                     }
03041                 #endif
03042             }
03043         };
03044
03045     protected:
03046         std::string(*m_translateFunction)( T& );
03047
03048     public:
03049         template<typename T>
03050         ExceptionTranslatorRegistrar( std::string(*translateFunction)( T& ) ) {
03051             getMutableRegistryHub().registerTranslator
03052                 ( new ExceptionTranslator<T>( translateFunction ) );
03053         }
03054     };
03055
03056 #define INTERNAL_CATCH_TRANSLATE_EXCEPTION2( translatorName, signature ) \
03057     static std::string translatorName( signature ); \
03058     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
03059     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
03060     namespace{ Catch::ExceptionTranslatorRegistrar INTERNAL_CATCH_UNIQUE_NAME( \
03061         catch_internal_ExceptionTranslator )( &translatorName ); } \
03062     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
03063     static std::string translatorName( signature )
03064
03065 #define INTERNAL_CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION2( \
03066     INTERNAL_CATCH_UNIQUE_NAME( catch_internal_ExceptionTranslator ), signature )
03067
03068 // end catch_interfaces_exception.h
03069 // start catch_approx.h
03070

```

```

03071 #include <type_traits>
03072
03073 namespace Catch {
03074 namespace Detail {
03075
03076     class Approx {
03077     private:
03078         bool equalityComparisonImpl(double other) const;
03079         // Validates the new margin (margin >= 0)
03080         // out-of-line to avoid including stdexcept in the header
03081         void setMargin(double margin);
03082         // Validates the new epsilon (0 < epsilon < 1)
03083         // out-of-line to avoid including stdexcept in the header
03084         void setEpsilon(double epsilon);
03085
03086     public:
03087         explicit Approx ( double value );
03088
03089         static Approx custom();
03090
03091         Approx operator-() const;
03092
03093     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03094         Approx operator()( T const& value ) {
03095             Approx approx( static_cast<double>(value) );
03096             approx.m_epsilon = m_epsilon;
03097             approx.m_margin = m_margin;
03098             approx.m_scale = m_scale;
03099             return approx;
03100         }
03101
03102     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03103         explicit Approx( T const& value ): Approx(static_cast<double>(value))
03104         {}
03105
03106     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03107         friend bool operator == ( const T& lhs, Approx const& rhs ) {
03108             auto lhs_v = static_cast<double>(lhs);
03109             return rhs.equalityComparisonImpl(lhs_v);
03110         }
03111
03112     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03113         friend bool operator == ( Approx const& lhs, const T& rhs ) {
03114             return operator==( rhs, lhs );
03115         }
03116
03117     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03118         friend bool operator != ( T const& lhs, Approx const& rhs ) {
03119             return !operator==( lhs, rhs );
03120         }
03121
03122     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03123         friend bool operator != ( Approx const& lhs, T const& rhs ) {
03124             return !operator==( rhs, lhs );
03125         }
03126
03127     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03128         friend bool operator <= ( T const& lhs, Approx const& rhs ) {
03129             return static_cast<double>(lhs) < rhs.m_value || lhs == rhs;
03130         }
03131
03132     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03133         friend bool operator <= ( Approx const& lhs, T const& rhs ) {
03134             return lhs.m_value < static_cast<double>(rhs) || lhs == rhs;
03135         }
03136
03137     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03138         friend bool operator >= ( T const& lhs, Approx const& rhs ) {
03139             return static_cast<double>(lhs) > rhs.m_value || lhs == rhs;
03140         }
03141
03142     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03143         friend bool operator >= ( Approx const& lhs, T const& rhs ) {
03144             return lhs.m_value > static_cast<double>(rhs) || lhs == rhs;
03145         }
03146
03147

```

```

        template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03148            Approx& epsilon( T const& newEpsilon ) {
03149                double epsilonAsDouble = static_cast<double>(newEpsilon);
03150                setEpsilon(epsilonAsDouble);
03151                return *this;
03152            }
03153
03154        template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03155            Approx& margin( T const& newMargin ) {
03156                double marginAsDouble = static_cast<double>(newMargin);
03157                setMargin(marginAsDouble);
03158                return *this;
03159            }
03160
03161        template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03162            Approx& scale( T const& newScale ) {
03163                m_scale = static_cast<double>(newScale);
03164                return *this;
03165            }
03166
03167            std::string toString() const;
03168
03169        private:
03170            double m_epsilon;
03171            double m_margin;
03172            double m_scale;
03173            double m_value;
03174    };
03175 } // end namespace Detail
03176
03177 namespace literals {
03178     Detail::Approx operator "" _a(long double val);
03179     Detail::Approx operator "" _a(unsigned long long val);
03180 } // end namespace literals
03181
03182 template<>
03183 struct StringMaker<Catch::Detail::Approx> {
03184     static std::string convert(Catch::Detail::Approx const& value);
03185 };
03186
03187 } // end namespace Catch
03188
03189 // end catch_approx.h
03190 // start catch_string_manip.h
03191
03192 #include <string>
03193 #include <iosfwd>
03194 #include <vector>
03195
03196 namespace Catch {
03197
03198     bool startsWith( std::string const& s, std::string const& prefix );
03199     bool startsWith( std::string const& s, char prefix );
03200     bool endsWith( std::string const& s, std::string const& suffix );
03201     bool endsWith( std::string const& s, char suffix );
03202     bool contains( std::string const& s, std::string const& infix );
03203     void toLowerInPlace( std::string& s );
03204     std::string toLower( std::string const& s );
03205     std::string trim( std::string const& str );
03206     StringRef trim(StringRef ref);
03207
03208     // !!! Be aware, returns refs into original string - make sure original string outlives them
03209     std::vector<StringRef> splitStringRef( StringRef str, char delimiter );
03210     bool replaceInPlace( std::string& str, std::string const& replaceThis, std::string const& withThis );
03211
03212     struct pluralise {
03213         pluralise( std::size_t count, std::string const& label );
03214
03215         friend std::ostream& operator << ( std::ostream& os, pluralise const& pluraliser );
03216
03217         std::size_t m_count;
03218         std::string m_label;
03219     };
03220
03221 }
03222
03223 // end catch_string_manip.h
03224 #ifndef CATCH_CONFIG_DISABLE_MATCHERS
03225 // start catch_capture_matchers.h
03226
03227 // start catch_matchers.h
03228
03229 #include <string>
03230 #include <vector>
03231
03232

```

```

03233 namespace Catch {
03234 namespace Matchers {
03235     namespace Impl {
03236
03237         template<typename ArgT> struct MatchAllOf;
03238         template<typename ArgT> struct MatchAnyOf;
03239         template<typename ArgT> struct MatchNotOf;
03240
03241         class MatcherUntypedBase {
03242         public:
03243             MatcherUntypedBase() = default;
03244             MatcherUntypedBase ( MatcherUntypedBase const& ) = default;
03245             MatcherUntypedBase& operator = ( MatcherUntypedBase const& ) = delete;
03246             std::string toString() const;
03247
03248         protected:
03249             virtual ~MatcherUntypedBase();
03250             virtual std::string describe() const = 0;
03251             mutable std::string m_cachedToString;
03252         };
03253
03254 #ifdef __clang__
03255 #   pragma clang diagnostic push
03256 #   pragma clang diagnostic ignored "-Wnon-virtual-dtor"
03257 #endif
03258
03259         template<typename ObjectT>
03260         struct MatcherMethod {
03261             virtual bool match( ObjectT const& arg ) const = 0;
03262         };
03263
03264 #if defined(__OBJC__)
03265         // Hack to fix Catch GH issue #1661. Could use id for generic Object support.
03266         // use of const for Object pointers is very uncommon and under ARC it causes some kind of
03267         // signature mismatch that breaks compilation
03268         template<>
03269         struct MatcherMethod<NSString*> {
03270             virtual bool match( NSString* arg ) const = 0;
03271         };
03272 #endif
03273
03274 #ifdef __clang__
03275 #   pragma clang diagnostic pop
03276 #endif
03277
03278         template<typename T>
03279         struct MatcherBase : MatcherUntypedBase, MatcherMethod<T> {
03280
03281             MatchAllOf<T> operator && ( MatcherBase const& other ) const;
03282             MatchAnyOf<T> operator || ( MatcherBase const& other ) const;
03283             MatchNotOf<T> operator ! ( ) const;
03284
03285         };
03286
03287         template<typename ArgT>
03288         struct MatchAllOf : MatcherBase<ArgT> {
03289             bool match( ArgT const& arg ) const override {
03290                 for( auto matcher : m_matchers ) {
03291                     if (!matcher->match(arg))
03292                         return false;
03293                 }
03294                 return true;
03295             }
03296             std::string describe() const override {
03297                 std::string description;
03298                 description.reserve( 4 + m_matchers.size()*32 );
03299                 description += "( ";
03300                 bool first = true;
03301                 for( auto matcher : m_matchers ) {
03302                     if( first )
03303                         first = false;
03304                     else
03305                         description += " and ";
03306                     description += matcher->toString();
03307                 }
03308                 description += " )";
03309                 return description;
03310             }
03311             MatchAllOf<ArgT> operator && ( MatcherBase<ArgT> const& other ) {
03312                 auto copy(*this);
03313                 copy.m_matchers.push_back( &other );
03314                 return copy;
03315             }
03316             std::vector<MatcherBase<ArgT> const*> m_matchers;
03317         };
03318         template<typename ArgT>

```

```

03319     struct MatchAnyOf : MatcherBase<ArgT> {
03320
03321         bool match( ArgT const& arg ) const override {
03322             for( auto matcher : m_matchers ) {
03323                 if (matcher->match(arg))
03324                     return true;
03325             }
03326             return false;
03327         }
03328         std::string describe() const override {
03329             std::string description;
03330             description.reserve( 4 + m_matchers.size()*32 );
03331             description += "( ";
03332             bool first = true;
03333             for( auto matcher : m_matchers ) {
03334                 if( first )
03335                     first = false;
03336                 else
03337                     description += " or ";
03338                 description += matcher->toString();
03339             }
03340             description += " )";
03341             return description;
03342         }
03343
03344         MatchAnyOf<ArgT> operator || ( MatcherBase<ArgT> const& other ) {
03345             auto copy(*this);
03346             copy.m_matchers.push_back( &other );
03347             return copy;
03348         }
03349
03350         std::vector<MatcherBase<ArgT> const*> m_matchers;
03351     };
03352
03353     template<typename ArgT>
03354     struct MatchNotOf : MatcherBase<ArgT> {
03355
03356         MatchNotOf( MatcherBase<ArgT> const& underlyingMatcher ) : m_underlyingMatcher(
underlyingMatcher ) {}
03357
03358         bool match( ArgT const& arg ) const override {
03359             return !m_underlyingMatcher.match( arg );
03360         }
03361
03362         std::string describe() const override {
03363             return "not " + m_underlyingMatcher.toString();
03364         }
03365         MatcherBase<ArgT> const& m_underlyingMatcher;
03366     };
03367
03368     template<typename T>
03369     MatchAllOf<T> MatcherBase<T>::operator && ( MatcherBase const& other ) const {
03370         return MatchAllOf<T>() && *this && other;
03371     }
03372     template<typename T>
03373     MatchAnyOf<T> MatcherBase<T>::operator || ( MatcherBase const& other ) const {
03374         return MatchAnyOf<T>() || *this || other;
03375     }
03376     template<typename T>
03377     MatchNotOf<T> MatcherBase<T>::operator ! () const {
03378         return MatchNotOf<T>( *this );
03379     }
03380
03381 } // namespace Impl
03382
03383 } // namespace Matchers
03384
03385 using namespace Matchers;
03386 using Matchers::Impl::MatcherBase;
03387
03388 } // namespace Catch
03389
03390 // end catch_matchers.h
03391 // start catch_matchers_exception.hpp
03392
03393 namespace Catch {
03394     namespace Matchers {
03395         namespace Exception {
03396
03397             class ExceptionMessageMatcher : public MatcherBase<std::exception> {
03398                 std::string m_message;
03399             public:
03400
03401                 ExceptionMessageMatcher( std::string const& message ) :
03402                     m_message( message )
03403             {}
03404

```

```

03405     bool match(std::exception const& ex) const override;
03406
03407     std::string describe() const override;
03408 };
03409
03410 } // namespace Exception
03411
03412 Exception::ExceptionMessageMatcher Message(std::string const& message);
03413
03414 } // namespace Matchers
03415 } // namespace Catch
03416
03417 // end catch_matchers_exception.hpp
03418 // start catch_matchers_floating.h
03419
03420 namespace Catch {
03421 namespace Matchers {
03422
03423     namespace Floating {
03424
03425         enum class FloatingPointKind : uint8_t;
03426
03427         struct WithinAbsMatcher : MatcherBase<double> {
03428             WithinAbsMatcher(double target, double margin);
03429             bool match(double const& matchee) const override;
03430             std::string describe() const override;
03431         private:
03432             double m_target;
03433             double m_margin;
03434         };
03435
03436         struct WithinUlpMatcher : MatcherBase<double> {
03437             WithinUlpMatcher(double target, uint64_t ulps, FloatingPointKind baseType);
03438             bool match(double const& matchee) const override;
03439             std::string describe() const override;
03440         private:
03441             double m_target;
03442             uint64_t m_ulps;
03443             FloatingPointKind m_type;
03444         };
03445
03446         // Given IEEE-754 format for floats and doubles, we can assume
03447         // that float -> double promotion is lossless. Given this, we can
03448         // assume that if we do the standard relative comparison of
03449         // |lhs - rhs| <= epsilon * max(fabs(lhs), fabs(rhs)), then we get
03450         // the same result if we do this for floats, as if we do this for
03451         // doubles that were promoted from floats.
03452         struct WithinRelMatcher : MatcherBase<double> {
03453             WithinRelMatcher(double target, double epsilon);
03454             bool match(double const& matchee) const override;
03455             std::string describe() const override;
03456         private:
03457             double m_target;
03458             double m_epsilon;
03459         };
03460
03461     } // namespace Floating
03462
03463     // The following functions create the actual matcher objects.
03464     // This allows the types to be inferred
03465     Floating::WithinUlpMatcher WithinULP(double target, uint64_t maxUlpDiff);
03466     Floating::WithinUlpMatcher WithinULP(float target, uint64_t maxUlpDiff);
03467     Floating::WithinAbsMatcher WithinAbs(double target, double margin);
03468     Floating::WithinRelMatcher WithinRel(double target, double eps);
03469     // defaults epsilon to 100*numeric_limits<double>::epsilon()
03470     Floating::WithinRelMatcher WithinRel(double target);
03471     Floating::WithinRelMatcher WithinRel(float target, float eps);
03472     // defaults epsilon to 100*numeric_limits<float>::epsilon()
03473     Floating::WithinRelMatcher WithinRel(float target);
03474
03475 } // namespace Matchers
03476 } // namespace Catch
03477
03478 // end catch_matchers_floating.h
03479 // start catch_matchers_generic.hpp
03480
03481 #include <functional>
03482 #include <string>
03483
03484 namespace Catch {
03485 namespace Matchers {
03486 namespace Generic {
03487
03488 namespace Detail {
03489     std::string finalizeDescription(const std::string& desc);
03490 }
03491

```



```

03492 template <typename T>
03493 class PredicateMatcher : public MatcherBase<T> {
03494     std::function<bool(T const&)> m_predicate;
03495     std::string m_description;
03496 public:
03497
03498     PredicateMatcher(std::function<bool(T const&)> const& elem, std::string const& descr)
03499         :m_predicate(std::move(elem)),
03500         m_description(Detail::finalizeDescription(descr))
03501     {}
03502
03503     bool match( T const& item ) const override {
03504         return m_predicate(item);
03505     }
03506
03507     std::string describe() const override {
03508         return m_description;
03509     }
03510 };
03511
03512 } // namespace Generic
03513
03514 // The following functions create the actual matcher objects.
03515 // The user has to explicitly specify type to the function, because
03516 // inferring std::function<bool(T const&)> is hard (but possible) and
03517 // requires a lot of TMP.
03518 template<typename T>
03519 Generic::PredicateMatcher<T> Predicate(std::function<bool(T const&)> const& predicate, std::string
const& description = "") {
03520     return Generic::PredicateMatcher<T>(predicate, description);
03521 }
03522
03523 } // namespace Matchers
03524 } // namespace Catch
03525
03526 // end catch_matchers_generic.hpp
03527 // start catch_matchers_string.h
03528
03529 #include <string>
03530
03531 namespace Catch {
03532 namespace Matchers {
03533
03534     namespace StdString {
03535
03536         struct CasedString
03537         {
03538             CasedString( std::string const& str, CaseSensitive::Choice caseSensitivity );
03539             std::string adjustString( std::string const& str ) const;
03540             std::string caseSensitivitySuffix() const;
03541
03542             CaseSensitive::Choice m_caseSensitivity;
03543             std::string m_str;
03544         };
03545
03546         struct StringMatcherBase : MatcherBase<std::string> {
03547             StringMatcherBase( std::string const& operation, CasedString const& comparator );
03548             std::string describe() const override;
03549
03550             CasedString m_comparator;
03551             std::string m_operation;
03552         };
03553
03554         struct EqualsMatcher : StringMatcherBase {
03555             EqualsMatcher( CasedString const& comparator );
03556             bool match( std::string const& source ) const override;
03557         };
03558         struct ContainsMatcher : StringMatcherBase {
03559             ContainsMatcher( CasedString const& comparator );
03560             bool match( std::string const& source ) const override;
03561         };
03562         struct StartsWithMatcher : StringMatcherBase {
03563             StartsWithMatcher( CasedString const& comparator );
03564             bool match( std::string const& source ) const override;
03565         };
03566         struct EndsWithMatcher : StringMatcherBase {
03567             EndsWithMatcher( CasedString const& comparator );
03568             bool match( std::string const& source ) const override;
03569         };
03570
03571         struct RegexMatcher : MatcherBase<std::string> {
03572             RegexMatcher( std::string regex, CaseSensitive::Choice caseSensitivity );
03573             bool match( std::string const& matchee ) const override;
03574             std::string describe() const override;
03575
03576         private:
03577             std::string m_regex;

```

```

03578         CaseSensitive::Choice m_caseSensitivity;
03579     };
03580
03581     } // namespace StdString
03582
03583     // The following functions create the actual matcher objects.
03584     // This allows the types to be inferred
03585
03586     StdString::EqualsMatcher Equals( std::string const& str, CaseSensitive::Choice caseSensitivity =
CaseSensitive::Yes );
03587     StdString::ContainsMatcher Contains( std::string const& str, CaseSensitive::Choice caseSensitivity
= CaseSensitive::Yes );
03588     StdString::EndsWithMatcher EndsWith( std::string const& str, CaseSensitive::Choice caseSensitivity
= CaseSensitive::Yes );
03589     StdString::StartsWithMatcher StartsWith( std::string const& str, CaseSensitive::Choice
caseSensitivity = CaseSensitive::Yes );
03590     StdString::RegexMatcher Matches( std::string const& regex, CaseSensitive::Choice caseSensitivity =
CaseSensitive::Yes );
03591
03592 } // namespace Matchers
03593 } // namespace Catch
03594
03595 // end catch_matchers_string.h
03596 // start catch_matchers_vector.h
03597
03598 #include <algorithm>
03599
03600 namespace Catch {
03601 namespace Matchers {
03602
03603     namespace Vector {
03604         template<typename T, typename Alloc>
03605         struct ContainsElementMatcher : MatcherBase<std::vector<T, Alloc> > {
03606
03607             ContainsElementMatcher(T const& comparator) : m_comparator( comparator ) {}
03608
03609             bool match(std::vector<T, Alloc> const& v) const override {
03610                 for (auto const& el : v) {
03611                     if (el == m_comparator) {
03612                         return true;
03613                     }
03614                 }
03615                 return false;
03616             }
03617
03618             std::string describe() const override {
03619                 return "Contains: " + ::Catch::Detail::stringify( m_comparator );
03620             }
03621
03622             T const& m_comparator;
03623         };
03624
03625         template<typename T, typename AllocComp, typename AllocMatch>
03626         struct ContainsMatcher : MatcherBase<std::vector<T, AllocMatch> > {
03627
03628             ContainsMatcher(std::vector<T, AllocComp> const& comparator) : m_comparator( comparator )
{}
03629
03630             bool match(std::vector<T, AllocMatch> const& v) const override {
03631                 // !TBD: see note in EqualsMatcher
03632                 if (m_comparator.size() > v.size())
03633                     return false;
03634                 for (auto const& comparator : m_comparator) {
03635                     auto present = false;
03636                     for (const auto& el : v) {
03637                         if (el == comparator) {
03638                             present = true;
03639                             break;
03640                         }
03641                     }
03642                     if (!present) {
03643                         return false;
03644                     }
03645                 }
03646                 return true;
03647             }
03648
03649             std::string describe() const override {
03650                 return "Contains: " + ::Catch::Detail::stringify( m_comparator );
03651             }
03652
03653             std::vector<T, AllocComp> const& m_comparator;
03654         };
03655
03656         template<typename T, typename AllocComp, typename AllocMatch>
03657         struct EqualsMatcher : MatcherBase<std::vector<T, AllocMatch> > {
03658
03659             EqualsMatcher(std::vector<T, AllocComp> const& comparator) : m_comparator( comparator ) {}

```

```

03659
03660     bool match(std::vector<T, AllocMatch> const &v) const override {
03661         // !TBD: This currently works if all elements can be compared using !=
03662         // - a more general approach would be via a compare template that defaults
03663         // to using !=. but could be specialised for, e.g. std::vector<T, Alloc> etc
03664         // - then just call that directly
03665         if (m_comparator.size() != v.size())
03666             return false;
03667         for (std::size_t i = 0; i < v.size(); ++i)
03668             if (m_comparator[i] != v[i])
03669                 return false;
03670         return true;
03671     }
03672     std::string describe() const override {
03673         return "Equals: " + ::Catch::Detail::stringify( m_comparator );
03674     }
03675     std::vector<T, AllocComp> const& m_comparator;
03676 };
03677
03678 template<typename T, typename AllocComp, typename AllocMatch>
03679 struct ApproxMatcher : MatcherBase<std::vector<T, AllocMatch> > {
03680     ApproxMatcher(std::vector<T, AllocComp> const& comparator) : m_comparator( comparator ) {}
03681
03682     bool match(std::vector<T, AllocMatch> const &v) const override {
03683         if (m_comparator.size() != v.size())
03684             return false;
03685         for (std::size_t i = 0; i < v.size(); ++i)
03686             if (m_comparator[i] != approx(v[i]))
03687                 return false;
03688         return true;
03689     }
03690     std::string describe() const override {
03691         return "is approx: " + ::Catch::Detail::stringify( m_comparator );
03692     }
03693 }
03694
03695 template <typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03696     ApproxMatcher& epsilon( T const& newEpsilon ) {
03697     approx.epsilon(newEpsilon);
03698     return *this;
03699 }
03700
03701 template <typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03702     ApproxMatcher& margin( T const& newMargin ) {
03703     approx.margin(newMargin);
03704     return *this;
03705 }
03706
03707 template <typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03708     ApproxMatcher& scale( T const& newScale ) {
03709     approx.scale(newScale);
03710     return *this;
03711 }
03712
03713 std::vector<T, AllocComp> const& m_comparator;
03714 mutable Catch::Detail::Approx approx = Catch::Detail::Approx::custom();
03715 };
03716
03717 template<typename T, typename AllocComp, typename AllocMatch>
03718 struct UnorderedEqualsMatcher : MatcherBase<std::vector<T, AllocMatch> > {
03719     UnorderedEqualsMatcher(std::vector<T, AllocComp> const& target) : m_target(target) {}
03720     bool match(std::vector<T, AllocMatch> const& vec) const override {
03721         if (m_target.size() != vec.size()) {
03722             return false;
03723         }
03724         return std::is_permutation(m_target.begin(), m_target.end(), vec.begin());
03725     }
03726     std::string describe() const override {
03727         return "UnorderedEquals: " + ::Catch::Detail::stringify(m_target);
03728     }
03729     private:
03730         std::vector<T, AllocComp> const& m_target;
03731 };
03732
03733 } // namespace Vector
03734
03735 // The following functions create the actual matcher objects.
03736 // This allows the types to be inferred
03737
03738 template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
03739 Vector::ContainsMatcher<T, AllocComp, AllocMatch> Contains( std::vector<T, AllocComp> const&
03740     comparator ) {
03741     return Vector::ContainsMatcher<T, AllocComp, AllocMatch>( comparator );
03742 }
03743
03744 template<typename T, typename Alloc = std::allocator<T>

```

```

03742     Vector::ContainsElementMatcher<T, Alloc> VectorContains( T const& comparator ) {
03743         return Vector::ContainsElementMatcher<T, Alloc>( comparator );
03744     }
03745
03746     template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
03747     Vector::EqualsMatcher<T, AllocComp, AllocMatch> Equals( std::vector<T, AllocComp> const&
comparator ) {
03748         return Vector::EqualsMatcher<T, AllocComp, AllocMatch>( comparator );
03749     }
03750
03751     template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
03752     Vector::ApproxMatcher<T, AllocComp, AllocMatch> Approx( std::vector<T, AllocComp> const&
comparator ) {
03753         return Vector::ApproxMatcher<T, AllocComp, AllocMatch>( comparator );
03754     }
03755
03756     template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
03757     Vector::UnorderedEqualsMatcher<T, AllocComp, AllocMatch> UnorderedEquals( std::vector<T, AllocComp>
const& target) {
03758         return Vector::UnorderedEqualsMatcher<T, AllocComp, AllocMatch>( target );
03759     }
03760
03761 } // namespace Matchers
03762 } // namespace Catch
03763
03764 // end catch_matchers_vector.h
03765 namespace Catch {
03766
03767     template<typename ArgT, typename MatcherT>
03768     class MatchExpr : public ITransientExpression {
03769     public:
03770         ArgT const& m_arg;
03771         MatcherT m_matcher;
03772         StringRef m_matcherString;
03773
03774         MatchExpr( ArgT const& arg, MatcherT const& matcher, StringRef const& matcherString )
03775         : ITransientExpression( true, matcher.match( arg ) ),
03776           m_arg( arg ),
03777           m_matcher( matcher ),
03778           m_matcherString( matcherString )
03779         {}
03780
03781         void streamReconstructedExpression( std::ostream &os ) const override {
03782             auto matcherAsString = m_matcher.toString();
03783             os << Catch::Detail::stringify( m_arg ) << ' ';
03784             if( matcherAsString == Detail::unprintableString )
03785                 os << m_matcherString;
03786             else
03787                 os << matcherAsString;
03788         }
03789
03790         using StringMatcher = Matchers::Impl::MatcherBase<std::string>;
03791
03792         void handleExceptionMatchExpr( AssertionHandler& handler, StringMatcher const& matcher, StringRef
const& matcherString );
03793
03794         template<typename ArgT, typename MatcherT>
03795         auto makeMatchExpr( ArgT const& arg, MatcherT const& matcher, StringRef const& matcherString ) ->
MatchExpr<ArgT, MatcherT> {
03796             return MatchExpr<ArgT, MatcherT>( arg, matcher, matcherString );
03797         }
03798     };
03799 } // namespace Catch
03800
03801 #define INTERNAL_CHECK_THAT( macroName, matcher, resultDisposition, arg ) \
03802     do { \
03803         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
CATCH_INTERNAL_STRINGIFY(arg) ", " CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
03804         INTERNAL_CATCH_TRY { \
03805             catchAssertionHandler.handleExpr( Catch::makeMatchExpr( arg, matcher, #matcher##_catch_sr \
)); \
03806         } INTERNAL_CATCH_CATCH( catchAssertionHandler ) \
03807         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
03808     } while( false )
03809
03810 #define INTERNAL_CATCH_THROWS_MATCHES( macroName, exceptionType, resultDisposition, matcher, ... ) \
03811     do { \
03812         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
CATCH_INTERNAL_STRINGIFY(__VA_ARGS__) ", " CATCH_INTERNAL_STRINGIFY(exceptionType) ", " \
CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
03813         if( catchAssertionHandler.allowThrows() ) \
03814             try { \
03815                 static_cast<void>(__VA_ARGS__ ); \
03816                 catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
03817             } \
03818             catch( exceptionType const& ex ) { \
03819                 catchAssertionHandler.handleExpr( Catch::makeMatchExpr( ex, matcher,

```

```

03822     #matcher##_catch_sr ) ); \
03823     } \
03824     catch( ... ) { \
03825         catchAssertionHandler.handleUnexpectedInflightException(); \
03826     } \
03827     else \
03828         catchAssertionHandler.handleThrowingCallSkipped(); \
03829     INTERNAL_CATCH_REACT( catchAssertionHandler ) \
03830     } while( false )
03831 // end catch_capture_matchers.h
03832 #endif
03833 // start catch_generators.hpp
03834
03835 // start catch_interfaces_generatortracker.h
03836
03837 #include <memory>
03838 namespace Catch {
03839     namespace Generators {
03840         class GeneratorUntypedBase {
03841         public:
03842             GeneratorUntypedBase() = default;
03843             virtual ~GeneratorUntypedBase();
03844             // Attempts to move the generator to the next element
03845             //
03846             // Returns true iff the move succeeded (and a valid element
03847             // can be retrieved).
03848             virtual bool next() = 0;
03849         };
03850         using GeneratorBasePtr = std::unique_ptr<GeneratorUntypedBase>;
03851     } // namespace Generators
03852
03853     struct IGeneratorTracker {
03854     public:
03855         virtual ~IGeneratorTracker();
03856         virtual auto hasGenerator() const -> bool = 0;
03857         virtual auto getGenerator() const -> Generators::GeneratorBasePtr const& = 0;
03858         virtual void setGenerator( Generators::GeneratorBasePtr&& generator ) = 0;
03859     };
03860 } // namespace Catch
03861
03862 // end catch_interfaces_generatortracker.h
03863 // start catch_enforce.h
03864
03865 #include <exception>
03866 namespace Catch {
03867     template <typename Ex>
03868     [[noreturn]]
03869     void throw_exception(Ex const& e) {
03870         throw e;
03871     }
03872 #else // ^^ Exceptions are enabled // Exceptions are disabled vv
03873     [[noreturn]]
03874     void throw_exception(std::exception const& e);
03875 #endif
03876
03877     [[noreturn]]
03878     void throw_logic_error(std::string const& msg);
03879     [[noreturn]]
03880     void throw_domain_error(std::string const& msg);
03881     [[noreturn]]
03882     void throw_runtime_error(std::string const& msg);
03883 } // namespace Catch;
03884
03885 #define CATCH_MAKE_MSG(...) \
03886     (Catch::ReusableStringStream() << __VA_ARGS__).str()
03887
03888 #define CATCH_INTERNAL_ERROR(...) \
03889     Catch::throw_logic_error(CATCH_MAKE_MSG( CATCH_INTERNAL_LINEINFO << " Internal Catch2 error: " << __VA_ARGS__))
03890
03891 #define CATCH_ERROR(...) \
03892     Catch::throw_domain_error(CATCH_MAKE_MSG( __VA_ARGS__ ))
03893
03894 #define CATCH_RUNTIME_ERROR(...) \
03895     Catch::throw_runtime_error(CATCH_MAKE_MSG( __VA_ARGS__ ))
03896
03897 #define CATCH_ENFORCE( condition, ... ) \
03898     do{ if( !(condition) ) CATCH_ERROR( __VA_ARGS__ ); } while(false)
03899
03900

```

```

03907 // end catch_enforce.h
03908 #include <memory>
03909 #include <vector>
03910 #include <cassert>
03911
03912 #include <utility>
03913 #include <exception>
03914
03915 namespace Catch {
03916
03917 class GeneratorException : public std::exception {
03918     const char* const m_msg = "";
03919
03920 public:
03921     GeneratorException(const char* msg):
03922         m_msg(msg)
03923     {}
03924
03925     const char* what() const noexcept override final;
03926 };
03927
03928 namespace Generators {
03929
03930     // !TBD move this into its own location?
03931     namespace pf{
03932         template<typename T, typename... Args>
03933         std::unique_ptr<T> make_unique( Args&&... args ) {
03934             return std::unique_ptr<T>(new T(std::forward<Args>(args)...));
03935         }
03936     }
03937
03938     template<typename T>
03939     struct IGenerator : GeneratorUntypedBase {
03940         virtual ~IGenerator() = default;
03941
03942         // Returns the current element of the generator
03943         //
03944         // \Precondition The generator is either freshly constructed,
03945         // or the last call to `next()` returned true
03946         virtual T const& get() const = 0;
03947         using type = T;
03948     };
03949
03950     template<typename T>
03951     class SingleValueGenerator final : public IGenerator<T> {
03952         T m_value;
03953     public:
03954         SingleValueGenerator(T&& value) : m_value(std::move(value)) {}
03955
03956         T const& get() const override {
03957             return m_value;
03958         }
03959         bool next() override {
03960             return false;
03961         }
03962     };
03963
03964     template<typename T>
03965     class FixedValuesGenerator final : public IGenerator<T> {
03966         static_assert(!std::is_same<T, bool>::value,
03967             "FixedValuesGenerator does not support bools because of std::vector<bool>"
03968             "specialization, use SingleValue Generator instead.");
03969         std::vector<T> m_values;
03970         size_t m_idx = 0;
03971     public:
03972         FixedValuesGenerator( std::initializer_list<T> values ) : m_values( values ) {}
03973
03974         T const& get() const override {
03975             return m_values[m_idx];
03976         }
03977         bool next() override {
03978             ++m_idx;
03979             return m_idx < m_values.size();
03980         }
03981     };
03982
03983     template <typename T>
03984     class GeneratorWrapper final {
03985         std::unique_ptr<IGenerator<T>> m_generator;
03986     public:
03987         GeneratorWrapper( std::unique_ptr<IGenerator<T>> generator ):
03988             m_generator( std::move( generator ) )
03989         {}
03990         T const& get() const {
03991             return m_generator->get();
03992         }
03993         bool next() {

```

```

03994         return m_generator->next();
03995     }
03996 };
03997
03998 template <typename T>
03999 GeneratorWrapper<T> value(T&& value) {
04000     return GeneratorWrapper<T>(pf::make_unique<SingleValueGenerator<T>>(std::forward<T>(value)));
04001 }
04002 template <typename T>
04003 GeneratorWrapper<T> values(std::initializer_list<T> values) {
04004     return GeneratorWrapper<T>(pf::make_unique<FixedValuesGenerator<T>>(values));
04005 }
04006
04007 template<typename T>
04008 class Generators : public IGenerator<T> {
04009     std::vector<GeneratorWrapper<T>> m_generators;
04010     size_t m_current = 0;
04011
04012     void populate(GeneratorWrapper<T>&& generator) {
04013         m_generators.emplace_back(std::move(generator));
04014     }
04015     void populate(T&& val) {
04016         m_generators.emplace_back(value(std::forward<T>(val)));
04017     }
04018     template<typename U>
04019     void populate(U&& val) {
04020         populate(T(std::forward<U>(val)));
04021     }
04022     template<typename U, typename... Gs>
04023     void populate(U&& valueOrGenerator, Gs &&... moreGenerators) {
04024         populate(std::forward<U>(valueOrGenerator));
04025         populate(std::forward<Gs>(moreGenerators)...);
04026     }
04027
04028 public:
04029     template <typename... Gs>
04030     Generators(Gs &&... moreGenerators) {
04031         m_generators.reserve(sizeof...(Gs));
04032         populate(std::forward<Gs>(moreGenerators)...);
04033     }
04034
04035     T const& get() const override {
04036         return m_generators[m_current].get();
04037     }
04038
04039     bool next() override {
04040         if (m_current >= m_generators.size()) {
04041             return false;
04042         }
04043         const bool current_status = m_generators[m_current].next();
04044         if (!current_status) {
04045             ++m_current;
04046         }
04047         return m_current < m_generators.size();
04048     }
04049 };
04050
04051 template<typename... Ts>
04052 GeneratorWrapper<std::tuple<Ts...>> table( std::initializer_list<std::tuple<typename
std::decay<Ts>::type...> tuples ) {
04053     return values<std::tuple<Ts...>>( tuples );
04054 }
04055
04056 // Tag type to signal that a generator sequence should convert arguments to a specific type
04057 template <typename T>
04058 struct as {};
04059
04060 template<typename T, typename... Gs>
04061 auto makeGenerators( GeneratorWrapper<T>&& generator, Gs &&... moreGenerators ) -> Generators<T> {
04062     return Generators<T>(std::move(generator), std::forward<Gs>(moreGenerators)...);
04063 }
04064 template<typename T>
04065 auto makeGenerators( GeneratorWrapper<T>&& generator ) -> Generators<T> {
04066     return Generators<T>(std::move(generator));
04067 }
04068 template<typename T, typename... Gs>
04069 auto makeGenerators( T&& val, Gs &&... moreGenerators ) -> Generators<T> {
04070     return makeGenerators( value( std::forward<T>( val ) ), std::forward<Gs>( moreGenerators )...
);
04071 }
04072 template<typename T, typename U, typename... Gs>
04073 auto makeGenerators( as<T>, U&& val, Gs &&... moreGenerators ) -> Generators<T> {
04074     return makeGenerators( value( T( std::forward<U>( val ) ) ), std::forward<Gs>( moreGenerators
)... );
04075 }
04076
04077 auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->

```

```

    IGeneratorTracker&;

04078
04079     template<typename L>
04080     // Note: The type after -> is weird, because VS2015 cannot parse
04081     //       the expression used in the typedef inside, when it is in
04082     //       return type. Yeah.
04083     auto generate( StringRef generatorName, SourceLineInfo const& lineInfo, L const&
generatorExpression ) -> decltype(std::declval<decltype(generatorExpression())>().get()) {
04084         using UnderlyingType = typename decltype(generatorExpression())::type;
04085
04086         IGeneratorTracker& tracker = acquireGeneratorTracker( generatorName, lineInfo );
04087         if (!tracker.hasGenerator()) {
04088             tracker.setGenerator(pf::make_unique<Generators<UnderlyingType>>(generatorExpression()));
04089         }
04090
04091         auto const& generator = static_cast<IGenerator<UnderlyingType>> const&(&
*tracker.getGenerator() );
04092         return generator.get();
04093     }
04094
04095 } // namespace Generators
04096 } // namespace Catch
04097
04098 #define GENERATE( ... ) \
04099     Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
04100                                 CATCH_INTERNAL_LINEINFO, \
04101                                 [ ]{ using namespace Catch::Generators; return makeGenerators(
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
04102 #define GENERATE_COPY( ... ) \
04103     Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
04104                                 CATCH_INTERNAL_LINEINFO, \
04105                                 [=]{ using namespace Catch::Generators; return makeGenerators(
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
04106 #define GENERATE_REF( ... ) \
04107     Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
04108                                 CATCH_INTERNAL_LINEINFO, \
04109                                 [&]{ using namespace Catch::Generators; return makeGenerators(
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
04110
04111 // end catch_generators.hpp
04112 // start catch_generators_generic.hpp
04113
04114 namespace Catch {
04115     namespace Generators {
04116
04117         template <typename T>
04118         class TakeGenerator : public IGenerator<T> {
04119             GeneratorWrapper<T> m_generator;
04120             size_t m_returned = 0;
04121             size_t m_target;
04122         public:
04123             TakeGenerator(size_t target, GeneratorWrapper<T>&& generator):
04124                 m_generator(std::move(generator)),
04125                 m_target(target)
04126             {
04127                 assert(target != 0 && "Empty generators are not allowed");
04128             }
04129             T const& get() const override {
04130                 return m_generator.get();
04131             }
04132             bool next() override {
04133                 ++m_returned;
04134                 if (m_returned >= m_target) {
04135                     return false;
04136                 }
04137
04138                 const auto success = m_generator.next();
04139                 // If the underlying generator does not contain enough values
04140                 // then we cut short as well
04141                 if (!success) {
04142                     m_returned = m_target;
04143                 }
04144                 return success;
04145             }
04146         };
04147
04148         template <typename T>
04149         GeneratorWrapper<T> take(size_t target, GeneratorWrapper<T>&& generator) {
04150             return GeneratorWrapper<T>(pf::make_unique<TakeGenerator<T>>(target, std::move(generator)));
04151         }
04152
04153         template <typename T, typename Predicate>
04154         class FilterGenerator : public IGenerator<T> {
04155             GeneratorWrapper<T> m_generator;
04156             Predicate m_predicate;
04157         public:
04158             template <typename P = Predicate>

```



```

04159     FilterGenerator(P&& pred, GeneratorWrapper<T>&& generator):
04160         m_generator(std::move(generator)),
04161         m_predicate(std::forward<P>(pred))
04162     {
04163         if (!m_predicate(m_generator.get())) {
04164             // It might happen that there are no values that pass the
04165             // filter. In that case we throw an exception.
04166             auto has_initial_value = next();
04167             if (!has_initial_value) {
04168                 Catch::throw_exception(GeneratorException("No valid value found in filtered
generator"));
04169             }
04170         }
04171     }
04172
04173     T const& get() const override {
04174         return m_generator.get();
04175     }
04176
04177     bool next() override {
04178         bool success = m_generator.next();
04179         if (!success) {
04180             return false;
04181         }
04182         while (!m_predicate(m_generator.get()) && (success = m_generator.next()) == true);
04183         return success;
04184     }
04185 };
04186
04187 template <typename T, typename Predicate>
04188 GeneratorWrapper<T> filter(Predicate&& pred, GeneratorWrapper<T>&& generator) {
04189     return
GeneratorWrapper<T>(std::unique_ptr<IGenerator<T>>(pf::make_unique<FilterGenerator<T, Predicate>>(std::forward<Predicate>
std::move(generator))));
04190 }
04191
04192 template <typename T>
04193 class RepeatGenerator : public IGenerator<T> {
04194     static_assert(!std::is_same<T, bool>::value,
04195         "RepeatGenerator currently does not support bools"
04196         "because of std::vector<bool> specialization");
04197     GeneratorWrapper<T> m_generator;
04198     mutable std::vector<T> m_returned;
04199     size_t m_target_repeats;
04200     size_t m_current_repeat = 0;
04201     size_t m_repeat_index = 0;
04202 public:
04203     RepeatGenerator(size_t repeats, GeneratorWrapper<T>&& generator):
04204         m_generator(std::move(generator)),
04205         m_target_repeats(repeats)
04206     {
04207         assert(m_target_repeats > 0 && "Repeat generator must repeat at least once");
04208     }
04209
04210     T const& get() const override {
04211         if (m_current_repeat == 0) {
04212             m_returned.push_back(m_generator.get());
04213             return m_returned.back();
04214         }
04215         return m_returned[m_repeat_index];
04216     }
04217
04218     bool next() override {
04219         // There are 2 basic cases:
04220         // 1) We are still reading the generator
04221         // 2) We are reading our own cache
04222
04223         // In the first case, we need to poke the underlying generator.
04224         // If it happily moves, we are left in that state, otherwise it is time to start reading
from our cache
04225         if (m_current_repeat == 0) {
04226             const auto success = m_generator.next();
04227             if (!success) {
04228                 ++m_current_repeat;
04229             }
04230             return m_current_repeat < m_target_repeats;
04231         }
04232
04233         // In the second case, we need to move indices forward and check that we haven't run up
against the end
04234         ++m_repeat_index;
04235         if (m_repeat_index == m_returned.size()) {
04236             m_repeat_index = 0;
04237             ++m_current_repeat;
04238         }
04239         return m_current_repeat < m_target_repeats;
04240     }

```

```

04241     };
04242
04243     template <typename T>
04244     GeneratorWrapper<T> repeat(size_t repeats, GeneratorWrapper<T>&& generator) {
04245         return GeneratorWrapper<T>(pf::make_unique<RepeatGenerator<T>>(repeats,
std::move(generator)));
04246     }
04247
04248     template <typename T, typename U, typename Func>
04249     class MapGenerator : public IGenerator<T> {
04250         // TBD: provide static assert for mapping function, for friendly error message
04251         GeneratorWrapper<U> m_generator;
04252         Func m_function;
04253         // To avoid returning dangling reference, we have to save the values
04254         T m_cache;
04255     public:
04256         template <typename F2 = Func>
04257         MapGenerator(F2&& function, GeneratorWrapper<U>&& generator) :
04258             m_generator(std::move(generator)),
04259             m_function(std::forward<F2>(function)),
04260             m_cache(m_function(m_generator.get()))
04261         {}
04262
04263         T const& get() const override {
04264             return m_cache;
04265         }
04266         bool next() override {
04267             const auto success = m_generator.next();
04268             if (success) {
04269                 m_cache = m_function(m_generator.get());
04270             }
04271             return success;
04272         }
04273     };
04274
04275     template <typename Func, typename U, typename T = FunctionReturnType<Func, U>
04276     GeneratorWrapper<T> map(Func&& function, GeneratorWrapper<U>&& generator) {
04277         return GeneratorWrapper<T>(
04278             pf::make_unique<MapGenerator<T, U, Func>>(std::forward<Func>(function),
std::move(generator))
04279         );
04280     }
04281
04282     template <typename T, typename U, typename Func>
04283     GeneratorWrapper<T> map(Func&& function, GeneratorWrapper<U>&& generator) {
04284         return GeneratorWrapper<T>(
04285             pf::make_unique<MapGenerator<T, U, Func>>(std::forward<Func>(function),
std::move(generator))
04286         );
04287     }
04288
04289     template <typename T>
04290     class ChunkGenerator final : public IGenerator<std::vector<T>> {
04291     std::vector<T> m_chunk;
04292     size_t m_chunk_size;
04293     GeneratorWrapper<T> m_generator;
04294     bool m_used_up = false;
04295     public:
04296     ChunkGenerator(size_t size, GeneratorWrapper<T> generator) :
04297         m_chunk_size(size), m_generator(std::move(generator))
04298     {
04299         m_chunk.reserve(m_chunk_size);
04300         if (m_chunk_size != 0) {
04301             m_chunk.push_back(m_generator.get());
04302             for (size_t i = 1; i < m_chunk_size; ++i) {
04303                 if (!m_generator.next()) {
04304                     Catch::throw_exception(GeneratorException("Not enough values to initialize the
first chunk"));
04305                 }
04306                 m_chunk.push_back(m_generator.get());
04307             }
04308         }
04309     }
04310     std::vector<T> const& get() const override {
04311         return m_chunk;
04312     }
04313     bool next() override {
04314         m_chunk.clear();
04315         for (size_t idx = 0; idx < m_chunk_size; ++idx) {
04316             if (!m_generator.next()) {
04317                 return false;
04318             }
04319             m_chunk.push_back(m_generator.get());
04320         }
04321         return true;
04322     }
04323 };

```

```

04324
04325     template <typename T>
04326     GeneratorWrapper<std::vector<T>> chunk(size_t size, GeneratorWrapper<T>&& generator) {
04327         return GeneratorWrapper<std::vector<T>> (
04328             pf::make_unique<ChunkGenerator<T>>(size, std::move(generator))
04329         );
04330     }
04331
04332 } // namespace Generators
04333 } // namespace Catch
04334
04335 // end catch_generators_generic.hpp
04336 // start catch_generators_specific.hpp
04337
04338 // start catch_context.h
04339
04340 #include <memory>
04341
04342 namespace Catch {
04343
04344     struct IResultCapture;
04345     struct IRunner;
04346     struct IConfig;
04347     struct IMutableContext;
04348
04349     using IConfigPtr = std::shared_ptr<IConfig const>;
04350
04351     struct IContext
04352     {
04353         virtual ~IContext();
04354
04355         virtual IResultCapture* getResultCapture() = 0;
04356         virtual IRunner* getRunner() = 0;
04357         virtual IConfigPtr const& getConfig() const = 0;
04358     };
04359
04360     struct IMutableContext : IContext
04361     {
04362         virtual ~IMutableContext();
04363         virtual void setResultCapture( IResultCapture* resultCapture ) = 0;
04364         virtual void setRunner( IRunner* runner ) = 0;
04365         virtual void setConfig( IConfigPtr const& config ) = 0;
04366
04367     private:
04368         static IMutableContext *currentContext;
04369         friend IMutableContext& getCurrentMutableContext();
04370         friend void cleanUpContext();
04371         static void createContext();
04372     };
04373
04374     inline IMutableContext& getCurrentMutableContext()
04375     {
04376         if ( !IMutableContext::currentContext )
04377             IMutableContext::createContext();
04378         // NOLINTNEXTLINE(clang-analyzer-core.uninitialized.UndefReturn)
04379         return *IMutableContext::currentContext;
04380     }
04381
04382     inline IContext& getCurrentContext()
04383     {
04384         return getCurrentMutableContext();
04385     }
04386
04387     void cleanUpContext();
04388
04389     class SimplePcg32;
04390     SimplePcg32& rng();
04391 }
04392
04393 // end catch_context.h
04394 // start catch_interfaces_config.h
04395
04396 // start catch_option.hpp
04397
04398 namespace Catch {
04399
04400     // An optional type
04401     template<typename T>
04402     class Option {
04403     public:
04404         Option() : nullableValue( nullptr ) {}
04405         Option( T const& _value )
04406             : nullableValue( new( storage ) T( _value ) )
04407             {}
04408         Option( Option const& _other )
04409             : nullableValue( _other ? new( storage ) T( *_other ) : nullptr )
04410             {}

```

```

04411
04412     ~Option() {
04413         reset();
04414     }
04415
04416     Option& operator= ( Option const& _other ) {
04417         if( &_other != this ) {
04418             reset();
04419             if( !_other )
04420                 nullableValue = new( storage ) T( *_other );
04421         }
04422         return *this;
04423     }
04424     Option& operator = ( T const& _value ) {
04425         reset();
04426         nullableValue = new( storage ) T( _value );
04427         return *this;
04428     }
04429
04430     void reset() {
04431         if( nullableValue )
04432             nullableValue->~T();
04433         nullableValue = nullptr;
04434     }
04435
04436     T& operator*() { return *nullableValue; }
04437     T const& operator*() const { return *nullableValue; }
04438     T* operator->() { return nullableValue; }
04439     const T* operator->() const { return nullableValue; }
04440
04441     T valueOr( T const& defaultValue ) const {
04442         return nullableValue ? *nullableValue : defaultValue;
04443     }
04444
04445     bool some() const { return nullableValue != nullptr; }
04446     bool none() const { return nullableValue == nullptr; }
04447
04448     bool operator !() const { return nullableValue == nullptr; }
04449     explicit operator bool() const {
04450         return some();
04451     }
04452
04453     private:
04454         T *nullableValue;
04455         alignas(alignof(T)) char storage[sizeof(T)];
04456     };
04457
04458 } // end namespace Catch
04459
04460 // end catch_option.hpp
04461 #include <chrono>
04462 #include <iosfwd>
04463 #include <string>
04464 #include <vector>
04465 #include <memory>
04466
04467 namespace Catch {
04468
04469     enum class Verbosity {
04470         Quiet = 0,
04471         Normal,
04472         High
04473     };
04474
04475     struct WarnAbout { enum What {
04476         Nothing = 0x00,
04477         NoAssertions = 0x01,
04478         NoTests = 0x02
04479     }; };
04480
04481     struct ShowDurations { enum OrNot {
04482         DefaultForReporter,
04483         Always,
04484         Never
04485     }; };
04486     struct RunTests { enum InWhatOrder {
04487         InDeclarationOrder,
04488         InLexicographicalOrder,
04489         InRandomOrder
04490     }; };
04491     struct UseColour { enum YesOrNo {
04492         Auto,
04493         Yes,
04494         No
04495     }; };
04496     struct WaitForKeypress { enum When {
04497         Never,

```

```

04498     BeforeStart = 1,
04499     BeforeExit = 2,
04500     BeforeStartAndExit = BeforeStart | BeforeExit
04501 }; };
04502
04503 class TestSpec;
04504
04505 struct IConfig : NonCopyable {
04506
04507     virtual ~IConfig();
04508
04509     virtual bool allowThrows() const = 0;
04510     virtual std::ostream& stream() const = 0;
04511     virtual std::string name() const = 0;
04512     virtual bool includeSuccessfulResults() const = 0;
04513     virtual bool shouldDebugBreak() const = 0;
04514     virtual bool warnAboutMissingAssertions() const = 0;
04515     virtual bool warnAboutNoTests() const = 0;
04516     virtual int abortAfter() const = 0;
04517     virtual bool showInvisibles() const = 0;
04518     virtual ShowDurations::OrNot showDurations() const = 0;
04519     virtual double minDuration() const = 0;
04520     virtual TestSpec const& testSpec() const = 0;
04521     virtual bool hasTestFilters() const = 0;
04522     virtual std::vector<std::string> const& getTestsOrTags() const = 0;
04523     virtual RunTests::InWhatOrder runOrder() const = 0;
04524     virtual unsigned int rngSeed() const = 0;
04525     virtual UseColour::YesOrNo useColour() const = 0;
04526     virtual std::vector<std::string> const& getSectionsToRun() const = 0;
04527     virtual Verbosity verbosity() const = 0;
04528
04529     virtual bool benchmarkNoAnalysis() const = 0;
04530     virtual int benchmarkSamples() const = 0;
04531     virtual double benchmarkConfidenceInterval() const = 0;
04532     virtual unsigned int benchmarkResamples() const = 0;
04533     virtual std::chrono::milliseconds benchmarkWarmupTime() const = 0;
04534 };
04535
04536 using IConfigPtr = std::shared_ptr<IConfig const>;
04537 }
04538
04539 // end catch_interfaces_config.h
04540 // start catch_random_number_generator.h
04541
04542 #include <cstdint>
04543
04544 namespace Catch {
04545
04546     // This is a simple implementation of C++11 Uniform Random Number
04547     // Generator. It does not provide all operators, because Catch2
04548     // does not use it, but it should behave as expected inside stdlib's
04549     // distributions.
04550     // The implementation is based on the PCG family (http://pcg-random.org)
04551     class SimplePcg32 {
04552     public:
04553         using state_type = std::uint64_t;
04554         using result_type = std::uint32_t;
04555         static constexpr result_type (min)() {
04556             return 0;
04557         }
04558         static constexpr result_type (max)() {
04559             return static_cast<result_type>(-1);
04560         }
04561
04562         // Provide some default initial state for the default constructor
04563         SimplePcg32():SimplePcg32(0xed743cc4U) {}
04564
04565         explicit SimplePcg32(result_type seed_);
04566
04567         void seed(result_type seed_);
04568         void discard(uint64_t skip);
04569
04570         result_type operator()();
04571
04572     private:
04573         friend bool operator==(SimplePcg32 const& lhs, SimplePcg32 const& rhs);
04574         friend bool operator!=(SimplePcg32 const& lhs, SimplePcg32 const& rhs);
04575
04576         // In theory we also need operator< and operator>
04577         // In practice we do not use them, so we will skip them for now
04578
04579         std::uint64_t m_state;
04580         // This part of the state determines which "stream" of the numbers
04581         // is chosen -- we take it as a constant for Catch2, so we only
04582         // need to deal with seeding the main state.
04583         // Picked by reading 8 bytes from `/dev/random` :-)
04584         static const std::uint64_t s_inc = (0x13ed0cc53f939476ULL « 1ULL) | 1ULL;

```

```

04585     };
04586
04587 } // end namespace Catch
04588
04589 // end catch_random_number_generator.h
04590 #include <random>
04591
04592 namespace Catch {
04593 namespace Generators {
04594
04595     template <typename Float>
04596     class RandomFloatingGenerator final : public IGenerator<Float> {
04597     public:
04598         Catch::SimplePcg32& m_rng;
04599         std::uniform_real_distribution<Float> m_dist;
04600         Float m_current_number;
04601
04602         RandomFloatingGenerator(Float a, Float b):
04603             m_rng(rng()),
04604             m_dist(a, b) {
04605             static_cast<void>(next());
04606         }
04607
04608         Float const& get() const override {
04609             return m_current_number;
04610         }
04611         bool next() override {
04612             m_current_number = m_dist(m_rng);
04613             return true;
04614         }
04615     };
04616
04617     template <typename Integer>
04618     class RandomIntegerGenerator final : public IGenerator<Integer> {
04619     public:
04620         Catch::SimplePcg32& m_rng;
04621         std::uniform_int_distribution<Integer> m_dist;
04622         Integer m_current_number;
04623
04624         RandomIntegerGenerator(Integer a, Integer b):
04625             m_rng(rng()),
04626             m_dist(a, b) {
04627             static_cast<void>(next());
04628         }
04629
04630         Integer const& get() const override {
04631             return m_current_number;
04632         }
04633         bool next() override {
04634             m_current_number = m_dist(m_rng);
04635             return true;
04636         }
04637     };
04638
04639     // TODO: Ideally this would be also constrained against the various char types,
04640     // but I don't expect users to run into that in practice.
04641     template <typename T>
04642     typename std::enable_if<std::is_integral<T>::value && !std::is_same<T, bool>::value,
04643     GeneratorWrapper<T>>::type
04644     random(T a, T b) {
04645         return GeneratorWrapper<T>{
04646             pf::make_unique<RandomIntegerGenerator<T>>(a, b)
04647         };
04648     }
04649
04650     template <typename T>
04651     typename std::enable_if<std::is_floating_point<T>::value,
04652     GeneratorWrapper<T>>::type
04653     random(T a, T b) {
04654         return GeneratorWrapper<T>{
04655             pf::make_unique<RandomFloatingGenerator<T>>(a, b)
04656         };
04657     }
04658
04659     template <typename T>
04660     class RangeGenerator final : public IGenerator<T> {
04661     public:
04662         T m_current;
04663         T m_end;
04664         T m_step;
04665         bool m_positive;
04666
04667         RangeGenerator(T const& start, T const& end, T const& step):
04668             m_current(start),
04669             m_end(end),
04670             m_step(step),
04671             m_positive(m_step > T(0))
04672     };

```

```

04672     {
04673         assert(m_current != m_end && "Range start and end cannot be equal");
04674         assert(m_step != T(0) && "Step size cannot be zero");
04675         assert(((m_positive && m_current <= m_end) || (!m_positive && m_current >= m_end)) && "Step
moves away from end");
04676     }
04677     RangeGenerator(T const& start, T const& end):
04678     RangeGenerator(start, end, (start < end) ? T(1) : T(-1))
04679     {}
04680
04681     T const& get() const override {
04682         return m_current;
04683     }
04684
04685     bool next() override {
04686         m_current += m_step;
04687         return (m_positive) ? (m_current < m_end) : (m_current > m_end);
04688     }
04689 };
04690
04691 template <typename T>
04692 GeneratorWrapper<T> range(T const& start, T const& end, T const& step) {
04693     static_assert(std::is_arithmetic<T>::value && !std::is_same<T, bool>::value, "Type must be
numeric");
04694     return GeneratorWrapper<T>(pf::make_unique<RangeGenerator<T>>(start, end, step));
04695 }
04696
04697 template <typename T>
04698 GeneratorWrapper<T> range(T const& start, T const& end) {
04699     static_assert(std::is_integral<T>::value && !std::is_same<T, bool>::value, "Type must be an
integer");
04700     return GeneratorWrapper<T>(pf::make_unique<RangeGenerator<T>>(start, end));
04701 }
04702
04703 template <typename T>
04704 class IteratorGenerator final : public IGenerator<T> {
04705     static_assert(!std::is_same<T, bool>::value,
04706         "IteratorGenerator currently does not support bools"
04707         "because of std::vector<bool> specialization");
04708
04709     std::vector<T> m_elems;
04710     size_t m_current = 0;
04711 public:
04712     template <typename InputIterator, typename InputSentinel>
04713     IteratorGenerator(InputIterator first, InputSentinel last): m_elems(first, last) {
04714         if (m_elems.empty()) {
04715             Catch::throw_exception(GeneratorException("IteratorGenerator received no valid values"));
04716         }
04717     }
04718
04719     T const& get() const override {
04720         return m_elems[m_current];
04721     }
04722
04723     bool next() override {
04724         ++m_current;
04725         return m_current != m_elems.size();
04726     }
04727 };
04728
04729 template <typename InputIterator,
04730     typename InputSentinel,
04731     typename ResultType = typename std::iterator_traits<InputIterator>::value_type>
04732 GeneratorWrapper<ResultType> from_range(InputIterator from, InputSentinel to) {
04733     return GeneratorWrapper<ResultType>(pf::make_unique<IteratorGenerator<ResultType>>(from, to));
04734 }
04735
04736 template <typename Container,
04737     typename ResultType = typename Container::value_type>
04738 GeneratorWrapper<ResultType> from_range(Container const& cnt) {
04739     return GeneratorWrapper<ResultType>(pf::make_unique<IteratorGenerator<ResultType>>(cnt.begin(),
cnt.end()));
04740 }
04741
04742 } // namespace Generators
04743 } // namespace Catch
04744
04745 // end catch_generators_specific.hpp
04746
04747 // These files are included here so the single_include script doesn't put them
04748 // in the conditionally compiled sections
04749 // start catch_test_case_info.h
04750
04751 #include <string>
04752 #include <vector>
04753 #include <memory>

```

```

04755
04756 #ifdef __clang__
04757 #pragma clang diagnostic push
04758 #pragma clang diagnostic ignored "-Wpadded"
04759 #endif
04760
04761 namespace Catch {
04762
04763     struct ITestInvoker;
04764
04765     struct TestCaseInfo {
04766         enum SpecialProperties{
04767             None = 0,
04768             IsHidden = 1 << 1,
04769             ShouldFail = 1 << 2,
04770             MayFail = 1 << 3,
04771             Throws = 1 << 4,
04772             NonPortable = 1 << 5,
04773             Benchmark = 1 << 6
04774         };
04775
04776         TestCaseInfo( std::string const& _name,
04777                     std::string const& _className,
04778                     std::string const& _description,
04779                     std::vector<std::string> const& _tags,
04780                     SourceLineInfo const& _lineInfo );
04781
04782         friend void setTags( TestCaseInfo& testCaseInfo, std::vector<std::string> tags );
04783
04784         bool isHidden() const;
04785         bool throws() const;
04786         bool okToFail() const;
04787         bool expectedToFail() const;
04788
04789         std::string tagsAsString() const;
04790
04791         std::string name;
04792         std::string className;
04793         std::string description;
04794         std::vector<std::string> tags;
04795         std::vector<std::string> lcCaseTags;
04796         SourceLineInfo lineInfo;
04797         SpecialProperties properties;
04798     };
04799
04800     class TestCase : public TestCaseInfo {
04801     public:
04802
04803         TestCase( ITestInvoker* testCase, TestCaseInfo&& info );
04804
04805         TestCase withName( std::string const& _newName ) const;
04806
04807         void invoke() const;
04808
04809         TestCaseInfo const& getTestCaseInfo() const;
04810
04811         bool operator == ( TestCase const& other ) const;
04812         bool operator < ( TestCase const& other ) const;
04813
04814     private:
04815         std::shared_ptr<ITestInvoker> test;
04816     };
04817
04818     TestCase makeTestCase( ITestInvoker* testCase,
04819                          std::string const& className,
04820                          NameAndTags const& nameAndTags,
04821                          SourceLineInfo const& lineInfo );
04822 }
04823
04824 #ifdef __clang__
04825 #pragma clang diagnostic pop
04826 #endif
04827
04828 // end catch_test_case_info.h
04829 // start catch_interfaces_runner.h
04830
04831 namespace Catch {
04832
04833     struct IRunner {
04834         virtual ~IRunner();
04835         virtual bool aborting() const = 0;
04836     };
04837 }
04838
04839 // end catch_interfaces_runner.h
04840
04841 #ifdef __OBJC__

```



```

04842 // start catch_objc.hpp
04843
04844 #import <objc/runtime.h>
04845
04846 #include <string>
04847
04848 // NB. Any general catch headers included here must be included
04849 // in catch.hpp first to make sure they are included by the single
04850 // header for non objc-usage
04851
04852 // This protocol is really only here for (self) documenting purposes, since
04853 // all its methods are optional.
04854 @protocol OcFixture
04855 @optional
04856 -(void) setUp;
04857 -(void) tearDown;
04858 @end
04859
04860 namespace Catch {
04861
04862     class OcMethod : public ITestInvoker {
04863     public:
04864         OcMethod( Class cls, SEL sel ) : m_cls( cls ), m_sel( sel ) {}
04865
04866         virtual void invoke() const {
04867             id obj = [[m_cls alloc] init];
04868
04869             performOptionalSelector( obj, @selector(setUp) );
04870             performOptionalSelector( obj, m_sel );
04871             performOptionalSelector( obj, @selector(tearDown) );
04872
04873             arcSafeRelease( obj );
04874         }
04875     private:
04876         virtual ~OcMethod() {}
04877
04878         Class m_cls;
04879         SEL m_sel;
04880     };
04881
04882     namespace Detail{
04883
04884         inline std::string getAnnotation( Class cls,
04885                                         std::string const& annotationName,
04886                                         std::string const& testCaseName ) {
04887             NSString* selStr = [[NSString alloc] initWithFormat:@"Catch_%s_%s",
04888                             annotationName.c_str(), testCaseName.c_str()];
04889             SEL sel = NSSelectorFromString( selStr );
04890             arcSafeRelease( selStr );
04891             id value = performOptionalSelector( cls, sel );
04892             if( value )
04893                 return [(NSString*)value UTF8String];
04894             return "";
04895         }
04896     }
04897
04898     inline std::size_t registerTestMethods() {
04899         std::size_t noTestMethods = 0;
04900         int noClasses = objc_getClassList( nullptr, 0 );
04901
04902         Class* classes = (CATCH_UNSAFE_UNRETAINED Class *)malloc( sizeof(Class) * noClasses);
04903         objc_getClassList( classes, noClasses );
04904
04905         for( int c = 0; c < noClasses; c++ ) {
04906             Class cls = classes[c];
04907             {
04908                 u_int count;
04909                 Method* methods = class_copyMethodList( cls, &count );
04910                 for( u_int m = 0; m < count ; m++ ) {
04911                     SEL selector = method_getName(methods[m]);
04912                     std::string methodName = sel_getName(selector);
04913                     if( startsWith( methodName, "Catch_TestCase_" ) ) {
04914                         std::string testName = methodName.substr( 15 );
04915                         std::string desc = Detail::getAnnotation( cls, "Name", testCaseName );
04916                         std::string desc2 = Detail::getAnnotation( cls, "Description", testCaseName );
04917                         const char* className = class_getName( cls );
04918
04919                         getMutableRegistryHub().registerTest( makeTestCase( new OcMethod( cls,
04920                             selector ), className, NameAndTags( name.c_str(), desc.c_str() ), SourceLineInfo("",0) ) );
04921                         noTestMethods++;
04922                     }
04923                 }
04924             }
04925             free(methods);
04926         }
04927     }

```

```

04928     }
04929     }
04930     return noTestMethods;
04931 }
04932
04933 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
04934 namespace Matchers {
04935     namespace Impl {
04936         namespace NSStringMatchers {
04937             struct StringHolder : MatcherBase<NSString*>{
04938                 StringHolder( NSString* substr ) : m_substr( [substr copy] ){}
04939                 StringHolder( StringHolder const& other ) : m_substr( [other.m_substr copy] ){}
04940                 StringHolder() {
04941                     arcSafeRelease( m_substr );
04942                 }
04943                 bool match( NSString* str ) const override {
04944                     return false;
04945                 }
04946                 NSString* CATCH_ARC_STRONG m_substr;
04947             };
04948             struct Equals : StringHolder {
04949                 Equals( NSString* substr ) : StringHolder( substr ){}
04950                 bool match( NSString* str ) const override {
04951                     return (str != nil || m_substr == nil) &&
04952                             [str isEqualToString:m_substr];
04953                 }
04954                 std::string describe() const override {
04955                     return "equals string: " + Catch::Detail::stringify( m_substr );
04956                 }
04957             };
04958             struct Contains : StringHolder {
04959                 Contains( NSString* substr ) : StringHolder( substr ){}
04960                 bool match( NSString* str ) const override {
04961                     return (str != nil || m_substr == nil) &&
04962                             [str rangeOfString:m_substr].location != NSNotFound;
04963                 }
04964                 std::string describe() const override {
04965                     return "contains string: " + Catch::Detail::stringify( m_substr );
04966                 }
04967             };
04968             struct StartsWith : StringHolder {
04969                 StartsWith( NSString* substr ) : StringHolder( substr ){}
04970                 bool match( NSString* str ) const override {
04971                     return (str != nil || m_substr == nil) &&
04972                             [str rangeOfString:m_substr].location == 0;
04973                 }
04974                 std::string describe() const override {
04975                     return "starts with: " + Catch::Detail::stringify( m_substr );
04976                 }
04977             };
04978             struct EndsWith : StringHolder {
04979                 EndsWith( NSString* substr ) : StringHolder( substr ){}
04980                 bool match( NSString* str ) const override {
04981                     return (str != nil || m_substr == nil) &&
04982                             [str rangeOfString:m_substr].location == [str length] - [m_substr length];
04983                 }
04984                 std::string describe() const override {
04985                     return "ends with: " + Catch::Detail::stringify( m_substr );
04986                 }
04987             };
04988         }
04989     }
04990 } // namespace NSStringMatchers
04991 } // namespace Impl
04992
04993 inline Impl::NSStringMatchers::Equals
04994     Equals( NSString* substr ){ return Impl::NSStringMatchers::Equals( substr ); }
04995
04996 inline Impl::NSStringMatchers::Contains
04997     Contains( NSString* substr ){ return Impl::NSStringMatchers::Contains( substr ); }
04998
04999 inline Impl::NSStringMatchers::StartsWith
05000     StartsWith( NSString* substr ){ return Impl::NSStringMatchers::StartsWith( substr ); }
05001
05002
05003
05004
05005
05006
05007
05008
05009
05010
05011
05012
05013
05014

```

```

05015
05016     inline Impl::NSStringMatchers::EndsWith
05017         EndsWith( NSString* substr ){ return Impl::NSStringMatchers::EndsWith( substr ); }
05018
05019     } // namespace Matchers
05020
05021     using namespace Matchers;
05022
05023 #endif // CATCH_CONFIG_DISABLE_MATCHERS
05024
05025 } // namespace Catch
05026
05027 #define OC_MAKE_UNIQUE_NAME( root, uniqueSuffix ) root##uniqueSuffix
05028 #define OC_TEST_CASE2( name, desc, uniqueSuffix ) \
05029     +(NSString*) OC_MAKE_UNIQUE_NAME( Catch_Name_test_, uniqueSuffix ) \
05030 { \
05031     return @ name; \
05032 } \
05033 +(NSString*) OC_MAKE_UNIQUE_NAME( Catch_Description_test_, uniqueSuffix ) \
05034 { \
05035     return @ desc; \
05036 } \
05037 -(void) OC_MAKE_UNIQUE_NAME( Catch_TestCase_test_, uniqueSuffix )
05038
05039 #define OC_TEST_CASE( name, desc ) OC_TEST_CASE2( name, desc, __LINE__ )
05040
05041 // end catch_objc.hpp
05042 #endif
05043
05044 // Benchmarking needs the externally-facing parts of reporters to work
05045 #if defined(CATCH_CONFIG_EXTERNAL_INTERFACES) || defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
05046 // start catch_external_interfaces.h
05047
05048 // start catch_reporter_bases.hpp
05049
05050 // start catch_interfaces_reporter.h
05051
05052 // start catch_config.hpp
05053
05054 // start catch_test_spec_parser.h
05055
05056 #ifdef __clang__
05057 #pragma clang diagnostic push
05058 #pragma clang diagnostic ignored "-Wpadded"
05059 #endif
05060
05061 // start catch_test_spec.h
05062
05063 #ifdef __clang__
05064 #pragma clang diagnostic push
05065 #pragma clang diagnostic ignored "-Wpadded"
05066 #endif
05067
05068 // start catch_wildcard_pattern.h
05069
05070 namespace Catch {
05071 {
05072     class WildcardPattern {
05073     public:
05074         enum WildcardPosition {
05075             NoWildcard = 0,
05076             WildcardAtStart = 1,
05077             WildcardAtEnd = 2,
05078             WildcardAtBothEnds = WildcardAtStart | WildcardAtEnd
05079         };
05080
05081         WildcardPattern( std::string const& pattern, CaseSensitive::Choice caseSensitivity );
05082         virtual ~WildcardPattern() = default;
05083         virtual bool matches( std::string const& str ) const;
05084
05085     private:
05086         std::string normaliseString( std::string const& str ) const;
05087         CaseSensitive::Choice m_caseSensitivity;
05088         WildcardPosition m_wildcard = NoWildcard;
05089         std::string m_pattern;
05090     };
05091 }
05092
05093 // end catch_wildcard_pattern.h
05094
05095 #include <string>
05096 #include <vector>
05097 #include <memory>
05098
05099 namespace Catch {
05100     struct IConfig;

```

```

05103
05104     class TestSpec {
05105     class Pattern {
05106     public:
05107         explicit Pattern( std::string const& name );
05108         virtual ~Pattern();
05109         virtual bool matches( TestCaseInfo const& testCase ) const = 0;
05110         std::string const& name() const;
05111     private:
05112         std::string const m_name;
05113     };
05114     using PatternPtr = std::shared_ptr<Pattern>;
05115
05116     class NamePattern : public Pattern {
05117     public:
05118         explicit NamePattern( std::string const& name, std::string const& filterString );
05119         bool matches( TestCaseInfo const& testCase ) const override;
05120     private:
05121         WildcardPattern m_wildcardPattern;
05122     };
05123
05124     class TagPattern : public Pattern {
05125     public:
05126         explicit TagPattern( std::string const& tag, std::string const& filterString );
05127         bool matches( TestCaseInfo const& testCase ) const override;
05128     private:
05129         std::string m_tag;
05130     };
05131
05132     class ExcludedPattern : public Pattern {
05133     public:
05134         explicit ExcludedPattern( PatternPtr const& underlyingPattern );
05135         bool matches( TestCaseInfo const& testCase ) const override;
05136     private:
05137         PatternPtr m_underlyingPattern;
05138     };
05139
05140     struct Filter {
05141         std::vector<PatternPtr> m_patterns;
05142
05143         bool matches( TestCaseInfo const& testCase ) const;
05144         std::string name() const;
05145     };
05146
05147     public:
05148         struct FilterMatch {
05149             std::string name;
05150             std::vector<TestCase const*> tests;
05151         };
05152         using Matches = std::vector<FilterMatch>;
05153         using vectorStrings = std::vector<std::string>;
05154
05155         bool hasFilters() const;
05156         bool matches( TestCaseInfo const& testCase ) const;
05157         Matches matchesByFilter( std::vector<TestCase> const& testCases, IConfig const& config )
05158     const;
05159         const vectorStrings & getInvalidArgs() const;
05160     private:
05161         std::vector<Filter> m_filters;
05162         std::vector<std::string> m_invalidArgs;
05163         friend class TestSpecParser;
05164     };
05165 }
05166
05167 #ifdef __clang__
05168 #pragma clang diagnostic pop
05169 #endif
05170
05171 // end catch_test_spec.h
05172 // start catch_interfaces_tag_alias_registry.h
05173
05174 #include <string>
05175
05176 namespace Catch {
05177
05178     struct TagAlias;
05179
05180     struct ITagAliasRegistry {
05181         virtual ~ITagAliasRegistry();
05182         // Nullptr if not present
05183         virtual TagAlias const* find( std::string const& alias ) const = 0;
05184         virtual std::string expandAliases( std::string const& unexpandedTestSpec ) const = 0;
05185
05186         static ITagAliasRegistry const& get();
05187     };
05188

```

```

05189 } // end namespace Catch
05190
05191 // end catch_interfaces_tag_alias_registry.h
05192 namespace Catch {
05193
05194     class TestSpecParser {
05195     public:
05196         enum Mode{ None, Name, QuotedName, Tag, EscapedName };
05197         Mode m_mode = None;
05198         Mode lastMode = None;
05199         bool m_exclusion = false;
05200         std::size_t m_pos = 0;
05201         std::size_t m_realPatternPos = 0;
05202         std::string m_arg;
05203         std::string m_substring;
05204         std::string m_patternName;
05205         std::vector<std::size_t> m_escapeChars;
05206         TestSpec::Filter m_currentFilter;
05207         TestSpec m_testSpec;
05208         ITagAliasRegistry const* m_tagAliases = nullptr;
05209
05210     public:
05211         TestSpecParser( ITagAliasRegistry const& tagAliases );
05212
05213         TestSpecParser& parse( std::string const& arg );
05214         TestSpec testSpec();
05215
05216     private:
05217         bool visitChar( char c );
05218         void startNewMode( Mode mode );
05219         bool processNoneChar( char c );
05220         void processNameChar( char c );
05221         bool processOtherChar( char c );
05222         void endMode();
05223         void escape();
05224         bool isControlChar( char c ) const;
05225         void saveLastMode();
05226         void revertBackToLastMode();
05227         void addFilter();
05228         bool separate();
05229
05230         // Handles common preprocessing of the pattern for name/tag patterns
05231         std::string preprocessPattern();
05232         // Adds the current pattern as a test name
05233         void addNamePattern();
05234         // Adds the current pattern as a tag
05235         void addTagPattern();
05236
05237         inline void addCharToPattern(char c) {
05238             m_substring += c;
05239             m_patternName += c;
05240             m_realPatternPos++;
05241         }
05242     };
05243
05244     TestSpec parseTestSpec( std::string const& arg );
05245 } // namespace Catch
05246
05247 #ifdef __clang__
05248 #pragma clang diagnostic pop
05249 #endif
05250
05251 // end catch_test_spec_parser.h
05252 // Libstdc++ doesn't like incomplete classes for unique_ptr
05253
05254 #include <memory>
05255 #include <vector>
05256 #include <string>
05257
05258 #ifndef CATCH_CONFIG_CONSOLE_WIDTH
05259 #define CATCH_CONFIG_CONSOLE_WIDTH 80
05260 #endif
05261
05262 namespace Catch {
05263
05264     struct IStream;
05265
05266     struct ConfigData {
05267         bool listTests = false;
05268         bool listTags = false;
05269         bool listReporters = false;
05270         bool listTestNamesOnly = false;
05271
05272         bool showSuccessfulTests = false;
05273         bool shouldDebugBreak = false;
05274         bool noThrow = false;
05275         bool showHelp = false;

```

```

05276     bool showInvisibles = false;
05277     bool filenamesAsTags = false;
05278     bool libIdentify = false;
05279
05280     int abortAfter = -1;
05281     unsigned int rngSeed = 0;
05282
05283     bool benchmarkNoAnalysis = false;
05284     unsigned int benchmarkSamples = 100;
05285     double benchmarkConfidenceInterval = 0.95;
05286     unsigned int benchmarkResamples = 100000;
05287     std::chrono::milliseconds::rep benchmarkWarmupTime = 100;
05288
05289     Verbosity verbosity = Verbosity::Normal;
05290     WarnAbout::What warnings = WarnAbout::Nothing;
05291     ShowDurations::OrNot showDurations = ShowDurations::DefaultForReporter;
05292     double minDuration = -1;
05293     RunTests::InWhatOrder runOrder = RunTests::InDeclarationOrder;
05294     UseColour::YesOrNo useColour = UseColour::Auto;
05295     WaitForKeypress::When waitForKeypress = WaitForKeypress::Never;
05296
05297     std::string outputFilename;
05298     std::string name;
05299     std::string processName;
05300 #ifndef CATCH_CONFIG_DEFAULT_REPORTER
05301 #define CATCH_CONFIG_DEFAULT_REPORTER "console"
05302 #endif
05303     std::string reporterName = CATCH_CONFIG_DEFAULT_REPORTER;
05304 #undef CATCH_CONFIG_DEFAULT_REPORTER
05305
05306     std::vector<std::string> testsOrTags;
05307     std::vector<std::string> sectionsToRun;
05308 };
05309
05310 class Config : public IConfig {
05311 public:
05312
05313     Config() = default;
05314     Config( ConfigData const& data );
05315     virtual ~Config() = default;
05316
05317     std::string const& getFilename() const;
05318
05319     bool listTests() const;
05320     bool listTestNamesOnly() const;
05321     bool listTags() const;
05322     bool listReporters() const;
05323
05324     std::string getProcessName() const;
05325     std::string const& getReporterName() const;
05326
05327     std::vector<std::string> const& getTestsOrTags() const override;
05328     std::vector<std::string> const& getSectionsToRun() const override;
05329
05330     TestSpec const& testSpec() const override;
05331     bool hasTestFilters() const override;
05332
05333     bool showHelp() const;
05334
05335     // IConfig interface
05336     bool allowThrows() const override;
05337     std::ostream& stream() const override;
05338     std::string name() const override;
05339     bool includeSuccessfulResults() const override;
05340     bool warnAboutMissingAssertions() const override;
05341     bool warnAboutNoTests() const override;
05342     ShowDurations::OrNot showDurations() const override;
05343     double minDuration() const override;
05344     RunTests::InWhatOrder runOrder() const override;
05345     unsigned int rngSeed() const override;
05346     UseColour::YesOrNo useColour() const override;
05347     bool shouldDebugBreak() const override;
05348     int abortAfter() const override;
05349     bool showInvisibles() const override;
05350     Verbosity verbosity() const override;
05351     bool benchmarkNoAnalysis() const override;
05352     int benchmarkSamples() const override;
05353     double benchmarkConfidenceInterval() const override;
05354     unsigned int benchmarkResamples() const override;
05355     std::chrono::milliseconds benchmarkWarmupTime() const override;
05356
05357 private:
05358
05359     IStream const* openStream();
05360     ConfigData m_data;
05361
05362     std::unique_ptr<IStream const> m_stream;

```

```

05363         TestSpec m_testSpec;
05364         bool m_hasTestFilters = false;
05365     };
05366
05367 } // end namespace Catch
05368
05369 // end catch_config.hpp
05370 // start catch_assertionresult.h
05371
05372 #include <string>
05373
05374 namespace Catch {
05375     struct AssertionResultData
05376     {
05377         AssertionResultData() = delete;
05378
05379         AssertionResultData( ResultWas::OfType _resultType, LazyExpression const& _lazyExpression );
05380
05381         std::string message;
05382         mutable std::string reconstructedExpression;
05383         LazyExpression lazyExpression;
05384         ResultWas::OfType resultType;
05385
05386         std::string reconstructExpression() const;
05387     };
05388
05389     class AssertionResult {
05390     public:
05391         AssertionResult() = delete;
05392         AssertionResult( AssertionInfo const& info, AssertionResultData const& data );
05393
05394         bool isOk() const;
05395         bool succeeded() const;
05396         ResultWas::OfType getResultType() const;
05397         bool hasExpression() const;
05398         bool hasMessage() const;
05399         std::string getExpression() const;
05400         std::string getExpressionInMacro() const;
05401         bool hasExpandedExpression() const;
05402         std::string getExpandedExpression() const;
05403         std::string getMessage() const;
05404         SourceLineInfo getSourceInfo() const;
05405         StringRef getTestMacroName() const;
05406
05407     protected:
05408         AssertionInfo m_info;
05409         AssertionResultData m_resultData;
05410     };
05411
05412 } // end namespace Catch
05413
05414 // end catch_assertionresult.h
05415 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
05416 // start catch_estimate.hpp
05417
05418 // Statistics estimates
05419
05420 namespace Catch {
05421     namespace Benchmark {
05422         template <typename Duration>
05423         struct Estimate {
05424             Duration point;
05425             Duration lower_bound;
05426             Duration upper_bound;
05427             double confidence_interval;
05428
05429             template <typename Duration2>
05430             operator Estimate<Duration2>() const {
05431                 return { point, lower_bound, upper_bound, confidence_interval };
05432             }
05433         };
05434     }
05435 }
05436 // namespace Benchmark
05437 } // namespace Catch
05438
05439 // end catch_estimate.hpp
05440 // start catch_outlier_classification.hpp
05441
05442 // Outlier information
05443 namespace Catch {
05444     namespace Benchmark {
05445         struct OutlierClassification {
05446             int samples_seen = 0;
05447             int low_severe = 0; // more than 3 times IQR below Q1
05448             int low_mild = 0; // 1.5 to 3 times IQR below Q1
05449
05450             int high_mild = 0; // 1.5 to 3 times IQR above Q3
05451             int high_severe = 0; // more than 3 times IQR above Q3
05452
05453             bool is_outlier() const {
05454                 return low_severe > 0 || low_mild > 0 || high_mild > 0 || high_severe > 0;
05455             }
05456
05457             bool is_significant() const {
05458                 return low_severe > 0 || high_severe > 0;
05459             }
05460         };
05461     }
05462 }
05463
05464 // end catch_outlier_classification.hpp

```

```

05450         int high_mild = 0;        // 1.5 to 3 times IQR above Q3
05451         int high_severe = 0;      // more than 3 times IQR above Q3
05452
05453         int total() const {
05454             return low_severe + low_mild + high_mild + high_severe;
05455         }
05456     };
05457 } // namespace Benchmark
05458 } // namespace Catch
05459 // end catch_outlier_classification.hpp
05461
05462 #include <iterator>
05463 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
05464
05465 #include <string>
05466 #include <iosfwd>
05467 #include <map>
05468 #include <set>
05469 #include <memory>
05470 #include <algorithm>
05471
05472 namespace Catch {
05473
05474     struct ReporterConfig {
05475         explicit ReporterConfig( IConfigPtr const& _fullConfig );
05476
05477         ReporterConfig( IConfigPtr const& _fullConfig, std::ostream& _stream );
05478
05479         std::ostream& stream() const;
05480         IConfigPtr fullConfig() const;
05481
05482     private:
05483         std::ostream* m_stream;
05484         IConfigPtr m_fullConfig;
05485     };
05486
05487     struct ReporterPreferences {
05488         bool shouldRedirectStdOut = false;
05489         bool shouldReportAllAssertions = false;
05490     };
05491
05492     template<typename T>
05493     struct LazyStat : Option<T> {
05494         LazyStat& operator=( T const& _value ) {
05495             Option<T>::operator=( _value );
05496             used = false;
05497             return *this;
05498         }
05499         void reset() {
05500             Option<T>::reset();
05501             used = false;
05502         }
05503         bool used = false;
05504     };
05505
05506     struct TestRunInfo {
05507         TestRunInfo( std::string const& _name );
05508         std::string name;
05509     };
05510
05511     struct GroupInfo {
05512         GroupInfo( std::string const& _name,
05513                   std::size_t _groupIndex,
05514                   std::size_t _groupsCount );
05515
05516         std::string name;
05517         std::size_t groupIndex;
05518         std::size_t groupsCounts;
05519     };
05520
05521     struct AssertionStats {
05522         AssertionStats( AssertionResult const& _assertionResult,
05523                         std::vector<MessageInfo> const& _infoMessages,
05524                         Totals const& _totals );
05525
05526         AssertionStats( AssertionStats const& ) = default;
05527         AssertionStats& operator=( AssertionStats const& ) = default;
05528         AssertionStats& operator = ( AssertionStats const& ) = delete;
05529         AssertionStats& operator = ( AssertionStats const& ) = delete;
05530         virtual ~AssertionStats();
05531
05532         AssertionResult assertionResult;
05533         std::vector<MessageInfo> infoMessages;
05534         Totals totals;
05535     };
05536
05537     struct SectionStats {

```



```

05537     SectionStats( SectionInfo const& _sectionInfo,
05538                   Counts const& _assertions,
05539                   double _durationInSeconds,
05540                   bool _missingAssertions );
05541     SectionStats( SectionStats const& ) = default;
05542     SectionStats( SectionStats && ) = default;
05543     SectionStats& operator = ( SectionStats const& ) = default;
05544     SectionStats& operator = ( SectionStats && ) = default;
05545     virtual ~SectionStats();
05546
05547     SectionInfo sectionInfo;
05548     Counts assertions;
05549     double durationInSeconds;
05550     bool missingAssertions;
05551 };
05552
05553 struct TestCaseStats {
05554     TestCaseStats( TestCaseInfo const& _testInfo,
05555                   Totals const& _totals,
05556                   std::string const& _stdOut,
05557                   std::string const& _stdErr,
05558                   bool _aborting );
05559
05560     TestCaseStats( TestCaseStats const& ) = default;
05561     TestCaseStats( TestCaseStats && ) = default;
05562     TestCaseStats& operator = ( TestCaseStats const& ) = default;
05563     TestCaseStats& operator = ( TestCaseStats && ) = default;
05564     virtual ~TestCaseStats();
05565
05566     TestCaseInfo testInfo;
05567     Totals totals;
05568     std::string stdOut;
05569     std::string stdErr;
05570     bool aborting;
05571 };
05572
05573 struct TestGroupStats {
05574     TestGroupStats( GroupInfo const& _groupInfo,
05575                   Totals const& _totals,
05576                   bool _aborting );
05577     TestGroupStats( GroupInfo const& _groupInfo );
05578
05579     TestGroupStats( TestGroupStats const& ) = default;
05580     TestGroupStats( TestGroupStats && ) = default;
05581     TestGroupStats& operator = ( TestGroupStats const& ) = default;
05582     TestGroupStats& operator = ( TestGroupStats && ) = default;
05583     virtual ~TestGroupStats();
05584
05585     GroupInfo groupInfo;
05586     Totals totals;
05587     bool aborting;
05588 };
05589
05590 struct TestRunStats {
05591     TestRunStats( TestRunInfo const& _runInfo,
05592                   Totals const& _totals,
05593                   bool _aborting );
05594
05595     TestRunStats( TestRunStats const& ) = default;
05596     TestRunStats( TestRunStats && ) = default;
05597     TestRunStats& operator = ( TestRunStats const& ) = default;
05598     TestRunStats& operator = ( TestRunStats && ) = default;
05599     virtual ~TestRunStats();
05600
05601     TestRunInfo runInfo;
05602     Totals totals;
05603     bool aborting;
05604 };
05605
05606 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
05607 struct BenchmarkInfo {
05608     std::string name;
05609     double estimatedDuration;
05610     int iterations;
05611     int samples;
05612     unsigned int resamples;
05613     double clockResolution;
05614     double clockCost;
05615 };
05616
05617 template <class Duration>
05618 struct BenchmarkStats {
05619     BenchmarkInfo info;
05620
05621     std::vector<Duration> samples;
05622     Benchmark::Estimate<Duration> mean;
05623     Benchmark::Estimate<Duration> standardDeviation;

```

```

05624         Benchmark::OutlierClassification outliers;
05625         double outlierVariance;
05626
05627         template <typename Duration2>
05628         operator BenchmarkStats<Duration2>() const {
05629             std::vector<Duration2> samples2;
05630             samples2.reserve(samples.size());
05631             std::transform(samples.begin(), samples.end(), std::back_inserter(samples2), [](Duration
05632 d) { return Duration2(d); });
05633             return {
05634                 info,
05635                 std::move(samples2),
05636                 mean,
05637                 standardDeviation,
05638                 outliers,
05639                 outlierVariance,
05640             };
05641         };
05642 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
05643
05644         struct IStreamingReporter {
05645             virtual ~IStreamingReporter() = default;
05646
05647             // Implementing class must also provide the following static methods:
05648             // static std::string getDescription();
05649             // static std::set<Verbosity> getSupportedVerbsities()
05650
05651             virtual ReporterPreferences getPreferences() const = 0;
05652
05653             virtual void noMatchingTestCases( std::string const& spec ) = 0;
05654
05655             virtual void reportInvalidArguments(std::string const&) {}
05656
05657             virtual void testRunStarting( TestRunInfo const& testRunInfo ) = 0;
05658             virtual void testGroupStarting( GroupInfo const& groupInfo ) = 0;
05659
05660             virtual void testCaseStarting( TestCaseInfo const& testInfo ) = 0;
05661             virtual void sectionStarting( SectionInfo const& sectionInfo ) = 0;
05662
05663 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
05664             virtual void benchmarkPreparing( std::string const& ) {}
05665             virtual void benchmarkStarting( BenchmarkInfo const& ) {}
05666             virtual void benchmarkEnded( BenchmarkStats<> const& ) {}
05667             virtual void benchmarkFailed( std::string const& ) {}
05668 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
05669
05670             virtual void assertionStarting( AssertionInfo const& assertionInfo ) = 0;
05671
05672             // The return value indicates if the messages buffer should be cleared:
05673             virtual bool assertionEnded( AssertionStats const& assertionStats ) = 0;
05674
05675             virtual void sectionEnded( SectionStats const& sectionStats ) = 0;
05676             virtual void testCaseEnded( TestCaseStats const& testCaseStats ) = 0;
05677             virtual void testGroupEnded( TestGroupStats const& testGroupStats ) = 0;
05678             virtual void testRunEnded( TestRunStats const& testRunStats ) = 0;
05679
05680             virtual void skipTest( TestCaseInfo const& testInfo ) = 0;
05681
05682             // Default empty implementation provided
05683             virtual void fatalErrorEncountered( StringRef name );
05684
05685             virtual bool isMulti() const;
05686         };
05687         using IStreamingReporterPtr = std::unique_ptr<IStreamingReporter>;
05688
05689         struct IReporterFactory {
05690             virtual ~IReporterFactory();
05691             virtual IStreamingReporterPtr create( ReporterConfig const& config ) const = 0;
05692             virtual std::string getDescription() const = 0;
05693         };
05694         using IReporterFactoryPtr = std::shared_ptr<IReporterFactory>;
05695
05696         struct IReporterRegistry {
05697             using FactoryMap = std::map<std::string, IReporterFactoryPtr>;
05698             using Listeners = std::vector<IReporterFactoryPtr>;
05699
05700             virtual ~IReporterRegistry();
05701             virtual IStreamingReporterPtr create( std::string const& name, IConfigPtr const& config )
05702 const = 0;
05703             virtual FactoryMap const& getFactories() const = 0;
05704             virtual Listeners const& getListeners() const = 0;
05705         };
05706     } // end namespace Catch
05707
05708 // end catch_interfaces_reporter.h

```

```

05709 #include <algorithm>
05710 #include <cstring>
05711 #include <cfloat>
05712 #include <cstdio>
05713 #include <cassert>
05714 #include <memory>
05715 #include <ostream>
05716
05717 namespace Catch {
05718     void prepareExpandedExpression(AssertionResult& result);
05719
05720     // Returns double formatted as %.3f (format expected on output)
05721     std::string getFormattedDuration( double duration );
05722
05723     bool shouldShowDuration( IConfig const& config, double duration );
05724
05725     std::string serializeFilters( std::vector<std::string> const& container );
05726
05727     template<typename DerivedT>
05728     struct StreamingReporterBase : IStreamingReporter {
05729
05730         StreamingReporterBase( ReporterConfig const& _config )
05731             : m_config( _config.fullConfig() ),
05732               stream( _config.stream() ) {
05733             {
05734                 m_reporterPrefs.shouldRedirectStdOut = false;
05735                 if ( !DerivedT::getSupportedVerbsosities().count( m_config->verbosity() ) )
05736                     CATCH_ERROR( "Verbosity level not supported by this reporter" );
05737             }
05738         }
05739
05740         ReporterPreferences getPreferences() const override {
05741             return m_reporterPrefs;
05742         }
05743
05744         static std::set<Verbosity> getSupportedVerbsosities() {
05745             return { Verbosity::Normal };
05746         }
05747
05748         ~StreamingReporterBase() override = default;
05749
05750         void noMatchingTestCases( std::string const& ) override {}
05751
05752         void reportInvalidArguments( std::string const& ) override {}
05753
05754         void testRunStarting( TestRunInfo const& _testRunInfo ) override {
05755             currentTestRunInfo = _testRunInfo;
05756         }
05757
05758         void testGroupStarting( GroupInfo const& _groupInfo ) override {
05759             currentGroupInfo = _groupInfo;
05760         }
05761
05762         void testCaseStarting( TestCaseInfo const& _testInfo ) override {
05763             currentTestCaseInfo = _testInfo;
05764         }
05765         void sectionStarting( SectionInfo const& _sectionInfo ) override {
05766             m_sectionStack.push_back( _sectionInfo );
05767         }
05768
05769         void sectionEnded( SectionStats const& /* _sectionStats */ ) override {
05770             m_sectionStack.pop_back();
05771         }
05772         void testCaseEnded( TestCaseStats const& /* _testCaseStats */ ) override {
05773             currentTestCaseInfo.reset();
05774         }
05775         void testGroupEnded( TestGroupStats const& /* _testGroupStats */ ) override {
05776             currentGroupInfo.reset();
05777         }
05778         void testRunEnded( TestRunStats const& /* _testRunStats */ ) override {
05779             currentTestCaseInfo.reset();
05780             currentGroupInfo.reset();
05781             currentTestRunInfo.reset();
05782         }
05783
05784         void skipTest( TestCaseInfo const& ) override {
05785             // Don't do anything with this by default.
05786             // It can optionally be overridden in the derived class.
05787         }
05788
05789         IConfigPtr m_config;
05790         std::ostream& stream;
05791
05792         LazyStat<TestRunInfo> currentTestRunInfo;
05793         LazyStat<GroupInfo> currentGroupInfo;
05794         LazyStat<TestCaseInfo> currentTestCaseInfo;
05795
05796         std::vector<SectionInfo> m_sectionStack;

```

```

05797     ReporterPreferences m_reporterPrefs;
05798 };
05799
05800 template<typename DerivedT>
05801 struct CumulativeReporterBase : IStreamingReporter {
05802     template<typename T, typename ChildNodeT>
05803     struct Node {
05804         explicit Node( T const& _value ) : value( _value ) {}
05805         virtual ~Node() {}
05806
05807         using ChildNodes = std::vector<std::shared_ptr<ChildNodeT>;
05808         T value;
05809         ChildNodes children;
05810     };
05811     struct SectionNode {
05812         explicit SectionNode(SectionStats const& _stats) : stats(_stats) {}
05813         virtual ~SectionNode() = default;
05814
05815         bool operator == (SectionNode const& other) const {
05816             return stats.sectionInfo.lineInfo == other.stats.sectionInfo.lineInfo;
05817         }
05818         bool operator == (std::shared_ptr<SectionNode> const& other) const {
05819             return operator==( *other );
05820         }
05821
05822         SectionStats stats;
05823         using ChildSections = std::vector<std::shared_ptr<SectionNode>;
05824         using Assertions = std::vector<AssertionStats>;
05825         ChildSections childSections;
05826         Assertions assertions;
05827         std::string stdOut;
05828         std::string stdErr;
05829     };
05830
05831     struct BySectionInfo {
05832         BySectionInfo( SectionInfo const& other ) : m_other( other ) {}
05833         BySectionInfo( BySectionInfo const& other ) : m_other( other.m_other ) {}
05834         bool operator () (std::shared_ptr<SectionNode> const& node) const {
05835             return ((node->stats.sectionInfo.name == m_other.name) &&
05836                 (node->stats.sectionInfo.lineInfo == m_other.lineInfo));
05837         }
05838         void operator=(BySectionInfo const&) = delete;
05839
05840     private:
05841         SectionInfo const& m_other;
05842     };
05843
05844     using TestCaseNode = Node<TestCaseStats, SectionNode>;
05845     using TestGroupNode = Node<TestGroupStats, TestCaseNode>;
05846     using TestRunNode = Node<TestRunStats, TestGroupNode>;
05847
05848     CumulativeReporterBase( ReporterConfig const& _config )
05849     :   m_config( _config.fullConfig() ),
05850         stream( _config.stream() )
05851     {
05852         m_reporterPrefs.shouldRedirectStdOut = false;
05853         if( !DerivedT::getSupportedVerbsities().count( m_config->verbosity() ) )
05854             CATCH_ERROR( "Verbosity level not supported by this reporter" );
05855     }
05856     ~CumulativeReporterBase() override = default;
05857
05858     ReporterPreferences getPreferences() const override {
05859         return m_reporterPrefs;
05860     }
05861
05862     static std::set<Verbosity> getSupportedVerbsities() {
05863         return { Verbosity::Normal };
05864     }
05865
05866     void testRunStarting( TestRunInfo const& ) override {}
05867     void testGroupStarting( GroupInfo const& ) override {}
05868
05869     void testCaseStarting( TestCaseInfo const& ) override {}
05870
05871     void sectionStarting( SectionInfo const& sectionInfo ) override {
05872         SectionStats incompleteStats( sectionInfo, Counts(), 0, false );
05873         std::shared_ptr<SectionNode> node;
05874         if( m_sectionStack.empty() ) {
05875             if( !m_rootSection )
05876                 m_rootSection = std::make_shared<SectionNode>( incompleteStats );
05877             node = m_rootSection;
05878         }
05879         else {
05880             SectionNode& parentNode = *m_sectionStack.back();
05881             auto it =
05882                 std::find_if( parentNode.childSections.begin(),
05883                     parentNode.childSections.end(),

```

```

05884         BySectionInfo( sectionInfo ) );
05885         if( it == parentNode.childSections.end() ) {
05886             node = std::make_shared<SectionNode>( incompleteStats );
05887             parentNode.childSections.push_back( node );
05888         }
05889         else
05890             node = *it;
05891     }
05892     m_sectionStack.push_back( node );
05893     m_deepestSection = std::move(node);
05894 }
05895
05896 void assertionStarting(AssertionInfo const& override) {}
05897
05898 bool assertionEnded(AssertionStats const& assertionStats) override {
05899     assert(!m_sectionStack.empty());
05900     // AssertionResult holds a pointer to a temporary DecomposedExpression,
05901     // which getExpandedExpression() calls to build the expression string.
05902     // Our section stack copy of the assertionResult will likely outlive the
05903     // temporary, so it must be expanded or discarded now to avoid calling
05904     // a destroyed object later.
05905     prepareExpandedExpression(const_cast<AssertionResult&>( assertionStats.assertionResult ));
05906 };
05907     SectionNode& sectionNode = *m_sectionStack.back();
05908     sectionNode.assertions.push_back( assertionStats );
05909     return true;
05910 }
05911 void sectionEnded(SectionStats const& sectionStats) override {
05912     assert(!m_sectionStack.empty());
05913     SectionNode& node = *m_sectionStack.back();
05914     node.stats = sectionStats;
05915     m_sectionStack.pop_back();
05916 }
05917 void testCaseEnded(TestCaseStats const& testCaseStats) override {
05918     auto node = std::make_shared<TestCaseNode>( testCaseStats );
05919     assert(m_sectionStack.size() == 0);
05920     node->children.push_back( m_rootSection );
05921     m_testCases.push_back( node );
05922     m_rootSection.reset();
05923
05924     assert( m_deepestSection );
05925     m_deepestSection->stdOut = testCaseStats.stdOut;
05926     m_deepestSection->stdErr = testCaseStats.stdErr;
05927 }
05928 void testGroupEnded(TestGroupStats const& testGroupStats) override {
05929     auto node = std::make_shared<TestGroupNode>( testGroupStats );
05930     node->children.swap( m_testCases );
05931     m_testGroups.push_back( node );
05932 }
05933 void testRunEnded(TestRunStats const& testRunStats) override {
05934     auto node = std::make_shared<TestRunNode>( testRunStats );
05935     node->children.swap( m_testGroups );
05936     m_testRuns.push_back( node );
05937     testRunEndedCumulative();
05938 }
05939 virtual void testRunEndedCumulative() = 0;
05940
05941 void skipTest(TestCaseInfo const& override) {}
05942
05943 IConfigPtr m_config;
05944 std::ostream& stream;
05945 std::vector<AssertionStats> m_assertions;
05946 std::vector<std::vector<std::shared_ptr<SectionNode>>> m_sections;
05947 std::vector<std::shared_ptr<TestCaseNode>> m_testCases;
05948 std::vector<std::shared_ptr<TestGroupNode>> m_testGroups;
05949
05950 std::vector<std::shared_ptr<TestRunNode>> m_testRuns;
05951
05952 std::shared_ptr<SectionNode> m_rootSection;
05953 std::shared_ptr<SectionNode> m_deepestSection;
05954 std::vector<std::shared_ptr<SectionNode>> m_sectionStack;
05955 ReporterPreferences m_reporterPrefs;
05956 };
05957
05958 template<char C>
05959 char const* getLineOfChars() {
05960     static char line[CATCH_CONFIG_CONSOLE_WIDTH] = {0};
05961     if( !*line ) {
05962         std::memset( line, C, CATCH_CONFIG_CONSOLE_WIDTH-1 );
05963         line[CATCH_CONFIG_CONSOLE_WIDTH-1] = 0;
05964     }
05965     return line;
05966 }
05967
05968 struct TestEventListenerBase : StreamingReporterBase<TestEventListenerBase> {
05969     TestEventListenerBase( ReporterConfig const& _config );

```

```

05970         static std::set<Verbosity> getSupportedVerbsities();
05971
05972         void assertionStarting(AssertionInfo const&) override;
05973         bool assertionEnded(AssertionStats const&) override;
05974     };
05975
05976 } // end namespace Catch
05977
05978 // end catch_reporter_bases.hpp
05979 // start catch_console_colour.h
05980
05981 namespace Catch {
05982
05983     struct Colour {
05984         enum Code {
05985             None = 0,
05986
05987             White,
05988             Red,
05989             Green,
05990             Blue,
05991             Cyan,
05992             Yellow,
05993             Grey,
05994
05995             Bright = 0x10,
05996
05997             BrightRed = Bright | Red,
05998             BrightGreen = Bright | Green,
05999             LightGrey = Bright | Grey,
06000             BrightWhite = Bright | White,
06001             BrightYellow = Bright | Yellow,
06002
06003             // By intention
06004             FileName = LightGrey,
06005             Warning = BrightYellow,
06006             ResultError = BrightRed,
06007             ResultSuccess = BrightGreen,
06008             ResultExpectedFailure = Warning,
06009
06010             Error = BrightRed,
06011             Success = Green,
06012
06013             OriginalExpression = Cyan,
06014             ReconstructedExpression = BrightYellow,
06015
06016             SecondaryText = LightGrey,
06017             Headers = White
06018         };
06019
06020         // Use constructed object for RAII guard
06021         Colour( Code _colourCode );
06022         Colour( Colour&& other ) noexcept;
06023         Colour& operator=( Colour&& other ) noexcept;
06024         ~Colour();
06025
06026         // Use static method for one-shot changes
06027         static void use( Code _colourCode );
06028
06029     private:
06030         bool m_moved = false;
06031     };
06032
06033     std::ostream& operator << ( std::ostream& os, Colour const& );
06034
06035 } // end namespace Catch
06036
06037 // end catch_console_colour.h
06038 // start catch_reporter_registrars.hpp
06039
06040
06041 namespace Catch {
06042
06043     template<typename T>
06044     class ReporterRegistrar {
06045
06046         class ReporterFactory : public IReporterFactory {
06047
06048             IStreamingReporterPtr create( ReporterConfig const& config ) const override {
06049                 return std::unique_ptr<T>( new T( config ) );
06050             }
06051
06052             std::string getDescription() const override {
06053                 return T::getDescription();
06054             }
06055         };
06056

```

```

06057     public:
06058
06059         explicit ReporterRegistrar( std::string const& name ) {
06060             getMutableRegistryHub().registerReporter( name, std::make_shared<ReporterFactory>() );
06061         }
06062     };
06063
06064     template<typename T>
06065     class ListenerRegistrar {
06066     public:
06067         class ListenerFactory : public IReporterFactory {
06068         public:
06069             IStreamingReporterPtr create( ReporterConfig const& config ) const override {
06070                 return std::unique_ptr<T>( new T( config ) );
06071             }
06072             std::string getDescription() const override {
06073                 return std::string();
06074             }
06075         };
06076     public:
06077         ListenerRegistrar() {
06078             getMutableRegistryHub().registerListener( std::make_shared<ListenerFactory>() );
06079         }
06080     };
06081
06082 #if !defined(CATCH_CONFIG_DISABLE)
06083
06084 #define CATCH_REGISTER_REPORTER( name, reporterType ) \
06085     namespace{ Catch::ReporterRegistrar<reporterType> catch_internal_RegistrarFor##reporterType( name \
06086     ); } \
06087     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
06088
06089 #define CATCH_REGISTER_LISTENER( listenerType ) \
06090     namespace{ Catch::ListenerRegistrar<listenerType> catch_internal_RegistrarFor##listenerType; } \
06091     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
06092
06093 #else // CATCH_CONFIG_DISABLE
06094 #define CATCH_REGISTER_REPORTER( name, reporterType )
06095 #define CATCH_REGISTER_LISTENER( listenerType )
06096 #endif // CATCH_CONFIG_DISABLE
06097
06098 // end catch_reporter_registrars.hpp
06099 // Allow users to base their work off existing reporters
06100 // start catch_reporter_compact.h
06101
06102 namespace Catch {
06103
06104     struct CompactReporter : StreamingReporterBase<CompactReporter> {
06105     public:
06106         using StreamingReporterBase::StreamingReporterBase;
06107         ~CompactReporter() override;
06108         static std::string getDescription();
06109         void noMatchingTestCases( std::string const& spec ) override;
06110         void assertionStarting( AssertionInfo const& ) override;
06111         bool assertionEnded( AssertionStats const& _assertionStats ) override;
06112         void sectionEnded( SectionStats const& _sectionStats ) override;
06113         void testRunEnded( TestRunStats const& _testRunStats ) override;
06114     };
06115
06116 } // end namespace Catch
06117
06118 // end catch_reporter_compact.h
06119 // start catch_reporter_console.h
06120
06121 #if defined(_MSC_VER)
06122 #pragma warning(push)
06123 // Not all labels are EXPLICITLY handled in switch
06124 // Note that 4062 (not all labels are handled)
06125 // and default is missing) is enabled
06126 #endif
06127 #endif

```

```

06143 namespace Catch {
06144     // Fwd decls
06145     struct SummaryColumn;
06146     class TablePrinter;
06147
06148     struct ConsoleReporter : StreamingReporterBase<ConsoleReporter> {
06149         std::unique_ptr<TablePrinter> m_tablePrinter;
06150
06151         ConsoleReporter(ReporterConfig const& config);
06152         ~ConsoleReporter() override;
06153         static std::string getDescription();
06154
06155         void noMatchingTestCases(std::string const& spec) override;
06156
06157         void reportInvalidArguments(std::string const& arg) override;
06158
06159         void assertionStarting(AssertionInfo const&) override;
06160
06161         bool assertionEnded(AssertionStats const& _assertionStats) override;
06162
06163         void sectionStarting(SectionInfo const& _sectionInfo) override;
06164         void sectionEnded(SectionStats const& _sectionStats) override;
06165
06166 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
06167         void benchmarkPreparing(std::string const& name) override;
06168         void benchmarkStarting(BenchmarkInfo const& info) override;
06169         void benchmarkEnded(BenchmarkStats<> const& stats) override;
06170         void benchmarkFailed(std::string const& error) override;
06171 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
06172
06173         void testCaseEnded(TestCaseStats const& _testCaseStats) override;
06174         void testGroupEnded(TestGroupStats const& _testGroupStats) override;
06175         void testRunEnded(TestRunStats const& _testRunStats) override;
06176         void testRunStarting(TestRunInfo const& _testRunInfo) override;
06177     private:
06178
06179         void lazyPrint();
06180
06181         void lazyPrintWithoutClosingBenchmarkTable();
06182         void lazyPrintRunInfo();
06183         void lazyPrintGroupInfo();
06184         void printTestCaseAndSectionHeader();
06185
06186         void printClosedHeader(std::string const& _name);
06187         void printOpenHeader(std::string const& _name);
06188
06189         // if string has a : in first line will set indent to follow it on
06190         // subsequent lines
06191         void printHeaderString(std::string const& _string, std::size_t indent = 0);
06192
06193         void printTotals(Totals const& totals);
06194         void printSummaryRow(std::string const& label, std::vector<SummaryColumn> const& cols,
06195             std::size_t row);
06196
06197         void printTotalsDivider(Totals const& totals);
06198         void printSummaryDivider();
06199         void printTestFilters();
06200     private:
06201         bool m_headerPrinted = false;
06202     };
06203 } // end namespace Catch
06204
06205 #if defined(_MSC_VER)
06206 #pragma warning(pop)
06207 #endif
06208
06209 // end catch_reporter_console.h
06210 // start catch_reporter_junit.h
06211 // start catch_xmlwriter.h
06212
06213 #include <vector>
06214
06215 namespace Catch {
06216     enum class XmlFormatting {
06217         None = 0x00,
06218         Indent = 0x01,
06219         Newline = 0x02,
06220     };
06221
06222     XmlFormatting operator | (XmlFormatting lhs, XmlFormatting rhs);
06223     XmlFormatting operator & (XmlFormatting lhs, XmlFormatting rhs);
06224
06225     class XmlEncode {
06226     public:

```



```

06229     enum ForWhat { ForTextNodes, ForAttributes };
06230
06231     XmlEncode( std::string const& str, ForWhat forWhat = ForTextNodes );
06232
06233     void encodeTo( std::ostream& os ) const;
06234
06235     friend std::ostream& operator < ( std::ostream& os, XmlEncode const& xmlEncode );
06236
06237 private:
06238     std::string m_str;
06239     ForWhat m_forWhat;
06240 };
06241
06242 class XmlWriter {
06243 public:
06244
06245     class ScopedElement {
06246     public:
06247         ScopedElement( XmlWriter* writer, XmlFormatting fmt );
06248
06249         ScopedElement( ScopedElement&& other ) noexcept;
06250         ScopedElement& operator=( ScopedElement&& other ) noexcept;
06251
06252         ~ScopedElement();
06253
06254         ScopedElement& writeText( std::string const& text, XmlFormatting fmt =
06255             XmlFormatting::Newline | XmlFormatting::Indent );
06256
06257         template<typename T>
06258         ScopedElement& writeAttribute( std::string const& name, T const& attribute ) {
06259             m_writer->writeAttribute( name, attribute );
06260             return *this;
06261         }
06262
06263     private:
06264         mutable XmlWriter* m_writer = nullptr;
06265         XmlFormatting m_fmt;
06266     };
06267
06268     XmlWriter( std::ostream& os = Catch::cout() );
06269     ~XmlWriter();
06270
06271     XmlWriter( XmlWriter const& ) = delete;
06272     XmlWriter& operator=( XmlWriter const& ) = delete;
06273
06274     XmlWriter& startElement( std::string const& name, XmlFormatting fmt = XmlFormatting::Newline |
06275         XmlFormatting::Indent);
06276
06277     ScopedElement scopedElement( std::string const& name, XmlFormatting fmt =
06278         XmlFormatting::Newline | XmlFormatting::Indent);
06279
06280     XmlWriter& endElement(XmlFormatting fmt = XmlFormatting::Newline | XmlFormatting::Indent);
06281
06282     XmlWriter& writeAttribute( std::string const& name, std::string const& attribute );
06283
06284     XmlWriter& writeAttribute( std::string const& name, bool attribute );
06285
06286     template<typename T>
06287     XmlWriter& writeAttribute( std::string const& name, T const& attribute ) {
06288         ReusableStringStream rss;
06289         rss << attribute;
06290         return writeAttribute( name, rss.str() );
06291     }
06292
06293     XmlWriter& writeText( std::string const& text, XmlFormatting fmt = XmlFormatting::Newline |
06294         XmlFormatting::Indent);
06295
06296     XmlWriter& writeComment(std::string const& text, XmlFormatting fmt = XmlFormatting::Newline |
06297         XmlFormatting::Indent);
06298
06299     void writeStylesheetRef( std::string const& url );
06300
06301     XmlWriter& writeBlankLine();
06302
06303     void ensureTagClosed();
06304
06305 private:
06306     void applyFormatting(XmlFormatting fmt);
06307
06308     void writeDeclaration();
06309
06310     void newlineIfNecessary();
06311
06312     bool m_tagIsOpen = false;
06313     bool m_needsNewline = false;
06314     std::vector<std::string> m_tags;

```

```

06311         std::string m_indent;
06312         std::ostream& m_os;
06313     };
06314 }
06315 }
06316
06317 // end catch_xmlwriter.h
06318 namespace Catch {
06319
06320     class JunitReporter : public CumulativeReporterBase<JunitReporter> {
06321     public:
06322         JunitReporter(ReporterConfig const& _config);
06323
06324         ~JunitReporter() override;
06325
06326         static std::string getDescription();
06327
06328         void noMatchingTestCases(std::string const& /*spec*/) override;
06329
06330         void testRunStarting(TestRunInfo const& runInfo) override;
06331
06332         void testGroupStarting(GroupInfo const& groupInfo) override;
06333
06334         void testCaseStarting(TestCaseInfo const& testCaseInfo) override;
06335         bool assertionEnded(AssertionStats const& assertionStats) override;
06336
06337         void testCaseEnded(TestCaseStats const& testCaseStats) override;
06338
06339         void testGroupEnded(TestGroupStats const& testGroupStats) override;
06340
06341         void testRunEndedCumulative() override;
06342
06343         void writeGroup(TestGroupNode const& groupNode, double suiteTime);
06344
06345         void writeTestCase(TestCaseNode const& testCaseNode);
06346
06347         void writeSection( std::string const& className,
06348                           std::string const& rootName,
06349                           SectionNode const& sectionNode,
06350                           bool testOkToFail );
06351
06352         void writeAssertions(SectionNode const& sectionNode);
06353         void writeAssertion(AssertionStats const& stats);
06354
06355         XmlWriter xml;
06356         Timer suiteTimer;
06357         std::string stdOutForSuite;
06358         std::string stdErrForSuite;
06359         unsigned int unexpectedExceptions = 0;
06360         bool m_okToFail = false;
06361     };
06362 }
06363 // end namespace Catch
06364
06365 // end catch_reporter_junit.h
06366 // start catch_reporter_xml.h
06367
06368 namespace Catch {
06369     class XmlReporter : public StreamingReporterBase<XmlReporter> {
06370     public:
06371         XmlReporter(ReporterConfig const& _config);
06372
06373         ~XmlReporter() override;
06374
06375         static std::string getDescription();
06376
06377         virtual std::string getStylesheetRef() const;
06378
06379         void writeSourceInfo(SourceLineInfo const& sourceInfo);
06380
06381     public: // StreamingReporterBase
06382
06383         void noMatchingTestCases(std::string const& s) override;
06384
06385         void testRunStarting(TestRunInfo const& testInfo) override;
06386
06387         void testGroupStarting(GroupInfo const& groupInfo) override;
06388
06389         void testCaseStarting(TestCaseInfo const& testInfo) override;
06390
06391         void sectionStarting(SectionInfo const& sectionInfo) override;
06392
06393         void assertionStarting(AssertionInfo const&) override;
06394
06395         bool assertionEnded(AssertionStats const& assertionStats) override;
06396
06397         void sectionEnded(SectionStats const& sectionStats) override;

```

```

06398
06399     void testCaseEnded(TestCaseStats const& testCaseStats) override;
06400
06401     void testGroupEnded(TestGroupStats const& testGroupStats) override;
06402
06403     void testRunEnded(TestRunStats const& testRunStats) override;
06404
06405 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
06406     void benchmarkPreparing(std::string const& name) override;
06407     void benchmarkStarting(BenchmarkInfo const&) override;
06408     void benchmarkEnded(BenchmarkStats<> const&) override;
06409     void benchmarkFailed(std::string const&) override;
06410 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
06411
06412     private:
06413         Timer m_testCaseTimer;
06414         XmlWriter m_xml;
06415         int m_sectionDepth = 0;
06416     };
06417
06418 } // end namespace Catch
06419
06420 // end catch_reporter_xml.h
06421
06422 // end catch_external_interfaces.h
06423 #endif
06424
06425 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
06426 // start catch_benchmarking_all.hpp
06427
06428 // A proxy header that includes all of the benchmarking headers to allow
06429 // concise include of the benchmarking features. You should prefer the
06430 // individual includes in standard use.
06431
06432 // start catch_benchmark.hpp
06433
06434 // Benchmark
06435
06436 // start catch_chronometer.hpp
06437
06438 // User-facing chronometer
06439
06440
06441 // start catch_clock.hpp
06442
06443 // Clocks
06444
06445
06446 #include <chrono>
06447 #include <ratio>
06448
06449 namespace Catch {
06450     namespace Benchmark {
06451         template <typename Clock>
06452         using ClockDuration = typename Clock::duration;
06453         template <typename Clock>
06454         using FloatDuration = std::chrono::duration<double, typename Clock::period>;
06455
06456         template <typename Clock>
06457         using TimePoint = typename Clock::time_point;
06458
06459         using default_clock = std::chrono::steady_clock;
06460
06461         template <typename Clock>
06462         struct now {
06463             TimePoint<Clock> operator()() const {
06464                 return Clock::now();
06465             }
06466         };
06467
06468         using fp_seconds = std::chrono::duration<double, std::ratio<1>;
06469     } // namespace Benchmark
06470 } // namespace Catch
06471
06472 // end catch_clock.hpp
06473 // start catch_optimizer.hpp
06474
06475 // Hinting the optimizer
06476
06477
06478 #if defined(_MSC_VER)
06479 #    include <atomic> // atomic_thread_fence
06480 #endif
06481
06482 namespace Catch {
06483     namespace Benchmark {
06484 #if defined(__GNUC__) || defined(__clang__)

```

```

06485     template <typename T>
06486     inline void keep_memory(T* p) {
06487         asm volatile("" : : "g"(p) : "memory");
06488     }
06489     inline void keep_memory() {
06490         asm volatile("" : : : "memory");
06491     }
06492
06493     namespace Detail {
06494         inline void optimizer_barrier() { keep_memory(); }
06495     } // namespace Detail
06496 #elif defined(_MSC_VER)
06497
06498 #pragma optimize("", off)
06499     template <typename T>
06500     inline void keep_memory(T* p) {
06501         // thanks @milleniumbug
06502         *reinterpret_cast<char volatile*>(p) = *reinterpret_cast<char const volatile*>(p);
06503     }
06504     // TODO equivalent keep_memory()
06505 #pragma optimize("", on)
06506
06507     namespace Detail {
06508         inline void optimizer_barrier() {
06509             std::atomic_thread_fence(std::memory_order_seq_cst);
06510         }
06511     } // namespace Detail
06512
06513 #endif
06514
06515     template <typename T>
06516     inline void deoptimize_value(T&& x) {
06517         keep_memory(&x);
06518     }
06519
06520     template <typename Fn, typename... Args>
06521     inline auto invoke_deoptimized(Fn&& fn, Args&&... args) -> typename
std::enable_if<!std::is_same<void, decltype(fn(args...))>::value>::type {
06522         deoptimize_value(std::forward<Fn>(fn) (std::forward<Args...>(args...)));
06523     }
06524
06525     template <typename Fn, typename... Args>
06526     inline auto invoke_deoptimized(Fn&& fn, Args&&... args) -> typename
std::enable_if<std::is_same<void, decltype(fn(args...))>::value>::type {
06527         std::forward<Fn>(fn) (std::forward<Args...>(args...));
06528     }
06529 } // namespace Benchmark
06530 } // namespace Catch
06531
06532 // end catch_optimizer.hpp
06533 // start catch_complete_invoke.hpp
06534
06535 // Invoke with a special case for void
06536
06537 #include <type_traits>
06538 #include <utility>
06539
06540 namespace Catch {
06541     namespace Benchmark {
06542         namespace Detail {
06543             template <typename T>
06544             struct CompleteType { using type = T; };
06545             template <>
06546             struct CompleteType<void> { struct type {}; };
06547
06548             template <typename T>
06549             using CompleteType_t = typename CompleteType<T>::type;
06550
06551             template <typename Result>
06552             struct CompleteInvoker {
06553                 template <typename Fun, typename... Args>
06554                 static Result invoke(Fun&& fun, Args&&... args) {
06555                     return std::forward<Fun>(fun) (std::forward<Args>(args)...);
06556                 }
06557             };
06558
06559             template <>
06560             struct CompleteInvoker<void> {
06561                 template <typename Fun, typename... Args>
06562                 static CompleteType_t<void> invoke(Fun&& fun, Args&&... args) {
06563                     std::forward<Fun>(fun) (std::forward<Args>(args)...);
06564                     return {};
06565                 }
06566             };
06567
06568             // invoke and not return void :(
06569             template <typename Fun, typename... Args>

```

```

06570         CompleteType_t<FunctionReturnType<Fun, Args...> complete_invoke(Fun&& fun, Args&&... args)
06571     {
06572         return CompleteInvoker<FunctionReturnType<Fun,
06573     Args...>::invoke(std::forward<Fun>(fun), std::forward<Args>(args)...);
06574     }
06575     const std::string benchmarkErrorMsg = "a benchmark failed to run successfully";
06576     } // namespace Detail
06577     template <typename Fun>
06578     Detail::CompleteType_t<FunctionReturnType<Fun> user_code(Fun&& fun) {
06579         CATCH_TRY{
06580             return Detail::complete_invoke(std::forward<Fun>(fun));
06581         } CATCH_CATCH_ALL{
06582             getResultCapture().benchmarkFailed(translateActiveException());
06583             CATCH_RUNTIME_ERROR(Detail::benchmarkErrorMsg);
06584         }
06585     }
06586     } // namespace Benchmark
06587 } // namespace Catch
06588
06589 // end catch_complete_invoke.hpp
06590 namespace Catch {
06591     namespace Benchmark {
06592         namespace Detail {
06593             struct ChronometerConcept {
06594                 virtual void start() = 0;
06595                 virtual void finish() = 0;
06596                 virtual ~ChronometerConcept() = default;
06597             };
06598             template <typename Clock>
06599             struct ChronometerModel final : public ChronometerConcept {
06600                 void start() override { started = Clock::now(); }
06601                 void finish() override { finished = Clock::now(); }
06602
06603                 ClockDuration<Clock> elapsed() const { return finished - started; }
06604
06605                 TimePoint<Clock> started;
06606                 TimePoint<Clock> finished;
06607             };
06608         } // namespace Detail
06609
06610         struct Chronometer {
06611         public:
06612             template <typename Fun>
06613             void measure(Fun&& fun) { measure(std::forward<Fun>(fun), is_callable<Fun(int)>()); }
06614
06615             int runs() const { return k; }
06616
06617             Chronometer(Detail::ChronometerConcept& meter, int k)
06618                 : impl(&meter)
06619                 , k(k) {}
06620
06621         private:
06622             template <typename Fun>
06623             void measure(Fun&& fun, std::false_type) {
06624                 measure([&fun](int) { return fun(); }, std::true_type());
06625             }
06626
06627             template <typename Fun>
06628             void measure(Fun&& fun, std::true_type) {
06629                 Detail::optimizer_barrier();
06630                 impl->start();
06631                 for (int i = 0; i < k; ++i) invoke_deoptimized(fun, i);
06632                 impl->finish();
06633                 Detail::optimizer_barrier();
06634             }
06635
06636             Detail::ChronometerConcept* impl;
06637             int k;
06638         };
06639     } // namespace Benchmark
06640 } // namespace Catch
06641
06642 // end catch_chronometer.hpp
06643 // start catch_environment.hpp
06644
06645 // Environment information
06646
06647 namespace Catch {
06648     namespace Benchmark {
06649         template <typename Duration>
06650         struct EnvironmentEstimate {
06651             Duration mean;
06652             OutlierClassification outliers;
06653         };
06654     }

```

```

06655         template <typename Duration2>
06656         operator EnvironmentEstimate<Duration2>() const {
06657             return { mean, outliers };
06658         }
06659     };
06660     template <typename Clock>
06661     struct Environment {
06662         using clock_type = Clock;
06663         EnvironmentEstimate<FloatDuration<Clock>> clock_resolution;
06664         EnvironmentEstimate<FloatDuration<Clock>> clock_cost;
06665     };
06666     } // namespace Benchmark
06667 } // namespace Catch
06668
06669 // end catch_environment.hpp
06670 // start catch_execution_plan.hpp
06671
06672 // Execution plan
06673
06674
06675 // start catch_benchmark_function.hpp
06676
06677 // Dumb std::function implementation for consistent call overhead
06678
06679
06680 #include <cassert>
06681 #include <type_traits>
06682 #include <utility>
06683 #include <memory>
06684
06685 namespace Catch {
06686     namespace Benchmark {
06687         namespace Detail {
06688             template <typename T>
06689             using Decay = typename std::decay<T>::type;
06690             template <typename T, typename U>
06691             struct is_related
06692                 : std::is_same<Decay<T>, Decay<U>> {};
06693
06694             struct BenchmarkFunction {
06695 private:
06696                 struct callable {
06697                     virtual void call(Chronometer meter) const = 0;
06698                     virtual callable* clone() const = 0;
06699                     virtual ~callable() = default;
06700                 };
06701                 template <typename Fun>
06702                 struct model : public callable {
06703                     model(Fun&& fun) : fun(std::move(fun)) {}
06704                     model(Fun const& fun) : fun(fun) {}
06705
06706                     model<Fun>* clone() const override { return new model<Fun>(*this); }
06707
06708                     void call(Chronometer meter) const override {
06709                         call(meter, is_callable<Fun(Chronometer)>());
06710                     }
06711                     void call(Chronometer meter, std::true_type) const {
06712                         fun(meter);
06713                     }
06714                     void call(Chronometer meter, std::false_type) const {
06715                         meter.measure(fun);
06716                     }
06717
06718                     Fun fun;
06719                 };
06720
06721                 struct do_nothing { void operator()() const {} };
06722
06723                 template <typename T>
06724                 BenchmarkFunction(model<T>* c) : f(c) {}
06725
06726 public:
06727                 BenchmarkFunction()
06728                     : f(new model<do_nothing>{ {} }) {}
06729
06730                 template <typename Fun,
06731                     typename std::enable_if<!is_related<Fun, BenchmarkFunction>::value, int>::type =
06732
06733                     0>
06734                     BenchmarkFunction(Fun&& fun)
06735                         : f(new model<typename std::decay<Fun>::type>(std::forward<Fun>(fun))) {}
06736
06737                 BenchmarkFunction(BenchmarkFunction&& that)
06738                     : f(std::move(that.f)) {}
06739
06740                 BenchmarkFunction(BenchmarkFunction const& that)
06741                     : f(that.f->clone()) {}
06742
06743
06744
06745
06746
06747

```

```

06748         BenchmarkFunction& operator=(BenchmarkFunction&& that) {
06749             f = std::move(that.f);
06750             return *this;
06751         }
06752
06753         BenchmarkFunction& operator=(BenchmarkFunction const& that) {
06754             f.reset(that.f->clone());
06755             return *this;
06756         }
06757
06758         void operator()(Chronometer meter) const { f->call(meter); }
06759
06760     private:
06761         std::unique_ptr<callable> f;
06762     };
06763 } // namespace Detail
06764 } // namespace Benchmark
06765 } // namespace Catch
06766
06767 // end catch_benchmark_function.hpp
06768 // start catch_repeat.hpp
06769
06770 // repeat algorithm
06771
06772 #include <type_traits>
06773 #include <utility>
06774
06775 namespace Catch {
06776     namespace Benchmark {
06777         namespace Detail {
06778             template <typename Fun>
06779             struct repeater {
06780                 void operator()(int k) const {
06781                     for (int i = 0; i < k; ++i) {
06782                         fun();
06783                     }
06784                 }
06785             };
06786             Fun fun;
06787         };
06788         template <typename Fun>
06789         repeater<typename std::decay<Fun>::type> repeat(Fun&& fun) {
06790             return { std::forward<Fun>(fun) };
06791         }
06792     } // namespace Detail
06793 } // namespace Benchmark
06794 } // namespace Catch
06795
06796 // end catch_repeat.hpp
06797 // start catch_run_for_at_least.hpp
06798
06799 // Run a function for a minimum amount of time
06800
06801 // start catch_measure.hpp
06802
06803 // Measure
06804
06805 // start catch_timing.hpp
06806
06807 // Timing
06808
06809 #include <tuple>
06810 #include <type_traits>
06811
06812 namespace Catch {
06813     namespace Benchmark {
06814         template <typename Duration, typename Result>
06815         struct Timing {
06816             Duration elapsed;
06817             Result result;
06818             int iterations;
06819         };
06820         template <typename Clock, typename Func, typename... Args>
06821         using TimingOf = Timing<ClockDuration<Clock>, Detail::CompleteType_t<FunctionReturnType<Func,
06822             Args...>>>;
06823     } // namespace Benchmark
06824 } // namespace Catch
06825
06826 // end catch_timing.hpp
06827 #include <utility>
06828
06829 namespace Catch {
06830     namespace Benchmark {
06831         namespace Detail {

```

```

06834         template <typename Clock, typename Fun, typename... Args>
06835         TimingOf<Clock, Fun, Args...> measure(Fun&& fun, Args&&... args) {
06836             auto start = Clock::now();
06837             auto&& r = Detail::complete_invoke(fun, std::forward<Args>(args)...);
06838             auto end = Clock::now();
06839             auto delta = end - start;
06840             return { delta, std::forward<decltype(r)>(r), 1 };
06841         }
06842     } // namespace Detail
06843 } // namespace Benchmark
06844 } // namespace Catch
06845
06846 // end catch_measure.hpp
06847 #include <utility>
06848 #include <type_traits>
06849
06850 namespace Catch {
06851     namespace Benchmark {
06852         namespace Detail {
06853             template <typename Clock, typename Fun>
06854             TimingOf<Clock, Fun, int> measure_one(Fun&& fun, int iters, std::false_type) {
06855                 return Detail::measure<Clock>(fun, iters);
06856             }
06857             template <typename Clock, typename Fun>
06858             TimingOf<Clock, Fun, Chronometer> measure_one(Fun&& fun, int iters, std::true_type) {
06859                 Detail::ChronometerModel<Clock> meter;
06860                 auto&& result = Detail::complete_invoke(fun, Chronometer(meter, iters));
06861
06862                 return { meter.elapsed(), std::move(result), iters };
06863             }
06864
06865             template <typename Clock, typename Fun>
06866             using run_for_at_least_argument_t = typename
06867             std::conditional<is_callable<Fun(Chronometer)>::value, Chronometer, int>::type;
06868
06869             struct optimized_away_error : std::exception {
06870                 const char* what() const noexcept override {
06871                     return "could not measure benchmark, maybe it was optimized away";
06872                 }
06873             };
06874
06875             template <typename Clock, typename Fun>
06876             TimingOf<Clock, Fun, run_for_at_least_argument_t<Clock, Fun>
06877             run_for_at_least(ClockDuration<Clock> how_long, int seed, Fun&& fun) {
06878                 auto iters = seed;
06879                 while (iters < (1 << 30)) {
06880                     auto&& Timing = measure_one<Clock>(fun, iters, is_callable<Fun(Chronometer)>());
06881                     if (Timing.elapsed >= how_long) {
06882                         return { Timing.elapsed, std::move(Timing.result), iters };
06883                     }
06884                     iters *= 2;
06885                 }
06886                 Catch::throw_exception(optimized_away_error{});
06887             } // namespace Detail
06888         } // namespace Benchmark
06889     } // namespace Catch
06890
06891 // end catch_run_for_at_least.hpp
06892 #include <algorithm>
06893 #include <iterator>
06894
06895 namespace Catch {
06896     namespace Benchmark {
06897         template <typename Duration>
06898         struct ExecutionPlan {
06899             int iterations_per_sample;
06900             Duration estimated_duration;
06901             Detail::BenchmarkFunction benchmark;
06902             Duration warmup_time;
06903             int warmup_iterations;
06904
06905             template <typename Duration2>
06906             operator ExecutionPlan<Duration2>() const {
06907                 return { iterations_per_sample, estimated_duration, benchmark, warmup_time,
06908                     warmup_iterations };
06909             }
06910
06911             template <typename Clock>
06912             std::vector<FloatDuration<Clock>> run(const IConfig &cfg, Environment<FloatDuration<Clock>>
06913             env) const {
06914                 // warmup a bit
06915                 Detail::run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(warmup_time),
06916                     warmup_iterations, Detail::repeat(now<Clock>{}));
06917             }
06918         }
06919     }
06920 }

```



```

06915         std::vector<FloatDuration<Clock>> times;
06916         times.reserve(cfg.benchmarkSamples());
06917         std::generate_n(std::back_inserter(times), cfg.benchmarkSamples(), [this, env] {
06918             Detail::ChronometerModel<Clock> model;
06919             this->benchmark(Chronometer(model, iterations_per_sample));
06920             auto sample_time = model.elapsed() - env.clock_cost.mean;
06921             if (sample_time < FloatDuration<Clock>::zero()) sample_time =
FloatDuration<Clock>::zero();
06922             return sample_time / iterations_per_sample;
06923         });
06924         return times;
06925     }
06926 };
06927 } // namespace Benchmark
06928 } // namespace Catch
06929
06930 // end catch_execution_plan.hpp
06931 // start catch_estimate_clock.hpp
06932
06933 // Environment measurement
06934
06935
06936 // start catch_stats.hpp
06937
06938 // Statistical analysis tools
06939
06940
06941 #include <algorithm>
06942 #include <functional>
06943 #include <vector>
06944 #include <iterator>
06945 #include <numeric>
06946 #include <tuple>
06947 #include <cmath>
06948 #include <utility>
06949 #include <cstdint>
06950 #include <random>
06951
06952 namespace Catch {
06953     namespace Benchmark {
06954         namespace Detail {
06955             using sample = std::vector<double>;
06956
06957             double weighted_average_quantile(int k, int q, std::vector<double>::iterator first,
std::vector<double>::iterator last);
06958
06959             template <typename Iterator>
06960             OutlierClassification classify_outliers(Iterator first, Iterator last) {
06961                 std::vector<double> copy(first, last);
06962
06963                 auto q1 = weighted_average_quantile(1, 4, copy.begin(), copy.end());
06964                 auto q3 = weighted_average_quantile(3, 4, copy.begin(), copy.end());
06965                 auto iqr = q3 - q1;
06966                 auto los = q1 - (iqr * 3.);
06967                 auto lom = q1 - (iqr * 1.5);
06968                 auto him = q3 + (iqr * 1.5);
06969                 auto his = q3 + (iqr * 3.);
06970
06971                 OutlierClassification o;
06972                 for (; first != last; ++first) {
06973                     auto&& t = *first;
06974                     if (t < los) ++o.low_severe;
06975                     else if (t < lom) ++o.low_mild;
06976                     else if (t > his) ++o.high_severe;
06977                     else if (t > him) ++o.high_mild;
06978                     ++o.samples_seen;
06979                 }
06980                 return o;
06981             }
06982
06983             template <typename Iterator>
06984             double mean(Iterator first, Iterator last) {
06985                 auto count = last - first;
06986                 double sum = std::accumulate(first, last, 0.);
06987                 return sum / count;
06988             }
06989
06990             template <typename URng, typename Iterator, typename Estimator>
06991             sample resample(URng& rng, int resamples, Iterator first, Iterator last, Estimator&
estimator) {
06992                 auto n = last - first;
06993                 std::uniform_int_distribution<decltype(n)> dist(0, n - 1);
06994
06995                 sample out;
06996                 out.reserve(resamples);
06997                 std::generate_n(std::back_inserter(out), resamples, [n, first, &estimator, &dist,
&rng] {

```

```

06998         std::vector<double> resampled;
06999         resampled.reserve(n);
07000         std::generate_n(std::back_inserter(resampled), n, [first, &dist, &rng] { return
first[dist(rng)]; });
07001         return estimator(resampled.begin(), resampled.end());
07002     });
07003     std::sort(out.begin(), out.end());
07004     return out;
07005 }
07006
07007 template <typename Estimator, typename Iterator>
07008 sample jackknife(Estimator&& estimator, Iterator first, Iterator last) {
07009     auto n = last - first;
07010     auto second = std::next(first);
07011     sample results;
07012     results.reserve(n);
07013
07014     for (auto it = first; it != last; ++it) {
07015         std::iter_swap(it, first);
07016         results.push_back(estimator(second, last));
07017     }
07018
07019     return results;
07020 }
07021
07022 inline double normal_cdf(double x) {
07023     return std::erfc(-x / std::sqrt(2.0)) / 2.0;
07024 }
07025
07026 double erfc_inv(double x);
07027
07028 double normal_quantile(double p);
07029
07030 template <typename Iterator, typename Estimator>
07031 Estimate<double> bootstrap(double confidence_level, Iterator first, Iterator last, sample
const& resample, Estimator&& estimator) {
07032     auto n_samples = last - first;
07033
07034     double point = estimator(first, last);
07035     // Degenerate case with a single sample
07036     if (n_samples == 1) return { point, point, point, confidence_level };
07037
07038     sample jack = jackknife(estimator, first, last);
07039     double jack_mean = mean(jack.begin(), jack.end());
07040     double sum_squares, sum_cubes;
07041     std::tie(sum_squares, sum_cubes) = std::accumulate(jack.begin(), jack.end(),
std::make_pair(0., 0.), [jack_mean](std::pair<double, double> sqcb, double x) -> std::pair<double,
double> {
07042         auto d = jack_mean - x;
07043         auto d2 = d * d;
07044         auto d3 = d2 * d;
07045         return { sqcb.first + d2, sqcb.second + d3 };
07046     });
07047
07048     double accel = sum_cubes / (6 * std::pow(sum_squares, 1.5));
07049     int n = static_cast<int>(resample.size());
07050     double prob_n = std::count_if(resample.begin(), resample.end(), [point](double x) {
return x < point; }) / (double)n;
07051     // degenerate case with uniform samples
07052     if (prob_n == 0) return { point, point, point, confidence_level };
07053
07054     double bias = normal_quantile(prob_n);
07055     double z1 = normal_quantile((1. - confidence_level) / 2.);
07056
07057     auto cumn = [n](double x) -> int {
07058         return std::lround(normal_cdf(x) * n); };
07059     auto a = [bias, accel](double b) { return bias + b / (1. - accel * b); };
07060     double b1 = bias + z1;
07061     double b2 = bias - z1;
07062     double a1 = a(b1);
07063     double a2 = a(b2);
07064     auto lo = (std::max)(cumn(a1), 0);
07065     auto hi = (std::min)(cumn(a2), n - 1);
07066
07067     return { point, resample[lo], resample[hi], confidence_level };
07068 }
07069
07070 double outlier_variance(Estimate<double> mean, Estimate<double> stddev, int n);
07071
07072 struct bootstrap_analysis {
07073     Estimate<double> mean;
07074     Estimate<double> standard_deviation;
07075     double outlier_variance;
07076 };
07077
07078 bootstrap_analysis analyse_samples(double confidence_level, int n_resamples,
std::vector<double>::iterator first, std::vector<double>::iterator last);

```

```

07079         } // namespace Detail
07080     } // namespace Benchmark
07081 } // namespace Catch
07082
07083 // end catch_stats.hpp
07084 #include <algorithm>
07085 #include <iterator>
07086 #include <tuple>
07087 #include <vector>
07088 #include <cmath>
07089
07090 namespace Catch {
07091     namespace Benchmark {
07092         namespace Detail {
07093             template <typename Clock>
07094             std::vector<double> resolution(int k) {
07095                 std::vector<TimePoint<Clock>> times;
07096                 times.reserve(k + 1);
07097                 std::generate_n(std::back_inserter(times), k + 1, now<Clock>{});
07098
07099                 std::vector<double> deltas;
07100                 deltas.reserve(k);
07101                 std::transform(std::next(times.begin()), times.end(), times.begin(),
07102                     std::back_inserter(deltas),
07103                     [](TimePoint<Clock> a, TimePoint<Clock> b) { return static_cast<double>((a -
07104 b).count()); });
07105
07106                 return deltas;
07107             }
07108
07109             const auto warmup_iterations = 10000;
07110             const auto warmup_time = std::chrono::milliseconds(100);
07111             const auto minimum_ticks = 1000;
07112             const auto warmup_seed = 10000;
07113             const auto clock_resolution_estimation_time = std::chrono::milliseconds(500);
07114             const auto clock_cost_estimation_time_limit = std::chrono::seconds(1);
07115             const auto clock_cost_estimation_tick_limit = 100000;
07116             const auto clock_cost_estimation_time = std::chrono::milliseconds(10);
07117             const auto clock_cost_estimation_iterations = 10000;
07118
07119             template <typename Clock>
07120             int warmup() {
07121                 return
07122                     run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(warmup_time), warmup_seed,
07123                     &resolution<Clock>)
07124                         .iterations;
07125             }
07126
07127             template <typename Clock>
07128             EnvironmentEstimate<FloatDuration<Clock>> estimate_clock_resolution(int iterations) {
07129                 auto r =
07130                     run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(clock_resolution_estimation_time),
07131                     iterations, &resolution<Clock>)
07132                         .result;
07133                 return {
07134                     r,
07135                     FloatDuration<Clock>(mean(r.begin(), r.end())),
07136                     classify_outliers(r.begin(), r.end())
07137                 };
07138             }
07139
07140             template <typename Clock>
07141             EnvironmentEstimate<FloatDuration<Clock>> estimate_clock_cost(FloatDuration<Clock>
07142 resolution) {
07143                 auto time_limit = (std::min)(
07144                     resolution * clock_cost_estimation_tick_limit,
07145                     FloatDuration<Clock>(clock_cost_estimation_time_limit));
07146                 auto time_clock = [](int k) {
07147                     return Detail::measure<Clock>([k] {
07148                         for (int i = 0; i < k; ++i) {
07149                             volatile auto ignored = Clock::now();
07150                             (void)ignored;
07151                         }
07152                     }).elapsed;
07153                 };
07154                 time_clock(1);
07155                 int iters = clock_cost_estimation_iterations;
07156                 auto&& r =
07157                     run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(clock_cost_estimation_time),
07158                     iters, time_clock);
07159                 std::vector<double> times;
07160                 int nsamples = static_cast<int>(std::ceil(time_limit / r.elapsed));
07161                 times.reserve(nsamples);
07162                 std::generate_n(std::back_inserter(times), nsamples, [time_clock, &r] {
07163                     return static_cast<double>((time_clock(r.iterations) / r.iterations).count());
07164                 });
07165                 return {
07166                     FloatDuration<Clock>(mean(times.begin(), times.end())),
07167                     classify_outliers(times.begin(), times.end())
07168                 };
07169             }
07170
07171         }
07172     }
07173 }

```

```

07158         }
07159
07160         template <typename Clock>
07161         Environment<FloatDuration<Clock>> measure_environment() {
07162             static Environment<FloatDuration<Clock>>* env = nullptr;
07163             if (env) {
07164                 return *env;
07165             }
07166
07167             auto iters = Detail::warmup<Clock>();
07168             auto resolution = Detail::estimate_clock_resolution<Clock>(iters);
07169             auto cost = Detail::estimate_clock_cost<Clock>(resolution.mean);
07170
07171             env = new Environment<FloatDuration<Clock>>(resolution, cost);
07172             return *env;
07173         }
07174     } // namespace Detail
07175 } // namespace Benchmark
07176 } // namespace Catch
07177
07178 // end catch_estimate_clock.hpp
07179 // start catch_analyse.hpp
07180
07181 // Run and analyse one benchmark
07182
07183
07184 // start catch_sample_analysis.hpp
07185
07186 // Benchmark results
07187
07188
07189 #include <algorithm>
07190 #include <vector>
07191 #include <string>
07192 #include <iterator>
07193
07194 namespace Catch {
07195     namespace Benchmark {
07196         template <typename Duration>
07197         struct SampleAnalysis {
07198             std::vector<Duration> samples;
07199             Estimate<Duration> mean;
07200             Estimate<Duration> standard_deviation;
07201             OutlierClassification outliers;
07202             double outlier_variance;
07203
07204             template <typename Duration2>
07205             operator SampleAnalysis<Duration2>() const {
07206                 std::vector<Duration2> samples2;
07207                 samples2.reserve(samples.size());
07208                 std::transform(samples.begin(), samples.end(), std::back_inserter(samples2),
07209                     [](Duration d) { return Duration2(d); });
07210                 return {
07211                     std::move(samples2),
07212                     mean,
07213                     standard_deviation,
07214                     outliers,
07215                     outlier_variance,
07216                 };
07217             }
07218         } // namespace Benchmark
07219     } // namespace Catch
07220
07221 // end catch_sample_analysis.hpp
07222 #include <algorithm>
07223 #include <iterator>
07224 #include <vector>
07225
07226 namespace Catch {
07227     namespace Benchmark {
07228         namespace Detail {
07229             template <typename Duration, typename Iterator>
07230             SampleAnalysis<Duration> analyse(const IConfig &cfg, Environment<Duration>, Iterator
07231 first, Iterator last) {
07232                 if (!cfg.benchmarkNoAnalysis()) {
07233                     std::vector<double> samples;
07234                     samples.reserve(last - first);
07235                     std::transform(first, last, std::back_inserter(samples), [](Duration d) { return
07236 d.count(); });
07237
07238                     auto analysis =
07239                         Catch::Benchmark::Detail::analyse_samples(cfg.benchmarkConfidenceInterval(), cfg.benchmarkResamples(),
07240                             samples.begin(), samples.end());
07241                     auto outliers = Catch::Benchmark::Detail::classify_outliers(samples.begin(),
07242                         samples.end());
07243                 }
07244             }
07245         }
07246     }
07247 }

```

```

07239         auto wrap_estimate = [] (Estimate<double> e) {
07240             return Estimate<Duration> {
07241                 Duration(e.point),
07242                 Duration(e.lower_bound),
07243                 Duration(e.upper_bound),
07244                 e.confidence_interval,
07245             };
07246         };
07247         std::vector<Duration> samples2;
07248         samples2.reserve(samples.size());
07249         std::transform(samples.begin(), samples.end(), std::back_inserter(samples2),
07250             [](double d) { return Duration(d); });
07251         return {
07252             std::move(samples2),
07253             wrap_estimate(analysis.mean),
07254             wrap_estimate(analysis.standard_deviation),
07255             outliers,
07256             analysis.outlier_variance,
07257         };
07258     } else {
07259         std::vector<Duration> samples;
07260         samples.reserve(last - first);
07261
07262         Duration mean = Duration(0);
07263         int i = 0;
07264         for (auto it = first; it < last; ++it, ++i) {
07265             samples.push_back(Duration(*it));
07266             mean += Duration(*it);
07267         }
07268         mean /= i;
07269
07270         return {
07271             std::move(samples),
07272             Estimate<Duration>{mean, mean, mean, 0.0},
07273             Estimate<Duration>{Duration(0), Duration(0), Duration(0), 0.0},
07274             OutlierClassification{},
07275             0.0
07276         };
07277     }
07278 } // namespace Detail
07279 } // namespace Benchmark
07280 } // namespace Catch
07281
07282 // end catch_analyse.hpp
07283 #include <algorithm>
07284 #include <functional>
07285 #include <string>
07286 #include <vector>
07287 #include <cmath>
07288
07289 namespace Catch {
07290     namespace Benchmark {
07291         struct Benchmark {
07292             Benchmark(std::string &&name)
07293                 : name(std::move(name)) {}
07294
07295         template <class FUN>
07296             Benchmark(std::string &&name, FUN &&func)
07297                 : fun(std::move(func)), name(std::move(name)) {}
07298
07299         template <typename Clock>
07300             ExecutionPlan<FloatDuration<Clock>> prepare(const IConfig &cfg,
07301                 Environment<FloatDuration<Clock>> env) const {
07302             auto min_time = env.clock_resolution.mean * Detail::minimum_ticks;
07303             auto run_time = std::max(min_time,
07304                 std::chrono::duration_cast<decltype(min_time)>(cfg.benchmarkWarmupTime()));
07305             auto&& test =
07306                 Detail::run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(run_time), 1, fun);
07307             int new_iters = static_cast<int>(std::ceil(min_time * test.iterations /
07308                 test.elapsed));
07309             return { new_iters, test.elapsed / test.iterations * new_iters *
07310                 cfg.benchmarkSamples(), fun,
07311                 std::chrono::duration_cast<FloatDuration<Clock>>(cfg.benchmarkWarmupTime()), Detail::warmup_iterations
07312             };
07313         }
07314
07315         template <typename Clock = default_clock>
07316         void run() {
07317             IConfigPtr cfg = getCurrentContext().getConfig();
07318
07319             auto env = Detail::measure_environment<Clock>();
07320
07321             getResultCapture().benchmarkPreparing(name);
07322             CATCH_TRY{
07323                 auto plan = user_code([&] {
07324                     return prepare<Clock>(*cfg, env);
07325                 });
07326             }
07327         }
07328     }
07329 }

```

```

07318         });
07319
07320         BenchmarkInfo info {
07321             name,
07322             plan.estimated_duration.count(),
07323             plan.iterations_per_sample,
07324             cfg->benchmarkSamples(),
07325             cfg->benchmarkResamples(),
07326             env.clock_resolution.mean.count(),
07327             env.clock_cost.mean.count()
07328         };
07329
07330         getResultCapture().benchmarkStarting(info);
07331
07332         auto samples = user_code([&] {
07333             return plan.template run<Clock>(*cfg, env);
07334         });
07335
07336         auto analysis = Detail::analyse(*cfg, env, samples.begin(), samples.end());
07337         BenchmarkStats<FloatDuration<Clock>> stats{ info, analysis.samples, analysis.mean,
analysis.standard_deviation, analysis.outliers, analysis.outlier_variance };
07338         getResultCapture().benchmarkEnded(stats);
07339
07340         } CATCH_CATCH_ALL{
07341             if (translateActiveException() != Detail::benchmarkErrorMsg) // benchmark errors
have been reported, otherwise rethrow.
07342                 std::rethrow_exception(std::current_exception());
07343         }
07344     }
07345
07346     // sets lambda to be used in fun *and* executes benchmark!
07347     template <typename Fun,
07348             typename std::enable_if<!Detail::is_related<Fun, Benchmark>::value, int>::type = 0>
07349             Benchmark & operator=(Fun func) {
07350         fun = Detail::BenchmarkFunction(func);
07351         run();
07352         return *this;
07353     }
07354
07355     explicit operator bool() {
07356         return true;
07357     }
07358
07359     private:
07360         Detail::BenchmarkFunction fun;
07361         std::string name;
07362     };
07363 }
07364 } // namespace Catch
07365
07366 #define INTERNAL_CATCH_GET_1_ARG(arg1, arg2, ...) arg1
07367 #define INTERNAL_CATCH_GET_2_ARG(arg1, arg2, ...) arg2
07368
07369 #define INTERNAL_CATCH_BENCHMARK(BenchmarkName, name, benchmarkIndex)\
07370     if( Catch::Benchmark::Benchmark BenchmarkName{name} ) \
07371         BenchmarkName = [&](int benchmarkIndex)
07372
07373 #define INTERNAL_CATCH_BENCHMARK_ADVANCED(BenchmarkName, name)\
07374     if( Catch::Benchmark::Benchmark BenchmarkName{name} ) \
07375         BenchmarkName = [&]
07376
07377 // end catch_benchmark.hpp
07378 // start catch_constructor.hpp
07379
07380 // Constructor and destructor helpers
07381
07382
07383 #include <type_traits>
07384
07385 namespace Catch {
07386     namespace Benchmark {
07387         namespace Detail {
07388             template <typename T, bool Destruct>
07389             struct ObjectStorage
07390             {
07391                 using TStorage = typename std::aligned_storage<sizeof(T),
std::alignment_of<T>::value>::type;
07392
07393                 ObjectStorage() : data() {}
07394
07395                 ObjectStorage(const ObjectStorage& other)
07396                 {
07397                     new(&data) T(other.stored_object());
07398                 }
07399
07400                 ObjectStorage(ObjectStorage&& other)
07401                 {

```

```

07402         new(&data) T(std::move(other.stored_object()));
07403     }
07404
07405     ~ObjectStorage() { destruct_on_exit<T>(); }
07406
07407     template <typename... Args>
07408     void construct(Args&&... args)
07409     {
07410         new (&data) T(std::forward<Args>(args)...);
07411     }
07412
07413     template <bool AllowManualDestruction = !Destruct>
07414     typename std::enable_if<AllowManualDestruction>::type destruct()
07415     {
07416         stored_object().~T();
07417     }
07418
07419     private:
07420         // If this is a constructor benchmark, destruct the underlying object
07421         template <typename U>
07422         void destruct_on_exit(typename std::enable_if<Destruct, U>::type* = 0) {
07423             destruct<true>(); }
07424         // Otherwise, don't
07425         template <typename U>
07426         void destruct_on_exit(typename std::enable_if<!Destruct, U>::type* = 0) { }
07427
07428         T& stored_object() {
07429             return *static_cast<T*>(static_cast<void*>(&data));
07430         }
07431
07432         T const& stored_object() const {
07433             return *static_cast<T*>(static_cast<void*>(&data));
07434         }
07435
07436         TStorage data;
07437     };
07438
07439     template <typename T>
07440     using storage_for = Detail::ObjectStorage<T, true>;
07441
07442     template <typename T>
07443     using destructable_object = Detail::ObjectStorage<T, false>;
07444 }
07445
07446 // end catch_constructor.hpp
07447 // end catch_benchmarking_all.hpp
07448 #endif
07449
07450 #endif // ! CATCH_CONFIG_IMPL_ONLY
07451
07452 #ifdef CATCH_IMPL
07453 // start catch_impl.hpp
07454
07455 #ifdef __clang__
07456 #pragma clang diagnostic push
07457 #pragma clang diagnostic ignored "-Wweak-vtables"
07458 #endif
07459
07460 // Keep these here for external reporters
07461 // start catch_test_case_tracker.h
07462
07463 #include <string>
07464 #include <vector>
07465 #include <memory>
07466
07467 namespace Catch {
07468     namespace TestCaseTracking {
07469         struct NameAndLocation {
07470             std::string name;
07471             SourceLineInfo location;
07472
07473             NameAndLocation( std::string const& _name, SourceLineInfo const& _location );
07474             friend bool operator==(NameAndLocation const& lhs, NameAndLocation const& rhs) {
07475                 return lhs.name == rhs.name
07476                     && lhs.location == rhs.location;
07477             }
07478         };
07479     };
07480
07481     class ITracker;
07482
07483     using ITrackerPtr = std::shared_ptr<ITracker>;
07484
07485     class ITracker {
07486     public:
07487         NameAndLocation m_nameAndLocation;

```

```

07488
07489 public:
07490     ITracker(NameAndLocation const& nameAndLoc) :
07491         m_nameAndLocation(nameAndLoc)
07492     {}
07493
07494     // static queries
07495     NameAndLocation const& nameAndLocation() const {
07496         return m_nameAndLocation;
07497     }
07498
07499     virtual ~ITracker();
07500
07501     // dynamic queries
07502     virtual bool isComplete() const = 0; // Successfully completed or failed
07503     virtual bool isSuccessfullyCompleted() const = 0;
07504     virtual bool isOpen() const = 0; // Started but not complete
07505     virtual bool hasChildren() const = 0;
07506     virtual bool hasStarted() const = 0;
07507
07508     virtual ITracker& parent() = 0;
07509
07510     // actions
07511     virtual void close() = 0; // Successfully complete
07512     virtual void fail() = 0;
07513     virtual void markAsNeedingAnotherRun() = 0;
07514
07515     virtual void addChild( ITrackerPtr const& child ) = 0;
07516     virtual ITrackerPtr findChild( NameAndLocation const& nameAndLocation ) = 0;
07517     virtual void openChild() = 0;
07518
07519     // Debug/ checking
07520     virtual bool isSectionTracker() const = 0;
07521     virtual bool isGeneratorTracker() const = 0;
07522 };
07523
07524 class TrackerContext {
07525
07526     enum RunState {
07527         NotStarted,
07528         Executing,
07529         CompletedCycle
07530     };
07531
07532     ITrackerPtr m_rootTracker;
07533     ITracker* m_currentTracker = nullptr;
07534     RunState m_runState = NotStarted;
07535
07536 public:
07537     ITracker& startRun();
07538     void endRun();
07539
07540     void startCycle();
07541     void completeCycle();
07542
07543     bool completedCycle() const;
07544     ITracker& currentTracker();
07545     void setCurrentTracker( ITracker* tracker );
07546 };
07547
07548 class TrackerBase : public ITracker {
07549 protected:
07550     enum CycleState {
07551         NotStarted,
07552         Executing,
07553         ExecutingChildren,
07554         NeedsAnotherRun,
07555         CompletedSuccessfully,
07556         Failed
07557     };
07558
07559     using Children = std::vector<ITrackerPtr>;
07560     TrackerContext& m_ctx;
07561     ITracker* m_parent;
07562     Children m_children;
07563     CycleState m_runState = NotStarted;
07564
07565 public:
07566     TrackerBase( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker* parent );
07567
07568     bool isComplete() const override;
07569     bool isSuccessfullyCompleted() const override;
07570     bool isOpen() const override;
07571     bool hasChildren() const override;
07572     bool hasStarted() const override {
07573         return m_runState != NotStarted;
07574     }

```



```

07575     }
07576
07577     void addChild( ITrackerPtr const& child ) override;
07578
07579     ITrackerPtr findChild( NameAndLocation const& nameAndLocation ) override;
07580     ITracker& parent() override;
07581
07582     void openChild() override;
07583
07584     bool isSectionTracker() const override;
07585     bool isGeneratorTracker() const override;
07586
07587     void open();
07588
07589     void close() override;
07590     void fail() override;
07591     void markAsNeedingAnotherRun() override;
07592
07593 private:
07594     void moveToParent();
07595     void moveToThis();
07596 };
07597
07598 class SectionTracker : public TrackerBase {
07599     std::vector<std::string> m_filters;
07600     std::string m_trimmed_name;
07601 public:
07602     SectionTracker( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker* parent
07603 );
07604     bool isSectionTracker() const override;
07605
07606     bool isComplete() const override;
07607
07608     static SectionTracker& acquire( TrackerContext& ctx, NameAndLocation const& nameAndLocation );
07609
07610     void tryOpen();
07611
07612     void addInitialFilters( std::vector<std::string> const& filters );
07613     void addNextFilters( std::vector<std::string> const& filters );
07614     std::vector<std::string> const& getFilters() const;
07615     std::string const& trimmedName() const;
07616 };
07617
07618 };
07619
07620 } // namespace TestCaseTracking
07621
07622 using TestCaseTracking::ITracker;
07623 using TestCaseTracking::TrackerContext;
07624 using TestCaseTracking::SectionTracker;
07625
07626 } // namespace Catch
07627
07628 // end catch_test_case_tracker.h
07629
07630 // start catch_leak_detector.h
07631
07632 namespace Catch {
07633
07634     struct LeakDetector {
07635         LeakDetector();
07636         ~LeakDetector();
07637     };
07638
07639 }
07640 // end catch_leak_detector.h
07641 // Cpp files will be included in the single-header file here
07642 // start catch_stats.cpp
07643
07644 // Statistical analysis tools
07645
07646 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
07647
07648 #include <cassert>
07649 #include <random>
07650
07651 #if defined(CATCH_CONFIG_USE_ASYNC)
07652 #include <future>
07653 #endif
07654
07655 namespace {
07656     double erf_inv(double x) {
07657         // Code accompanying the article "Approximating the erfinv function" in GPU Computing Gems,
07658         // Volume 2
07659         double w, p;
07660         w = -log((1.0 - x) * (1.0 + x));
07661

```

```

07662         if (w < 6.250000) {
07663             w = w - 3.125000;
07664             p = -3.6444120640178196996e-21;
07665             p = -1.685059138182016589e-19 + p * w;
07666             p = 1.2858480715256400167e-18 + p * w;
07667             p = 1.115787767802518096e-17 + p * w;
07668             p = -1.333171662854620906e-16 + p * w;
07669             p = 2.0972767875968561637e-17 + p * w;
07670             p = 6.6376381343583238325e-15 + p * w;
07671             p = -4.0545662729752068639e-14 + p * w;
07672             p = -8.1519341976054721522e-14 + p * w;
07673             p = 2.6335093153082322977e-12 + p * w;
07674             p = -1.2975133253453532498e-11 + p * w;
07675             p = -5.4154120542946279317e-11 + p * w;
07676             p = 1.051212273321532285e-09 + p * w;
07677             p = -4.1126339803469836976e-09 + p * w;
07678             p = -2.9070369957882005086e-08 + p * w;
07679             p = 4.2347877827932403518e-07 + p * w;
07680             p = -1.3654692000834678645e-06 + p * w;
07681             p = -1.3882523362786468719e-05 + p * w;
07682             p = 0.0001867342080340571352 + p * w;
07683             p = -0.00074070253416626697512 + p * w;
07684             p = -0.0060336708714301490533 + p * w;
07685             p = 0.24015818242558961693 + p * w;
07686             p = 1.6536545626831027356 + p * w;
07687         } else if (w < 16.000000) {
07688             w = sqrt(w) - 3.250000;
07689             p = 2.2137376921775787049e-09;
07690             p = 9.0756561938885390979e-08 + p * w;
07691             p = -2.7517406297064545428e-07 + p * w;
07692             p = 1.8239629214389227755e-08 + p * w;
07693             p = 1.5027403968909827627e-06 + p * w;
07694             p = -4.013867526981545969e-06 + p * w;
07695             p = 2.9234449089955446044e-06 + p * w;
07696             p = 1.2475304481671778723e-05 + p * w;
07697             p = -4.7318229009055733981e-05 + p * w;
07698             p = 6.8284851459573175448e-05 + p * w;
07699             p = 2.4031110387097893999e-05 + p * w;
07700             p = -0.0003550375203628474796 + p * w;
07701             p = 0.00095328937973738049703 + p * w;
07702             p = -0.0016882755560235047313 + p * w;
07703             p = 0.0024914420961078508066 + p * w;
07704             p = -0.0037512085075692412107 + p * w;
07705             p = 0.005370914553590063617 + p * w;
07706             p = 1.0052589676941592334 + p * w;
07707             p = 3.0838856104922207635 + p * w;
07708         } else {
07709             w = sqrt(w) - 5.000000;
07710             p = -2.7109920616438573243e-11;
07711             p = -2.5556418169965252055e-10 + p * w;
07712             p = 1.5076572693500548083e-09 + p * w;
07713             p = -3.7894654401267369937e-09 + p * w;
07714             p = 7.6157012080783393804e-09 + p * w;
07715             p = -1.4960026627149240478e-08 + p * w;
07716             p = 2.9147953450901080826e-08 + p * w;
07717             p = -6.7711997758452339498e-08 + p * w;
07718             p = 2.2900482228026654717e-07 + p * w;
07719             p = -9.9298272942317002539e-07 + p * w;
07720             p = 4.5260625972231537039e-06 + p * w;
07721             p = -1.9681778105531670567e-05 + p * w;
07722             p = 7.5995277030017761139e-05 + p * w;
07723             p = -0.00021503011930044477347 + p * w;
07724             p = -0.00013871931833623122026 + p * w;
07725             p = 1.0103004648645343977 + p * w;
07726             p = 4.8499064014085844221 + p * w;
07727         }
07728         return p * x;
07729     }
07730
07731     double standard_deviation(std::vector<double>::iterator first, std::vector<double>::iterator last)
07732     {
07733         auto m = Catch::Benchmark::Detail::mean(first, last);
07734         double variance = std::accumulate(first, last, 0., [m](double a, double b) {
07735             double diff = b - m;
07736             return a + diff * diff;
07737         }) / (last - first);
07738         return std::sqrt(variance);
07739     }
07740 }
07741
07742 namespace Catch {
07743     namespace Benchmark {
07744         namespace Detail {
07745             double weighted_average_quantile(int k, int q, std::vector<double>::iterator first,
07746                 std::vector<double>::iterator last) {

```

```

07747         auto count = last - first;
07748         double idx = (count - 1) * k / static_cast<double>(q);
07749         int j = static_cast<int>(idx);
07750         double g = idx - j;
07751         std::nth_element(first, first + j, last);
07752         auto xj = first[j];
07753         if (g == 0) return xj;
07754
07755         auto xj1 = *std::min_element(first + (j + 1), last);
07756         return xj + g * (xj1 - xj);
07757     }
07758
07759     double erfc_inv(double x) {
07760         return erf_inv(1.0 - x);
07761     }
07762
07763     double normal_quantile(double p) {
07764         static const double ROOT_TWO = std::sqrt(2.0);
07765
07766         double result = 0.0;
07767         assert(p >= 0 && p <= 1);
07768         if (p < 0 || p > 1) {
07769             return result;
07770         }
07771
07772         result = -erfc_inv(2.0 * p);
07773         // result += normal distribution standard deviation (1.0) * sqrt(2)
07774         result *= /*sd * */ ROOT_TWO;
07775         // result += normal distribution mean (0)
07776         return result;
07777     }
07778
07779     double outlier_variance(Estimate<double> mean, Estimate<double> stddev, int n) {
07780         double sb = stddev.point();
07781         double mn = mean.point() / n;
07782         double mg_min = mn / 2.;
07783         double sg = (std::min)(mg_min / 4., sb / std::sqrt(n));
07784         double sg2 = sg * sg;
07785         double sb2 = sb * sb;
07786
07787         auto c_max = [n, mn, sb2, sg2](double x) -> double {
07788             double k = mn - x;
07789             double d = k * k;
07790             double nd = n * d;
07791             double k0 = -n * nd;
07792             double k1 = sb2 - n * sg2 + nd;
07793             double det = k1 * k1 - 4 * sg2 * k0;
07794             return (int)(-2. * k0 / (k1 + std::sqrt(det)));
07795         };
07796
07797         auto var_out = [n, sb2, sg2](double c) {
07798             double nc = n - c;
07799             return (nc / n) * (sb2 - nc * sg2);
07800         };
07801
07802         return (std::min)(var_out(1), var_out((std::min)(c_max(0.), c_max(mg_min)))) / sb2;
07803     }
07804
07805     bootstrap_analysis analyse_samples(double confidence_level, int n_resamples,
07806 std::vector<double>::iterator first, std::vector<double>::iterator last) {
07807         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION
07808         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS
07809         static std::random_device entropy;
07810         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
07811
07812         auto n = static_cast<int>(last - first); // seriously, one can't use integral types
07813         without hell in C++
07814
07815         auto mean = &Detail::mean<std::vector<double>::iterator>;
07816         auto stddev = &standard_deviation;
07817
07818         #if defined(CATCH_CONFIG_USE_ASYNC)
07819         auto Estimate = [=](double(*f)(std::vector<double>::iterator,
07820 std::vector<double>::iterator)) {
07821             auto seed = entropy();
07822             return std::async(std::launch::async, [=] {
07823                 std::mt19937 rng(seed);
07824                 auto resampled = resample(rng, n_resamples, first, last, f);
07825                 return bootstrap(confidence_level, first, last, resampled, f);
07826             });
07827         };
07828
07829         auto mean_future = Estimate(mean);
07830         auto stddev_future = Estimate(stddev);
07831
07832         auto mean_estimate = mean_future.get();
07833         auto stddev_estimate = stddev_future.get();

```

```

07831 #else
07832     auto Estimate = [=](double(*f)(std::vector<double>::iterator,
std::vector<double>::iterator)) {
07833         auto seed = entropy();
07834         std::mt19937 rng(seed);
07835         auto resampled = resample(rng, n_resamples, first, last, f);
07836         return bootstrap(confidence_level, first, last, resampled, f);
07837     };
07838
07839     auto mean_estimate = Estimate(mean);
07840     auto stddev_estimate = Estimate(stddev);
07841 #endif // CATCH_USE_ASYNC
07842
07843     double outlier_variance = Detail::outlier_variance(mean_estimate, stddev_estimate, n);
07844
07845     return { mean_estimate, stddev_estimate, outlier_variance };
07846 }
07847 } // namespace Detail
07848 } // namespace Benchmark
07849 } // namespace Catch
07850
07851 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
07852 // end catch_stats.cpp
07853 // start catch_approx.cpp
07854
07855 #include <cmath>
07856 #include <limits>
07857
07858 namespace {
07859
07860 // Performs equivalent check of std::fabs(lhs - rhs) <= margin
07861 // But without the subtraction to allow for INFINITY in comparison
07862 bool marginComparison(double lhs, double rhs, double margin) {
07863     return (lhs + margin >= rhs) && (rhs + margin >= lhs);
07864 }
07865
07866 namespace Catch {
07867     namespace Detail {
07871         Approx::Approx ( double value )
07872         :   m_epsilon( std::numeric_limits<float>::epsilon()*100 ),
07873             m_margin( 0.0 ),
07874             m_scale( 0.0 ),
07875             m_value( value )
07876         {}
07877
07878         Approx Approx::custom() {
07879             return Approx( 0 );
07880         }
07881
07882         Approx Approx::operator-() const {
07883             auto temp(*this);
07884             temp.m_value = -temp.m_value;
07885             return temp;
07886         }
07887
07888         std::string Approx::toString() const {
07889             ReusableStringStream rss;
07890             rss << "Approx( " << ::Catch::Detail::stringify( m_value ) << " )";
07891             return rss.str();
07892         }
07893
07894         bool Approx::equalityComparisonImpl(const double other) const {
07895             // First try with fixed margin, then compute margin based on epsilon, scale and Approx's value
07896             // Thanks to Richard Harris for his help refining the scaled margin value
07897             return marginComparison(m_value, other, m_margin) ||
07898                 marginComparison(m_value, other, m_epsilon * (m_scale + std::fabs(std::isinf(m_value)?
0 : m_value)));
07899         }
07900
07901         void Approx::setMargin(double newMargin) {
07902             CATCH_ENFORCE(newMargin >= 0,
07903                 "Invalid Approx::margin: " << newMargin << "."
07904                 << " Approx::Margin has to be non-negative.");
07905             m_margin = newMargin;
07906         }
07907
07908         void Approx::setEpsilon(double newEpsilon) {
07909             CATCH_ENFORCE(newEpsilon >= 0 && newEpsilon <= 1.0,
07910                 "Invalid Approx::epsilon: " << newEpsilon << "."
07911                 << " Approx::epsilon has to be in [0, 1]");
07912             m_epsilon = newEpsilon;
07913         }
07914
07915 } // end namespace Detail

```

```

07916
07917 namespace literals {
07918     Detail::Approx operator "" _a(long double val) {
07919         return Detail::Approx(val);
07920     }
07921     Detail::Approx operator "" _a(unsigned long long val) {
07922         return Detail::Approx(val);
07923     }
07924 } // end namespace literals
07925
07926 std::string StringMaker<Catch::Detail::Approx>::convert(Catch::Detail::Approx const& value) {
07927     return value.toString();
07928 }
07929
07930 } // end namespace Catch
07931 // end catch_approx.cpp
07932 // start catch_assertionhandler.cpp
07933
07934 // start catch_debugger.h
07935
07936 namespace Catch {
07937     bool isDebuggerActive();
07938 }
07939
07940 #ifdef CATCH_PLATFORM_MAC
07941
07942     #if defined(__i386__) || defined(__x86_64__)
07943         #define CATCH_TRAP() __asm__("int $3\n" : : ) /* NOLINT */
07944     #elif defined(__aarch64__)
07945         #define CATCH_TRAP() __asm__(".inst 0xd4200000")
07946     #endif
07947
07948 #elif defined(CATCH_PLATFORM_IPHONE)
07949
07950     // use inline assembler
07951     #if defined(__i386__) || defined(__x86_64__)
07952         #define CATCH_TRAP() __asm__("int $3")
07953     #elif defined(__aarch64__)
07954         #define CATCH_TRAP() __asm__(".inst 0xd4200000")
07955     #elif defined(__arm__) && !defined(__thumb__)
07956         #define CATCH_TRAP() __asm__(".inst 0xe7f001f0")
07957     #elif defined(__arm__) && defined(__thumb__)
07958         #define CATCH_TRAP() __asm__(".inst 0xde01")
07959     #endif
07960
07961 #elif defined(CATCH_PLATFORM_LINUX)
07962     // If we can use inline assembler, do it because this allows us to break
07963     // directly at the location of the failing check instead of breaking inside
07964     // raise() called from it, i.e. one stack frame below.
07965     #if defined(__GNUC__) && (defined(__i386__) || defined(__x86_64__))
07966         #define CATCH_TRAP() asm volatile ("int $3") /* NOLINT */
07967     #else // Fall back to the generic way.
07968         #include <signal.h>
07969
07970         #define CATCH_TRAP() raise(SIGTRAP)
07971     #endif
07972 #elif defined(_MSC_VER)
07973     #define CATCH_TRAP() __debugbreak()
07974 #elif defined(__MINGW32__)
07975     extern "C" __declspec(dllimport) void __stdcall DebugBreak();
07976     #define CATCH_TRAP() DebugBreak()
07977 #endif
07978
07979 #ifndef CATCH_BREAK_INTO_DEBUGGER
07980     #define CATCH_TRAP
07981     #define CATCH_BREAK_INTO_DEBUGGER() []{ if( Catch::isDebuggerActive() ) { CATCH_TRAP(); } }()
07982 #else
07983     #define CATCH_BREAK_INTO_DEBUGGER() []{}()
07984 #endif
07985 #endif
07986
07987 // end catch_debugger.h
07988 // start catch_run_context.h
07989
07990 // start catch_fatal_condition.h
07991
07992 #include <cassert>
07993
07994 namespace Catch {
07995
07996     // Wrapper for platform-specific fatal error (signals/SEH) handlers
07997     //
07998     // Tries to be cooperative with other handlers, and not step over
07999     // other handlers. This means that unknown structured exceptions
08000     // are passed on, previous signal handlers are called, and so on.
08001     //
08002     // Can only be instantiated once, and assumes that once a signal

```

```

08003 // is caught, the binary will end up terminating. Thus, there
08004 class FatalConditionHandler {
08005     bool m_started = false;
08006
08007     // Install/disengage implementation for specific platform.
08008     // Should be if-defed to work on current platform, can assume
08009     // engage-disengage 1:1 pairing.
08010     void engage_platform();
08011     void disengage_platform();
08012 public:
08013     // Should also have platform-specific implementations as needed
08014     FatalConditionHandler();
08015     ~FatalConditionHandler();
08016
08017     void engage() {
08018         assert(!m_started && "Handler cannot be installed twice.");
08019         m_started = true;
08020         engage_platform();
08021     }
08022
08023     void disengage() {
08024         assert(m_started && "Handler cannot be uninstalled without being installed first");
08025         m_started = false;
08026         disengage_platform();
08027     }
08028 };
08029
08031 class FatalConditionHandlerGuard {
08032     FatalConditionHandler* m_handler;
08033 public:
08034     FatalConditionHandlerGuard(FatalConditionHandler* handler):
08035         m_handler(handler) {
08036         m_handler->engage();
08037     }
08038     ~FatalConditionHandlerGuard() {
08039         m_handler->disengage();
08040     }
08041 };
08042
08043 } // end namespace Catch
08044
08045 // end catch_fatal_condition.h
08046 #include <string>
08047
08048 namespace Catch {
08049
08050     struct IMutableContext;
08051
08052
08053
08054     class RunContext : public IResultCapture, public IRunner {
08055 public:
08056         RunContext( RunContext const& ) = delete;
08057         RunContext& operator = ( RunContext const& ) = delete;
08058
08059         explicit RunContext( IConfigPtr const& _config, IStreamingReporterPtr&& reporter );
08060
08061         ~RunContext() override;
08062
08063         void testGroupStarting( std::string const& testSpec, std::size_t groupIndex, std::size_t
groupsCount );
08064         void testGroupEnded( std::string const& testSpec, Totals const& totals, std::size_t
groupIndex, std::size_t groupsCount );
08065
08066         Totals runTest( TestCase const& testCase);
08067
08068         IConfigPtr config() const;
08069         IStreamingReporter& reporter() const;
08070
08071 public: // IResultCapture
08072
08073         // Assertion handlers
08074         void handleExpr
08075             ( AssertionInfo const& info,
08076               ITransientExpression const& expr,
08077               AssertionReaction& reaction ) override;
08078         void handleMessage
08079             ( AssertionInfo const& info,
08080               ResultWas::OfType resultType,
08081               StringRef const& message,
08082               AssertionReaction& reaction ) override;
08083         void handleUnexpectedExceptionNotThrown
08084             ( AssertionInfo const& info,
08085               AssertionReaction& reaction ) override;
08086         void handleUnexpectedInflightException
08087             ( AssertionInfo const& info,
08088               std::string const& message,

```

```

08090         AssertionReaction& reaction ) override;
08091     void handleIncomplete
08092         ( AssertionInfo const& info ) override;
08093     void handleNonExpr
08094         ( AssertionInfo const& info,
08095           ResultWas::OfType resultType,
08096           AssertionReaction &reaction ) override;
08097
08098     bool sectionStarted( SectionInfo const& sectionInfo, Counts& assertions ) override;
08099
08100     void sectionEnded( SectionEndInfo const& endInfo ) override;
08101     void sectionEndedEarly( SectionEndInfo const& endInfo ) override;
08102
08103     auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
08104     IGeneratorTracker& override;
08105
08106     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
08107     void benchmarkPreparing( std::string const& name ) override;
08108     void benchmarkStarting( BenchmarkInfo const& info ) override;
08109     void benchmarkEnded( BenchmarkStats<> const& stats ) override;
08110     void benchmarkFailed( std::string const& error ) override;
08111     #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
08112
08113     void pushScopedMessage( MessageInfo const& message ) override;
08114     void popScopedMessage( MessageInfo const& message ) override;
08115
08116     void emplaceUnscopedMessage( MessageBuilder const& builder ) override;
08117
08118     std::string getCurrentTestName() const override;
08119
08120     const AssertionResult* getLastResult() const override;
08121
08122     void exceptionEarlyReported() override;
08123
08124     void handleFatalErrorCondition( StringRef message ) override;
08125
08126     bool lastAssertionPassed() override;
08127
08128     void assertionPassed() override;
08129
08130     public:
08131         // TBD We need to do this another way!
08132         bool aborting() const final;
08133
08134     private:
08135         void runCurrentTest( std::string& redirectedCout, std::string& redirectedCerr );
08136         void invokeActiveTestCase();
08137
08138         void resetAssertionInfo();
08139         bool testForMissingAssertions( Counts& assertions );
08140
08141         void assertionEnded( AssertionResult const& result );
08142         void reportExpr
08143             ( AssertionInfo const& info,
08144               ResultWas::OfType resultType,
08145               ITransientExpression const *expr,
08146               bool negated );
08147
08148         void populateReaction( AssertionReaction& reaction );
08149
08150     private:
08151         void handleUnfinishedSections();
08152
08153         TestRunInfo m_runInfo;
08154         IMutableContext& m_context;
08155         TestCase const* m_activeTestCase = nullptr;
08156         ITracker* m_testCaseTracker = nullptr;
08157         Option<AssertionResult> m_lastResult;
08158
08159         IConfigPtr m_config;
08160         Totals m_totals;
08161         IStreamingReporterPtr m_reporter;
08162         std::vector<MessageInfo> m_messages;
08163         std::vector<ScopedMessage> m_messageScopes; /* Keeps owners of so-called unscoped messages. */
08164         AssertionInfo m_lastAssertionInfo;
08165         std::vector<SectionEndInfo> m_unfinishedSections;
08166         std::vector<ITracker*> m_activeSections;
08167         TrackerContext m_trackerContext;
08168         FatalConditionHandler m_fatalConditionhandler;
08169         bool m_lastAssertionPassed = false;
08170         bool m_shouldReportUnexpected = true;
08171         bool m_includeSuccessfulResults;
08172     };
08173
08174     void seedRng(IConfig const& config);

```

```

08176     unsigned int rngSeed();
08177 } // end namespace Catch
08178
08179 // end catch_run_context.h
08180 namespace Catch {
08181
08182     namespace {
08183         auto operator «( std::ostream& os, ITransientExpression const& expr ) -> std::ostream& {
08184             expr.streamReconstructedExpression( os );
08185             return os;
08186         }
08187     }
08188
08189     LazyExpression::LazyExpression( bool isNegated )
08190     :   m_isNegated( isNegated )
08191     {}
08192
08193     LazyExpression::LazyExpression( LazyExpression const& other ) : m_isNegated( other.m_isNegated )
08194 {}
08195
08196     LazyExpression::operator bool() const {
08197         return m_transientExpression != nullptr;
08198     }
08199
08200     auto operator « ( std::ostream& os, LazyExpression const& lazyExpr ) -> std::ostream& {
08201         if( lazyExpr.m_isNegated )
08202             os << "!";
08203
08204         if( lazyExpr ) {
08205             if( lazyExpr.m_isNegated && lazyExpr.m_transientExpression->isBinaryExpression() )
08206                 os << "(" << *lazyExpr.m_transientExpression << ")";
08207             else
08208                 os << *lazyExpr.m_transientExpression;
08209         }
08210         else {
08211             os << "{** error - unchecked empty expression requested **}";
08212         }
08213         return os;
08214     }
08215
08216     AssertionHandler::AssertionHandler
08217     (   StringRef const& macroName,
08218         SourceLineInfo const& lineInfo,
08219         StringRef capturedExpression,
08220         ResultDisposition::Flags resultDisposition )
08221     :   m_assertionInfo{ macroName, lineInfo, capturedExpression, resultDisposition },
08222         m_resultCapture( getResultCapture() )
08223     {}
08224
08225     void AssertionHandler::handleExpr( ITransientExpression const& expr ) {
08226         m_resultCapture.handleExpr( m_assertionInfo, expr, m_reaction );
08227     }
08228     void AssertionHandler::handleMessage(ResultWas::OfType resultType, StringRef const& message) {
08229         m_resultCapture.handleMessage( m_assertionInfo, resultType, message, m_reaction );
08230     }
08231
08232     auto AssertionHandler::allowThrows() const -> bool {
08233         return getCurrentContext().getConfig()->allowThrows();
08234     }
08235
08236     void AssertionHandler::complete() {
08237         setCompleted();
08238         if( m_reaction.shouldDebugBreak ) {
08239             // If you find your debugger stopping you here then go one level up on the
08240             // call-stack for the code that caused it (typically a failed assertion)
08241
08242             // (To go back to the test and change execution, jump over the throw, next)
08243             CATCH_BREAK_INTO_DEBUGGER();
08244         }
08245         if (m_reaction.shouldThrow) {
08246 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
08247             throw Catch::TestFailureException();
08248 #else
08249             CATCH_ERROR( "Test failure requires aborting test!" );
08250 #endif
08251         }
08252     }
08253     void AssertionHandler::setCompleted() {
08254         m_completed = true;
08255     }
08256
08257     void AssertionHandler::handleUnexpectedInflightException() {
08258         m_resultCapture.handleUnexpectedInflightException( m_assertionInfo,
08259             Catch::translateActiveException(), m_reaction );
08260     }

```



```

08261     void AssertionHandler::handleExceptionThrownAsExpected() {
08262         m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
08263     }
08264     void AssertionHandler::handleExceptionNotThrownAsExpected() {
08265         m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
08266     }
08267
08268     void AssertionHandler::handleUnexpectedExceptionNotThrown() {
08269         m_resultCapture.handleUnexpectedExceptionNotThrown( m_assertionInfo, m_reaction );
08270     }
08271
08272     void AssertionHandler::handleThrowingCallSkipped() {
08273         m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
08274     }
08275
08276     // This is the overload that takes a string and infers the Equals matcher from it
08277     // The more general overload, that takes any string matcher, is in catch_capture_matchers.cpp
08278     void handleExceptionMatchExpr( AssertionHandler& handler, std::string const& str, StringRef const&
matcherString ) {
08279         handleExceptionMatchExpr( handler, Matchers::Equals( str ), matcherString );
08280     }
08281
08282 } // namespace Catch
08283 // end catch_assertionhandler.cpp
08284 // start catch_assertionresult.cpp
08285
08286 namespace Catch {
08287     AssertionResultData::AssertionResultData(ResultWas::OfType _resultType, LazyExpression const &
_lazyExpression):
08288         lazyExpression(_lazyExpression),
08289         resultType(_resultType) {}
08290
08291     std::string AssertionResultData::reconstructExpression() const {
08292
08293         if( reconstructedExpression.empty() ) {
08294             if( lazyExpression ) {
08295                 ReusableStringStream rss;
08296                 rss << lazyExpression;
08297                 reconstructedExpression = rss.str();
08298             }
08299         }
08300         return reconstructedExpression;
08301     }
08302
08303     AssertionResult::AssertionResult( AssertionInfo const& info, AssertionResultData const& data )
08304     :   m_info( info ),
08305         m_resultData( data )
08306     {}
08307
08308     // Result was a success
08309     bool AssertionResult::succeeded() const {
08310         return Catch::isOk( m_resultData.resultType );
08311     }
08312
08313     // Result was a success, or failure is suppressed
08314     bool AssertionResult::isOk() const {
08315         return Catch::isOk( m_resultData.resultType ) || shouldSuppressFailure(
m_info.resultDisposition );
08316     }
08317
08318     ResultWas::OfType AssertionResult::getResultType() const {
08319         return m_resultData.resultType;
08320     }
08321
08322     bool AssertionResult::hasExpression() const {
08323         return !m_info.capturedExpression.empty();
08324     }
08325
08326     bool AssertionResult::hasMessage() const {
08327         return !m_resultData.message.empty();
08328     }
08329
08330     std::string AssertionResult::getExpression() const {
08331         // Possibly overallocating by 3 characters should be basically free
08332         std::string expr; expr.reserve(m_info.capturedExpression.size() + 3);
08333         if (isFalseTest(m_info.resultDisposition)) {
08334             expr += "!(";
08335         }
08336         expr += m_info.capturedExpression;
08337         if (isFalseTest(m_info.resultDisposition)) {
08338             expr += ')';
08339         }
08340         return expr;
08341     }
08342
08343     std::string AssertionResult::getExpressionInMacro() const {
08344         std::string expr;

```

```

08345         if( m_info.macroName.empty() )
08346             expr = static_cast<std::string>(m_info.capturedExpression);
08347         else {
08348             expr.reserve( m_info.macroName.size() + m_info.capturedExpression.size() + 4 );
08349             expr += m_info.macroName;
08350             expr += " ( ";
08351             expr += m_info.capturedExpression;
08352             expr += " ) ";
08353         }
08354         return expr;
08355     }
08356
08357     bool AssertionResult::hasExpandedExpression() const {
08358         return hasExpression() && getExpandedExpression() != getExpression();
08359     }
08360
08361     std::string AssertionResult::getExpandedExpression() const {
08362         std::string expr = m_resultData.reconstructExpression();
08363         return expr.empty()
08364             ? getExpression()
08365             : expr;
08366     }
08367
08368     std::string AssertionResult::getMessage() const {
08369         return m_resultData.message;
08370     }
08371     SourceLineInfo AssertionResult::getSourceInfo() const {
08372         return m_info.lineInfo;
08373     }
08374
08375     StringRef AssertionResult::getTestMacroName() const {
08376         return m_info.macroName;
08377     }
08378
08379 } // end namespace Catch
08380 // end catch_assertionresult.cpp
08381 // start catch_capture_matchers.cpp
08382
08383 namespace Catch {
08384
08385     using StringMatcher = Matchers::Impl::MatcherBase<std::string>;
08386
08387     // This is the general overload that takes a any string matcher
08388     // There is another overload, in catch_assertionhandler.h/.cpp, that only takes a string and
08389     // infers
08390     // the Equals matcher (so the header does not mention matchers)
08391     void handleExceptionMatchExpr( AssertionHandler& handler, StringMatcher const& matcher, StringRef
08392     const& matcherString ) {
08393         std::string exceptionMessage = Catch::translateActiveException();
08394         MatchExpr<std::string, StringMatcher const&> expr( exceptionMessage, matcher, matcherString );
08395         handler.handleExpr( expr );
08396     }
08397 } // namespace Catch
08398 // end catch_capture_matchers.cpp
08399 // start catch_commandline.cpp
08400 // start catch_commandline.h
08401
08402 // start catch_clara.h
08403
08404 // Use Catch's value for console width (store Clara's off to the side, if present)
08405 #ifdef CLARA_CONFIG_CONSOLE_WIDTH
08406 #define CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
08407 #undef CATCH_CLARA_CONFIG_CONSOLE_WIDTH
08408 #endif
08409 #define CATCH_CLARA_CONFIG_CONSOLE_WIDTH CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
08410
08411 #ifdef __clang__
08412 #pragma clang diagnostic push
08413 #pragma clang diagnostic ignored "-Wweak-vtables"
08414 #pragma clang diagnostic ignored "-Wexit-time-destructors"
08415 #pragma clang diagnostic ignored "-Wshadow"
08416 #endif
08417
08418 // start clara.hpp
08419 // Copyright 2017 Two Blue Cubes Ltd. All rights reserved.
08420 //
08421 // Distributed under the Boost Software License, Version 1.0. (See accompanying
08422 // file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
08423 //
08424 // See https://github.com/philsquared/Clara for more details
08425
08426 // Clara v1.1.5
08427
08428 #ifndef CATCH_CLARA_CONFIG_CONSOLE_WIDTH
08429 #define CATCH_CLARA_CONFIG_CONSOLE_WIDTH

```

```

08430 #define CATCH_CLARA_CONFIG_CONSOLE_WIDTH 80
08431 #endif
08432
08433 #ifndef CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
08434 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH CATCH_CLARA_CONFIG_CONSOLE_WIDTH
08435 #endif
08436
08437 #ifndef CLARA_CONFIG_OPTIONAL_TYPE
08438 #ifdef __has_include
08439 #if __has_include(<optional>) && __cplusplus >= 201703L
08440 #include <optional>
08441 #define CLARA_CONFIG_OPTIONAL_TYPE std::optional
08442 #endif
08443 #endif
08444 #endif
08445
08446 // ----- #included from clara_textflow.hpp -----
08447
08448 // TextFlowCpp
08449 //
08450 // A single-header library for wrapping and laying out basic text, by Phil Nash
08451 //
08452 // Distributed under the Boost Software License, Version 1.0. (See accompanying
08453 // file LICENSE.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
08454 //
08455 // This project is hosted at https://github.com/philsquared/textflowcpp
08456
08457 #include <cassert>
08458 #include <ostream>
08459 #include <sstream>
08460 #include <vector>
08461
08462 #ifndef CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
08463 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH 80
08464 #endif
08465
08466 namespace Catch {
08467 namespace clara {
08468 namespace TextFlow {
08469
08470 inline auto isWhitespace(char c) -> bool {
08471     static std::string chars = " \t\n\r";
08472     return chars.find(c) != std::string::npos;
08473 }
08474
08475 inline auto isBreakableBefore(char c) -> bool {
08476     static std::string chars = "[(|<|";
08477     return chars.find(c) != std::string::npos;
08478 }
08479
08480 inline auto isBreakableAfter(char c) -> bool {
08481     static std::string chars = "])>.,;+*=~/\\|";
08482     return chars.find(c) != std::string::npos;
08483 }
08484
08485 class Columns;
08486
08487 class Column {
08488     std::vector<std::string> m_strings;
08489     size_t m_width = CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH;
08490     size_t m_indent = 0;
08491     size_t m_initialIndent = std::string::npos;
08492
08493 public:
08494     class iterator {
08495         friend Column;
08496
08497         Column const& m_column;
08498         size_t m_stringIndex = 0;
08499         size_t m_pos = 0;
08500
08501         size_t m_len = 0;
08502         size_t m_end = 0;
08503         bool m_suffix = false;
08504
08505         iterator(Column const& column, size_t stringIndex)
08506             : m_column(column),
08507               m_stringIndex(stringIndex) {}
08508
08509         auto line() const -> std::string const& { return m_column.m_strings[m_stringIndex]; }
08510
08511         auto isBoundary(size_t at) const -> bool {
08512             assert(at > 0);
08513             assert(at <= line().size());
08514
08515             return at == line().size() ||
08516                    (isWhitespace(line()[at]) && !isWhitespace(line()[at - 1])) ||
08517                    isBreakableBefore(line()[at]) ||

```

```

08517         isBreakableAfter(line()[at - 1]);
08518     }
08519
08520     void calcLength() {
08521         assert(m_stringIndex < m_column.m_strings.size());
08522
08523         m_suffix = false;
08524         auto width = m_column.m_width - indent();
08525         m_end = m_pos;
08526         if (line()[m_pos] == '\n') {
08527             ++m_end;
08528         }
08529         while (m_end < line().size() && line()[m_end] != '\n')
08530             ++m_end;
08531
08532         if (m_end < m_pos + width) {
08533             m_len = m_end - m_pos;
08534         } else {
08535             size_t len = width;
08536             while (len > 0 && !isBoundary(m_pos + len))
08537                 --len;
08538             while (len > 0 && isWhitespace(line()[m_pos + len - 1]))
08539                 --len;
08540
08541             if (len > 0) {
08542                 m_len = len;
08543             } else {
08544                 m_suffix = true;
08545                 m_len = width - 1;
08546             }
08547         }
08548     }
08549
08550     auto indent() const -> size_t {
08551         auto initial = m_pos == 0 && m_stringIndex == 0 ? m_column.m_initialIndent :
std::string::npos;
08552         return initial == std::string::npos ? m_column.m_indent : initial;
08553     }
08554
08555     auto addIndentAndSuffix(std::string const &plain) const -> std::string {
08556         return std::string(indent(), ' ') + (m_suffix ? plain + "-" : plain);
08557     }
08558
08559     public:
08560         using difference_type = std::ptrdiff_t;
08561         using value_type = std::string;
08562         using pointer = value_type * ;
08563         using reference = value_type & ;
08564         using iterator_category = std::forward_iterator_tag;
08565
08566         explicit iterator(Column const& column) : m_column(column) {
08567             assert(m_column.m_width > m_column.m_indent);
08568             assert(m_column.m_initialIndent == std::string::npos || m_column.m_width >
m_column.m_initialIndent);
08569             calcLength();
08570             if (m_len == 0)
08571                 m_stringIndex++; // Empty string
08572         }
08573
08574         auto operator *() const -> std::string {
08575             assert(m_stringIndex < m_column.m_strings.size());
08576             assert(m_pos <= m_end);
08577             return addIndentAndSuffix(line().substr(m_pos, m_len));
08578         }
08579
08580         auto operator ++() -> iterator& {
08581             m_pos += m_len;
08582             if (m_pos < line().size() && line()[m_pos] == '\n')
08583                 m_pos += 1;
08584             else
08585                 while (m_pos < line().size() && isWhitespace(line()[m_pos]))
08586                     ++m_pos;
08587
08588             if (m_pos == line().size()) {
08589                 m_pos = 0;
08590                 ++m_stringIndex;
08591             }
08592             if (m_stringIndex < m_column.m_strings.size())
08593                 calcLength();
08594             return *this;
08595         }
08596         auto operator ++(int) -> iterator {
08597             iterator prev(*this);
08598             operator ++();
08599             return prev;
08600         }
08601

```

```

08602         auto operator ==(iterator const& other) const -> bool {
08603             return
08604                 m_pos == other.m_pos &&
08605                 m_stringIndex == other.m_stringIndex &&
08606                 &m_column == &other.m_column;
08607         }
08608         auto operator !=(iterator const& other) const -> bool {
08609             return !operator==(other);
08610         }
08611     };
08612     using const_iterator = iterator;
08613
08614     explicit Column(std::string const& text) { m_strings.push_back(text); }
08615
08616     auto width(size_t newWidth) -> Column& {
08617         assert(newWidth > 0);
08618         m_width = newWidth;
08619         return *this;
08620     }
08621     auto indent(size_t newIndent) -> Column& {
08622         m_indent = newIndent;
08623         return *this;
08624     }
08625     auto initialIndent(size_t newIndent) -> Column& {
08626         m_initialIndent = newIndent;
08627         return *this;
08628     }
08629
08630     auto width() const -> size_t { return m_width; }
08631     auto begin() const -> iterator { return iterator(*this); }
08632     auto end() const -> iterator { return { *this, m_strings.size() }; }
08633
08634     inline friend std::ostream& operator < (std::ostream& os, Column const& col) {
08635         bool first = true;
08636         for (auto line : col) {
08637             if (first)
08638                 first = false;
08639             else
08640                 os << "\n";
08641             os << line;
08642         }
08643         return os;
08644     }
08645
08646     auto operator + (Column const& other) -> Columns;
08647
08648     auto toString() const -> std::string {
08649         std::ostringstream oss;
08650         oss << *this;
08651         return oss.str();
08652     }
08653 };
08654
08655 class Spacer : public Column {
08656 public:
08657     explicit Spacer(size_t spaceWidth) : Column("") {
08658         width(spaceWidth);
08659     }
08660 };
08661
08662 class Columns {
08663     std::vector<Column> m_columns;
08664 public:
08665     class iterator {
08666     friend Columns;
08667     struct EndTag {};
08668
08669     std::vector<Column> const& m_columns;
08670     std::vector<Column::iterator> m_iterators;
08671     size_t m_activeIterators;
08672
08673     iterator(Columns const& columns, EndTag)
08674         : m_columns(columns.m_columns),
08675         m_activeIterators(0) {
08676         m_iterators.reserve(m_columns.size());
08677
08678         for (auto const& col : m_columns)
08679             m_iterators.push_back(col.end());
08680     }
08681
08682     public:
08683         using difference_type = std::ptrdiff_t;
08684         using value_type = std::string;
08685         using pointer = value_type * ;

```

```

08689     using reference = value_type & ;
08690     using iterator_category = std::forward_iterator_tag;
08691
08692     explicit iterator(Columns const& columns)
08693         : m_columns(columns.m_columns),
08694         m_activeIterators(m_columns.size()) {
08695         m_iterators.reserve(m_columns.size());
08696
08697         for (auto const& col : m_columns)
08698             m_iterators.push_back(col.begin());
08699     }
08700
08701     auto operator ==(iterator const& other) const -> bool {
08702         return m_iterators == other.m_iterators;
08703     }
08704     auto operator !=(iterator const& other) const -> bool {
08705         return m_iterators != other.m_iterators;
08706     }
08707     auto operator *() const -> std::string {
08708         std::string row, padding;
08709
08710         for (size_t i = 0; i < m_columns.size(); ++i) {
08711             auto width = m_columns[i].width();
08712             if (m_iterators[i] != m_columns[i].end()) {
08713                 std::string col = *m_iterators[i];
08714                 row += padding + col;
08715                 if (col.size() < width)
08716                     padding = std::string(width - col.size(), ' ');
08717                 else
08718                     padding = "";
08719             } else {
08720                 padding += std::string(width, ' ');
08721             }
08722         }
08723         return row;
08724     }
08725     auto operator ++() -> iterator& {
08726         for (size_t i = 0; i < m_columns.size(); ++i) {
08727             if (m_iterators[i] != m_columns[i].end())
08728                 ++m_iterators[i];
08729         }
08730         return *this;
08731     }
08732     auto operator ++(int) -> iterator {
08733         iterator prev(*this);
08734         operator++;
08735         return prev;
08736     }
08737 };
08738 using const_iterator = iterator;
08739
08740 auto begin() const -> iterator { return iterator(*this); }
08741 auto end() const -> iterator { return { *this, iterator::EndTag() }; }
08742
08743 auto operator += (Column const& col) -> Columns& {
08744     m_columns.push_back(col);
08745     return *this;
08746 }
08747 auto operator + (Column const& col) -> Columns {
08748     Columns combined = *this;
08749     combined += col;
08750     return combined;
08751 }
08752
08753 inline friend std::ostream& operator << (std::ostream& os, Columns const& cols) {
08754     bool first = true;
08755     for (auto line : cols) {
08756         if (first)
08757             first = false;
08758         else
08759             os << "\n";
08760         os << line;
08761     }
08762     return os;
08763 }
08764
08765 auto toString() const -> std::string {
08766     std::ostringstream oss;
08767     oss << *this;
08768     return oss.str();
08769 }
08770 };
08771 };
08772
08773 inline auto Column::operator + (Column const& other) -> Columns {
08774     Columns cols;
08775     cols += *this;

```

```

08776     cols += other;
08777     return cols;
08778 }
08779 }
08780
08781 }
08782 }
08783
08784 // ----- end of #include from clara_textflow.hpp -----
08785 // ..... back in clara.hpp
08786
08787 #include <cctype>
08788 #include <string>
08789 #include <memory>
08790 #include <set>
08791 #include <algorithm>
08792
08793 #if !defined(CATCH_PLATFORM_WINDOWS) && ( defined(WIN32) || defined(__WIN32__) || defined(_WIN32) ||
    defined(_MSC_VER) )
08794 #define CATCH_PLATFORM_WINDOWS
08795 #endif
08796
08797 namespace Catch { namespace clara {
08798 namespace detail {
08799
08800     // Traits for extracting arg and return type of lambdas (for single argument lambdas)
08801     template<typename L>
08802     struct UnaryLambdaTraits : UnaryLambdaTraits<decltype( &L::operator() )> {};
08803
08804     template<typename ClassT, typename ReturnT, typename... Args>
08805     struct UnaryLambdaTraits<ReturnT( ClassT::* )( Args... ) const> {
08806         static const bool isValid = false;
08807     };
08808
08809     template<typename ClassT, typename ReturnT, typename ArgT>
08810     struct UnaryLambdaTraits<ReturnT( ClassT::* )( ArgT ) const> {
08811         static const bool isValid = true;
08812         using ArgType = typename std::remove_const<typename std::remove_reference<ArgT>::type>::type;
08813         using ReturnT = ReturnT;
08814     };
08815
08816     class TokenStream;
08817
08818     // Transport for raw args (copied from main args, or supplied via init list for testing)
08819     class Args {
08820     friend TokenStream;
08821     std::string m_exeName;
08822     std::vector<std::string> m_args;
08823
08824     public:
08825         Args( int argc, char const* const* argv )
08826             : m_exeName(argv[0]),
08827               m_args(argv + 1, argv + argc) {}
08828
08829         Args( std::initializer_list<std::string> args )
08830             : m_exeName( *args.begin() ),
08831               m_args( args.begin()+1, args.end() )
08832         {}
08833
08834         auto exeName() const -> std::string {
08835             return m_exeName;
08836         }
08837     };
08838
08839     // Wraps a token coming from a token stream. These may not directly correspond to strings as a
08840     // single string
08841     // may encode an option + its argument if the : or = form is used
08842     enum class TokenType {
08843         Option, Argument
08844     };
08845     struct Token {
08846         TokenType type;
08847         std::string token;
08848     };
08849
08850     inline auto isOptPrefix( char c ) -> bool {
08851         return c == '-' ||
08852             #ifdef CATCH_PLATFORM_WINDOWS
08853             || c == '/'
08854             #endif
08855         ;
08856     }
08857
08858     // Abstracts iterators into args as a stream of tokens, with option arguments uniformly handled
08859     class TokenStream {
08860     using Iterator = std::vector<std::string>::const_iterator;
08861     Iterator it;

```

```

08861         Iterator itEnd;
08862         std::vector<Token> m_tokenBuffer;
08863
08864         void loadBuffer() {
08865             m_tokenBuffer.resize( 0 );
08866
08867             // Skip any empty strings
08868             while( it != itEnd && it->empty() )
08869                 ++it;
08870
08871             if( it != itEnd ) {
08872                 auto const &next = *it;
08873                 if( isOptPrefix( next[0] ) ) {
08874                     auto delimiterPos = next.find_first_of( " :=" );
08875                     if( delimiterPos != std::string::npos ) {
08876                         m_tokenBuffer.push_back( { TokenType::Option, next.substr( 0, delimiterPos ) }
08877 );
08878                         m_tokenBuffer.push_back( { TokenType::Argument, next.substr( delimiterPos + 1
08879 ) } );
08880
08881                     } else {
08882                         if( next[1] != '-' && next.size() > 2 ) {
08883                             std::string opt = "- ";
08884                             for( size_t i = 1; i < next.size(); ++i ) {
08885                                 opt[i] = next[i];
08886                                 m_tokenBuffer.push_back( { TokenType::Option, opt } );
08887                             }
08888                         } else {
08889                             m_tokenBuffer.push_back( { TokenType::Option, next } );
08890                         }
08891                     } else {
08892                         m_tokenBuffer.push_back( { TokenType::Argument, next } );
08893                     }
08894                 }
08895             }
08896         }
08897     public:
08898         explicit TokenStream( Args const &args ) : TokenStream( args.m_args.begin(), args.m_args.end()
08899 ) {}
08900
08901         TokenStream( Iterator it, Iterator itEnd ) : it( it ), itEnd( itEnd ) {
08902             loadBuffer();
08903         }
08904
08905         explicit operator bool() const {
08906             return !m_tokenBuffer.empty() || it != itEnd;
08907         }
08908
08909         auto count() const -> size_t { return m_tokenBuffer.size() + (itEnd - it); }
08910
08911         auto operator*() const -> Token {
08912             assert( !m_tokenBuffer.empty() );
08913             return m_tokenBuffer.front();
08914         }
08915
08916         auto operator->() const -> Token const * {
08917             assert( !m_tokenBuffer.empty() );
08918             return &m_tokenBuffer.front();
08919         }
08920
08921         auto operator++() -> TokenStream & {
08922             if( m_tokenBuffer.size() >= 2 ) {
08923                 m_tokenBuffer.erase( m_tokenBuffer.begin() );
08924             } else {
08925                 if( it != itEnd )
08926                     ++it;
08927                 loadBuffer();
08928             }
08929             return *this;
08930         }
08931     };
08932
08933     class ResultBase {
08934     public:
08935         enum Type {
08936             Ok, LogicError, RuntimeError
08937         };
08938
08939     protected:
08940         ResultBase( Type type ) : m_type( type ) {}
08941         virtual ~ResultBase() = default;
08942         virtual void enforceOk() const = 0;
08943         Type m_type;
08944     };

```



```

08945     template<typename T>
08946     class ResultValueBase : public ResultBase {
08947     public:
08948         auto value() const -> T const & {
08949             enforceOk();
08950             return m_value;
08951         }
08952
08953     protected:
08954         ResultValueBase( Type type ) : ResultBase( type ) {}
08955
08956         ResultValueBase( ResultValueBase const &other ) : ResultBase( other ) {
08957             if( m_type == ResultBase::Ok )
08958                 new( &m_value ) T( other.m_value );
08959         }
08960
08961         ResultValueBase( Type, T const &value ) : ResultBase( Ok ) {
08962             new( &m_value ) T( value );
08963         }
08964
08965         auto operator=( ResultValueBase const &other ) -> ResultValueBase & {
08966             if( m_type == ResultBase::Ok )
08967                 m_value.~T();
08968             ResultBase::operator=(other);
08969             if( m_type == ResultBase::Ok )
08970                 new( &m_value ) T( other.m_value );
08971             return *this;
08972         }
08973
08974         ~ResultValueBase() override {
08975             if( m_type == Ok )
08976                 m_value.~T();
08977         }
08978
08979         union {
08980             T m_value;
08981         };
08982     };
08983
08984     template<>
08985     class ResultValueBase<void> : public ResultBase {
08986     protected:
08987         using ResultBase::ResultBase;
08988     };
08989
08990     template<typename T = void>
08991     class BasicResult : public ResultValueBase<T> {
08992     public:
08993         template<typename U>
08994         explicit BasicResult( BasicResult<U> const &other )
08995             : ResultValueBase<T>( other.type() ),
08996               m_errorMessage( other.errorMessage() )
08997         {
08998             assert( type() != ResultBase::Ok );
08999         }
09000
09001         template<typename U>
09002         static auto ok( U const &value ) -> BasicResult { return { ResultBase::Ok, value }; }
09003         static auto ok() -> BasicResult { return { ResultBase::Ok }; }
09004         static auto logicError( std::string const &message ) -> BasicResult { return {
ResultBase::LogicError, message }; }
09005         static auto runtimeError( std::string const &message ) -> BasicResult { return {
ResultBase::RuntimeError, message }; }
09006
09007         explicit operator bool() const { return m_type == ResultBase::Ok; }
09008         auto type() const -> ResultBase::Type { return m_type; }
09009         auto errorMessage() const -> std::string { return m_errorMessage; }
09010
09011     protected:
09012         void enforceOk() const override {
09013
09014             // Errors shouldn't reach this point, but if they do
09015             // the actual error message will be in m_errorMessage
09016             assert( m_type != ResultBase::LogicError );
09017             assert( m_type != ResultBase::RuntimeError );
09018             if( m_type != ResultBase::Ok )
09019                 std::abort();
09020         }
09021
09022         std::string m_errorMessage; // Only populated if resultType is an error
09023
09024         BasicResult( ResultBase::Type type, std::string const &message )
09025             : ResultValueBase<T>(type),
09026               m_errorMessage(message)
09027         {
09028             assert( m_type != ResultBase::Ok );
09029         }

```

```

09030
09031     using ResultValueBase<T>::ResultValueBase;
09032     using ResultBase::m_type;
09033 };
09034
09035 enum class ParseResultType {
09036     Matched, NoMatch, ShortCircuitAll, ShortCircuitSame
09037 };
09038
09039 class ParseState {
09040 public:
09041
09042     ParseState( ParseResultType type, TokenStream const &remainingTokens )
09043     : m_type(type),
09044       m_remainingTokens( remainingTokens )
09045     {}
09046
09047     auto type() const -> ParseResultType { return m_type; }
09048     auto remainingTokens() const -> TokenStream { return m_remainingTokens; }
09049
09050 private:
09051     ParseResultType m_type;
09052     TokenStream m_remainingTokens;
09053 };
09054
09055 using Result = BasicResult<void>;
09056 using ParserResult = BasicResult<ParseResultType>;
09057 using InternalParseResult = BasicResult<ParseState>;
09058
09059 struct HelpColumns {
09060     std::string left;
09061     std::string right;
09062 };
09063
09064 template<typename T>
09065 inline auto convertInto( std::string const &source, T& target ) -> ParserResult {
09066     std::stringstream ss;
09067     ss << source;
09068     ss >> target;
09069     if( ss.fail() )
09070         return ParserResult::runtimeError( "Unable to convert '" + source + "' to destination
type" );
09071     else
09072         return ParserResult::ok( ParseResultType::Matched );
09073 }
09074 inline auto convertInto( std::string const &source, std::string& target ) -> ParserResult {
09075     target = source;
09076     return ParserResult::ok( ParseResultType::Matched );
09077 }
09078 inline auto convertInto( std::string const &source, bool &target ) -> ParserResult {
09079     std::string srcLC = source;
09080     std::transform( srcLC.begin(), srcLC.end(), srcLC.begin(), []( unsigned char c ) { return
static_cast<char>( std::tolower(c) ); } );
09081     if (srcLC == "y" || srcLC == "1" || srcLC == "true" || srcLC == "yes" || srcLC == "on")
09082         target = true;
09083     else if (srcLC == "n" || srcLC == "0" || srcLC == "false" || srcLC == "no" || srcLC == "off")
09084         target = false;
09085     else
09086         return ParserResult::runtimeError( "Expected a boolean value but did not recognise: '" +
source + "'" );
09087     return ParserResult::ok( ParseResultType::Matched );
09088 }
09089 #ifdef CLARA_CONFIG_OPTIONAL_TYPE
09090 template<typename T>
09091 inline auto convertInto( std::string const &source, CLARA_CONFIG_OPTIONAL_TYPE<T>& target ) ->
ParserResult {
09092     T temp;
09093     auto result = convertInto( source, temp );
09094     if( result )
09095         target = std::move(temp);
09096     return result;
09097 }
09098 #endif // CLARA_CONFIG_OPTIONAL_TYPE
09099
09100 struct NonCopyable {
09101     NonCopyable() = default;
09102     NonCopyable( NonCopyable const & ) = delete;
09103     NonCopyable( NonCopyable && ) = delete;
09104     NonCopyable &operator=( NonCopyable const & ) = delete;
09105     NonCopyable &operator=( NonCopyable && ) = delete;
09106 };
09107
09108 struct BoundRef : NonCopyable {
09109     virtual ~BoundRef() = default;
09110     virtual auto isContainer() const -> bool { return false; }
09111     virtual auto isFlag() const -> bool { return false; }
09112 };

```

```

09113     struct BoundValueRefBase : BoundRef {
09114         virtual auto setValue( std::string const &arg ) -> ParserResult = 0;
09115     };
09116     struct BoundFlagRefBase : BoundRef {
09117         virtual auto setFlag( bool flag ) -> ParserResult = 0;
09118         virtual auto isFlag() const -> bool { return true; }
09119     };
09120
09121     template<typename T>
09122     struct BoundValueRef : BoundValueRefBase {
09123         T &m_ref;
09124
09125         explicit BoundValueRef( T &ref ) : m_ref( ref ) {}
09126
09127         auto setValue( std::string const &arg ) -> ParserResult override {
09128             return convertInto( arg, m_ref );
09129         }
09130     };
09131
09132     template<typename T>
09133     struct BoundValueRef<std::vector<T> > : BoundValueRefBase {
09134         std::vector<T> &m_ref;
09135
09136         explicit BoundValueRef( std::vector<T> &ref ) : m_ref( ref ) {}
09137
09138         auto isContainer() const -> bool override { return true; }
09139
09140         auto setValue( std::string const &arg ) -> ParserResult override {
09141             T temp;
09142             auto result = convertInto( arg, temp );
09143             if( result )
09144                 m_ref.push_back( temp );
09145             return result;
09146         }
09147     };
09148
09149     struct BoundFlagRef : BoundFlagRefBase {
09150         bool &m_ref;
09151
09152         explicit BoundFlagRef( bool &ref ) : m_ref( ref ) {}
09153
09154         auto setFlag( bool flag ) -> ParserResult override {
09155             m_ref = flag;
09156             return ParserResult::ok( ParseResultType::Matched );
09157         }
09158     };
09159
09160     template<typename ReturnType>
09161     struct LambdaInvoker {
09162         static_assert( std::is_same<ReturnType, ParserResult>::value, "Lambda must return void or
09163         clara::ParserResult" );
09164
09165         template<typename L, typename ArgType>
09166         static auto invoke( L const &lambda, ArgType const &arg ) -> ParserResult {
09167             return lambda( arg );
09168         }
09169     };
09170
09171     template<>
09172     struct LambdaInvoker<void> {
09173         template<typename L, typename ArgType>
09174         static auto invoke( L const &lambda, ArgType const &arg ) -> ParserResult {
09175             lambda( arg );
09176             return ParserResult::ok( ParseResultType::Matched );
09177         }
09178     };
09179
09180     template<typename ArgType, typename L>
09181     inline auto invokeLambda( L const &lambda, std::string const &arg ) -> ParserResult {
09182         ArgType temp{};
09183         auto result = convertInto( arg, temp );
09184         return !result
09185             ? result
09186             : LambdaInvoker<typename UnaryLambdaTraits<L>::ReturnType>::invoke( lambda, temp );
09187     }
09188
09189     template<typename L>
09190     struct BoundLambda : BoundValueRefBase {
09191         L m_lambda;
09192
09193         static_assert( UnaryLambdaTraits<L>::isValid, "Supplied lambda must take exactly one argument"
09194     );
09195         explicit BoundLambda( L const &lambda ) : m_lambda( lambda ) {}
09196
09197         auto setValue( std::string const &arg ) -> ParserResult override {
09198             return invokeLambda<typename UnaryLambdaTraits<L>::ArgType>( m_lambda, arg );
09199         }
09200     };

```

```

09198     };
09199
09200     template<typename L>
09201     struct BoundFlagLambda : BoundFlagRefBase {
09202         L m_lambda;
09203
09204         static_assert( UnaryLambdaTraits<L>::isValid, "Supplied lambda must take exactly one argument"
09205 );
09206         static_assert( std::is_same<typename UnaryLambdaTraits<L>::ArgType, bool>::value, "flags must
be boolean" );
09207
09208         explicit BoundFlagLambda( L const &lambda ) : m_lambda( lambda ) {}
09209
09210         auto setFlag( bool flag ) -> ParserResult override {
09211             return LambdaInvoker<typename UnaryLambdaTraits<L>::ReturnType>::invoke( m_lambda, flag );
09212         }
09213     };
09214
09215     enum class Optionality { Optional, Required };
09216
09217     struct Parser;
09218
09219     class ParserBase {
09220     public:
09221         virtual ~ParserBase() = default;
09222         virtual auto validate() const -> Result { return Result::ok(); }
09223         virtual auto parse( std::string const& exeName, TokenStream const &tokens) const ->
InternalParseResult = 0;
09224         virtual auto cardinality() const -> size_t { return 1; }
09225
09226         auto parse( Args const &args ) const -> InternalParseResult {
09227             return parse( args.exeName(), TokenStream( args ) );
09228         }
09229     };
09230
09231     template<typename DerivedT>
09232     class ComposableParserImpl : public ParserBase {
09233     public:
09234         template<typename T>
09235         auto operator|( T const &other ) const -> Parser;
09236
09237         template<typename T>
09238         auto operator+( T const &other ) const -> Parser;
09239     };
09240
09241     // Common code and state for Args and Opts
09242     template<typename DerivedT>
09243     class ParserRefImpl : public ComposableParserImpl<DerivedT> {
09244     protected:
09245         Optionality m_optionality = Optionality::Optional;
09246         std::shared_ptr<BoundRef> m_ref;
09247         std::string m_hint;
09248         std::string m_description;
09249
09250         explicit ParserRefImpl( std::shared_ptr<BoundRef> const &ref ) : m_ref( ref ) {}
09251     public:
09252         template<typename T>
09253         ParserRefImpl( T &ref, std::string const &hint )
09254             : m_ref( std::make_shared<BoundValueRef<T>( ref ) ),
09255               m_hint( hint )
09256         {}
09257
09258         template<typename LambdaT>
09259         ParserRefImpl( LambdaT const &ref, std::string const &hint )
09260             : m_ref( std::make_shared<BoundLambda<LambdaT>( ref ) ),
09261               m_hint( hint )
09262         {}
09263
09264         auto operator()( std::string const &description ) -> DerivedT & {
09265             m_description = description;
09266             return static_cast<DerivedT &>( *this );
09267         }
09268
09269         auto optional() -> DerivedT & {
09270             m_optionality = Optionality::Optional;
09271             return static_cast<DerivedT &>( *this );
09272         };
09273
09274         auto required() -> DerivedT & {
09275             m_optionality = Optionality::Required;
09276             return static_cast<DerivedT &>( *this );
09277         };
09278
09279         auto isOptional() const -> bool {
09280             return m_optionality == Optionality::Optional;
09281         }

```

```

09282
09283     auto cardinality() const -> size_t override {
09284         if( m_ref->isContainer() )
09285             return 0;
09286         else
09287             return 1;
09288     }
09289
09290     auto hint() const -> std::string { return m_hint; }
09291 };
09292
09293 class ExeName : public ComposableParserImpl<ExeName> {
09294     std::shared_ptr<std::string> m_name;
09295     std::shared_ptr<BoundValueRefBase> m_ref;
09296
09297     template<typename LambdaT>
09298     static auto makeRef(LambdaT const& lambda) -> std::shared_ptr<BoundValueRefBase> {
09299         return std::make_shared<BoundLambda<LambdaT>>( lambda );
09300     }
09301
09302 public:
09303     ExeName() : m_name( std::make_shared<std::string>( "<executable>" ) ) {}
09304
09305     explicit ExeName( std::string& ref ) : ExeName() {
09306         m_ref = std::make_shared<BoundValueRef<std::string>>( ref );
09307     }
09308
09309     template<typename LambdaT>
09310     explicit ExeName( LambdaT const& lambda ) : ExeName() {
09311         m_ref = std::make_shared<BoundLambda<LambdaT>>( lambda );
09312     }
09313
09314     // The exe name is not parsed out of the normal tokens, but is handled specially
09315     auto parse( std::string const&, TokenStream const& tokens ) const -> InternalParseResult
09316 override {
09317         return InternalParseResult::ok( ParseState( ParseResultType::NoMatch, tokens ) );
09318     }
09319
09320     auto name() const -> std::string { return *m_name; }
09321     auto set( std::string const& newName ) -> ParserResult {
09322         auto lastSlash = newName.find_last_of( "\\/" );
09323         auto filename = ( lastSlash == std::string::npos )
09324             ? newName
09325             : newName.substr( lastSlash+1 );
09326
09327         *m_name = filename;
09328         if( m_ref )
09329             return m_ref->setValue( filename );
09330         else
09331             return ParserResult::ok( ParseResultType::Matched );
09332     }
09333 };
09334
09335 class Arg : public ParserRefImpl<Arg> {
09336 public:
09337     using ParserRefImpl::ParserRefImpl;
09338
09339     auto parse( std::string const&, TokenStream const& tokens ) const -> InternalParseResult
09340 override {
09341         auto validationResult = validate();
09342         if( !validationResult )
09343             return InternalParseResult( validationResult );
09344
09345         auto remainingTokens = tokens;
09346         auto const& token = *remainingTokens;
09347         if( token.type != TokenType::Argument )
09348             return InternalParseResult::ok( ParseState( ParseResultType::NoMatch, remainingTokens ) );
09349
09350         assert( !m_ref->isFlag() );
09351         auto valueRef = static_cast<detail::BoundValueRefBase*>( m_ref.get() );
09352
09353         auto result = valueRef->setValue( remainingTokens->token );
09354         if( !result )
09355             return InternalParseResult( result );
09356         else
09357             return InternalParseResult::ok( ParseState( ParseResultType::Matched,
09358                 ++remainingTokens ) );
09359     }
09360 };
09361
09362 inline auto normaliseOpt( std::string const& optName ) -> std::string {
09363     #ifdef CATCH_PLATFORM_WINDOWS
09364         if( optName[0] == '/' )
09365             return "-" + optName.substr( 1 );
09366         else

```

```

09365 #endif
09366         return optName;
09367     }
09368
09369     class Opt : public ParserRefImpl<Opt> {
09370     protected:
09371         std::vector<std::string> m_optNames;
09372
09373     public:
09374         template<typename LambdaT>
09375         explicit Opt( LambdaT const &ref ) : ParserRefImpl( std::make_shared<BoundFlagLambda<LambdaT>(
09376             ref ) ) {}
09377
09378         explicit Opt( bool &ref ) : ParserRefImpl( std::make_shared<BoundFlagRef>( ref ) ) {}
09379
09380         template<typename LambdaT>
09381         Opt( LambdaT const &ref, std::string const &hint ) : ParserRefImpl( ref, hint ) {}
09382
09383         template<typename T>
09384         Opt( T &ref, std::string const &hint ) : ParserRefImpl( ref, hint ) {}
09385
09386         auto operator[]( std::string const &optName ) -> Opt & {
09387             m_optNames.push_back( optName );
09388             return *this;
09389         }
09390
09391         auto getHelpColumns() const -> std::vector<HelpColumns> {
09392             std::ostringstream oss;
09393             bool first = true;
09394             for( auto const &opt : m_optNames ) {
09395                 if (first)
09396                     first = false;
09397                 else
09398                     oss << ", ";
09399                 oss << opt;
09400             }
09401             if( !m_hint.empty() )
09402                 oss << " <" << m_hint << ">";
09403             return { { oss.str(), m_description } };
09404         }
09405
09406         auto isMatch( std::string const &optToken ) const -> bool {
09407             auto normalisedToken = normaliseOpt( optToken );
09408             for( auto const &name : m_optNames ) {
09409                 if( normaliseOpt( name ) == normalisedToken )
09410                     return true;
09411             }
09412             return false;
09413         }
09414
09415         using ParserBase::parse;
09416
09417         auto parse( std::string const&, TokenStream const &tokens ) const -> InternalParseResult
09418         override {
09419             auto validationResult = validate();
09420             if( !validationResult )
09421                 return InternalParseResult( validationResult );
09422
09423             auto remainingTokens = tokens;
09424             if( remainingTokens && remainingTokens->type == TokenType::Option ) {
09425                 auto const &token = *remainingTokens;
09426                 if( isMatch(token.token) ) {
09427                     if( m_ref->isFlag() ) {
09428                         auto flagRef = static_cast<detail::BoundFlagRefBase*>( m_ref.get() );
09429                         auto result = flagRef->setFlag( true );
09430                         if( !result )
09431                             return InternalParseResult( result );
09432                         if( result.value() == ParseResultType::ShortCircuitAll )
09433                             return InternalParseResult::ok( ParseState( result.value(),
09434                                 remainingTokens ) );
09435                     } else {
09436                         auto valueRef = static_cast<detail::BoundValueRefBase*>( m_ref.get() );
09437                         ++remainingTokens;
09438                         if( !remainingTokens )
09439                             return InternalParseResult::runtimeError( "Expected argument following " +
09440                                 token.token );
09441                         auto const &argToken = *remainingTokens;
09442                         if( argToken.type != TokenType::Argument )
09443                             return InternalParseResult::runtimeError( "Expected argument following " +
09444                                 token.token );
09445                         auto result = valueRef->setValue( argToken.token );
09446                         if( !result )
09447                             return InternalParseResult( result );
09448                         if( result.value() == ParseResultType::ShortCircuitAll )
09449                             return InternalParseResult::ok( ParseState( result.value(),
09450                                 remainingTokens ) );
09451                     }
09452                 }
09453             }
09454         }

```

```

09446         return InternalParseResult::ok( ParseState( ParseResultType::Matched,
09447             ++remainingTokens ) );
09448     }
09449     return InternalParseResult::ok( ParseState( ParseResultType::NoMatch, remainingTokens ) );
09450 }
09451
09452 auto validate() const -> Result override {
09453     if( m_optNames.empty() )
09454         return Result::logicError( "No options supplied to Opt" );
09455     for( auto const &name : m_optNames ) {
09456         if( name.empty() )
09457             return Result::logicError( "Option name cannot be empty" );
09458 #ifdef CATCH_PLATFORM_WINDOWS
09459         if( name[0] != '-' && name[0] != '/' )
09460             return Result::logicError( "Option name must begin with '-' or '/'" );
09461 #else
09462         if( name[0] != '-' )
09463             return Result::logicError( "Option name must begin with '-'" );
09464 #endif
09465     }
09466     return ParserRefImpl::validate();
09467 }
09468 };
09469
09470 struct Help : Opt {
09471     Help( bool &showHelpFlag )
09472     : Opt([&]( bool flag ) {
09473         showHelpFlag = flag;
09474         return ParserResult::ok( ParseResultType::ShortCircuitAll );
09475     })
09476     {
09477         static_cast<Opt &>( *this )
09478             ("display usage information")
09479             ["-?"]["-h"]["--help"]
09480             .optional();
09481     }
09482 };
09483
09484 struct Parser : ParserBase {
09485     mutable ExeName m_exeName;
09486     std::vector<Opt> m_options;
09487     std::vector<Arg> m_args;
09488
09489     auto operator|=( ExeName const &exeName ) -> Parser & {
09490         m_exeName = exeName;
09491         return *this;
09492     }
09493 }
09494
09495 auto operator|=( Arg const &arg ) -> Parser & {
09496     m_args.push_back(arg);
09497     return *this;
09498 }
09499
09500 auto operator|=( Opt const &opt ) -> Parser & {
09501     m_options.push_back(opt);
09502     return *this;
09503 }
09504
09505 auto operator|=( Parser const &other ) -> Parser & {
09506     m_options.insert(m_options.end(), other.m_options.begin(), other.m_options.end());
09507     m_args.insert(m_args.end(), other.m_args.begin(), other.m_args.end());
09508     return *this;
09509 }
09510
09511 template<typename T>
09512 auto operator|( T const &other ) const -> Parser {
09513     return Parser( *this ) |= other;
09514 }
09515
09516 // Forward deprecated interface with '+' instead of '|'
09517 template<typename T>
09518 auto operator+=( T const &other ) -> Parser & { return operator|=( other ); }
09519 template<typename T>
09520 auto operator+( T const &other ) const -> Parser { return operator|( other ); }
09521
09522 auto getHelpColumns() const -> std::vector<HelpColumns> {
09523     std::vector<HelpColumns> cols;
09524     for( auto const &o : m_options ) {
09525         auto childCols = o.getHelpColumns();
09526         cols.insert( cols.end(), childCols.begin(), childCols.end() );
09527     }
09528     return cols;
09529 }
09530
09531 void writeToStream( std::ostream &os ) const {

```

```

09532         if (!m_exeName.name().empty()) {
09533             os << "usage:\n" << " " << m_exeName.name() << " ";
09534             bool required = true, first = true;
09535             for( auto const &arg : m_args ) {
09536                 if (first)
09537                     first = false;
09538                 else
09539                     os << " ";
09540                 if( arg.isOptional() && required ) {
09541                     os << "[";
09542                     required = false;
09543                 }
09544                 os << "<" << arg.hint() << ">";
09545                 if( arg.cardinality() == 0 )
09546                     os << " ... ";
09547             }
09548             if( !required )
09549                 os << "]";
09550             if( !m_options.empty() )
09551                 os << " options";
09552             os << "\n\nwhere options are:" << std::endl;
09553         }
09554
09555         auto rows = getHelpColumns();
09556         size_t consoleWidth = CATCH_CLARA_CONFIG_CONSOLE_WIDTH;
09557         size_t optWidth = 0;
09558         for( auto const &cols : rows )
09559             optWidth = (std::max)(optWidth, cols.left.size() + 2);
09560
09561         optWidth = (std::min)(optWidth, consoleWidth/2);
09562
09563         for( auto const &cols : rows ) {
09564             auto row =
09565                 TextFlow::Column( cols.left ).width( optWidth ).indent( 2 ) +
09566                 TextFlow::Spacer(4) +
09567                 TextFlow::Column( cols.right ).width( consoleWidth - 7 - optWidth );
09568             os << row << std::endl;
09569         }
09570     }
09571
09572     friend auto operator<<( std::ostream &os, Parser const &parser ) -> std::ostream& {
09573         parser.writeToStream( os );
09574         return os;
09575     }
09576
09577     auto validate() const -> Result override {
09578         for( auto const &opt : m_options ) {
09579             auto result = opt.validate();
09580             if( !result )
09581                 return result;
09582         }
09583         for( auto const &arg : m_args ) {
09584             auto result = arg.validate();
09585             if( !result )
09586                 return result;
09587         }
09588         return Result::ok();
09589     }
09590
09591     using ParserBase::parse;
09592
09593     auto parse( std::string const& exeName, TokenStream const &tokens ) const ->
InternalParseResult override {
09594
09595         struct ParserInfo {
09596             ParserBase const* parser = nullptr;
09597             size_t count = 0;
09598         };
09599         const size_t totalParsers = m_options.size() + m_args.size();
09600         assert( totalParsers < 512 );
09601         // ParserInfo parseInfos[totalParsers]; // <-- this is what we really want to do
09602         ParserInfo parseInfos[512];
09603
09604         {
09605             size_t i = 0;
09606             for (auto const &opt : m_options) parseInfos[i++].parser = &opt;
09607             for (auto const &arg : m_args) parseInfos[i++].parser = &arg;
09608         }
09609
09610         m_exeName.set( exeName );
09611
09612         auto result = InternalParseResult::ok( ParseState( ParseResultType::NoMatch, tokens ) );
09613         while( result.value().remainingTokens() ) {
09614             bool tokenParsed = false;
09615
09616             for( size_t i = 0; i < totalParsers; ++i ) {
09617                 auto& parseInfo = parseInfos[i];

```



```

09618         if( parseInfo.parser->cardinality() == 0 || parseInfo.count <
    parseInfo.parser->cardinality() ) {
09619             result = parseInfo.parser->parse(exeName, result.value().remainingTokens());
09620             if (!result)
09621                 return result;
09622             if (result.value().type() != ParseResultType::NoMatch) {
09623                 tokenParsed = true;
09624                 ++parseInfo.count;
09625                 break;
09626             }
09627         }
09628     }
09629
09630     if( result.value().type() == ParseResultType::ShortCircuitAll )
09631         return result;
09632     if( !tokenParsed )
09633         return InternalParseResult::runtimeError( "Unrecognised token: " +
    result.value().remainingTokens()->token );
09634     }
09635     // !TBD Check missing required options
09636     return result;
09637 }
09638 };
09639
09640 template<typename DerivedT>
09641 template<typename T>
09642 auto ComposableParserImpl<DerivedT>::operator|( T const &other ) const -> Parser {
09643     return Parser() | static_cast<DerivedT const &>( *this ) | other;
09644 }
09645 } // namespace detail
09646
09647 // A Combined parser
09648 using detail::Parser;
09649
09650 // A parser for options
09651 using detail::Opt;
09652
09653 // A parser for arguments
09654 using detail::Arg;
09655
09656 // Wrapper for argc, argv from main()
09657 using detail::Args;
09658
09659 // Specifies the name of the executable
09660 using detail::ExeName;
09661
09662 // Convenience wrapper for option parser that specifies the help option
09663 using detail::Help;
09664
09665 // enum of result types from a parse
09666 using detail::ParseResultType;
09667
09668 // Result type for parser operation
09669 using detail::ParserResult;
09670
09671 }} // namespace Catch::clara
09672
09673 // end clara.hpp
09674 #ifdef __clang__
09675 #pragma clang diagnostic pop
09676 #endif
09677
09678 // Restore Clara's value for console width, if present
09679 #ifdef CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
09680 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
09681 #undef CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
09682 #endif
09683
09684 // end catch_clara.h
09685 namespace Catch {
09686
09687     clara::Parser makeCommandLineParser( ConfigData& config );
09688
09689 } // end namespace Catch
09690
09691 // end catch_commandline.h
09692 #include <fstream>
09693 #include <ctime>
09694
09695 namespace Catch {
09696
09697     clara::Parser makeCommandLineParser( ConfigData& config ) {
09698
09699         using namespace clara;
09700
09701         auto const setWarning = [&]( std::string const& warning ) {
09702             auto warningSet = [&]() {

```

```

09703         if( warning == "NoAssertions" )
09704             return WarnAbout::NoAssertions;
09705
09706         if ( warning == "NoTests" )
09707             return WarnAbout::NoTests;
09708
09709         return WarnAbout::Nothing;
09710     }();
09711
09712     if (warningSet == WarnAbout::Nothing)
09713         return ParserResult::runtimeError( "Unrecognised warning: '" + warning + "'" );
09714     config.warnings = static_cast<WarnAbout::What>( config.warnings | warningSet );
09715     return ParserResult::ok( ParseResultType::Matched );
09716 };
09717 auto const loadTestNamesFromFile = [&]( std::string const& filename ) {
09718     std::ifstream f( filename.c_str() );
09719     if( !f.is_open() )
09720         return ParserResult::runtimeError( "Unable to load input file: '" + filename + "'"
);
09721
09722     std::string line;
09723     while( std::getline( f, line ) ) {
09724         line = trim(line);
09725         if( !line.empty() && !startsWith( line, '#' ) ) {
09726             if( !startsWith( line, '"' ) )
09727                 line = '"' + line + '"';
09728             config.testsOrTags.push_back( line );
09729             config.testsOrTags.emplace_back( "," );
09730         }
09731     }
09732     //Remove comma in the end
09733     if(!config.testsOrTags.empty())
09734         config.testsOrTags.erase( config.testsOrTags.end()-1 );
09735
09736     return ParserResult::ok( ParseResultType::Matched );
09737 };
09738 auto const setTestOrder = [&]( std::string const& order ) {
09739     if( startsWith( "declared", order ) )
09740         config.runOrder = RunTests::InDeclarationOrder;
09741     else if( startsWith( "lexical", order ) )
09742         config.runOrder = RunTests::InLexicographicalOrder;
09743     else if( startsWith( "random", order ) )
09744         config.runOrder = RunTests::InRandomOrder;
09745     else
09746         return clara::ParserResult::runtimeError( "Unrecognised ordering: '" + order + "'"
);
09747     return ParserResult::ok( ParseResultType::Matched );
09748 };
09749 auto const setRngSeed = [&]( std::string const& seed ) {
09750     if( seed != "time" )
09751         return clara::detail::convertInto( seed, config.rngSeed );
09752     config.rngSeed = static_cast<unsigned int>( std::time(nullptr) );
09753     return ParserResult::ok( ParseResultType::Matched );
09754 };
09755 auto const setColourUsage = [&]( std::string const& useColour ) {
09756     auto mode = toLower( useColour );
09757
09758     if( mode == "yes" )
09759         config.useColour = UseColour::Yes;
09760     else if( mode == "no" )
09761         config.useColour = UseColour::No;
09762     else if( mode == "auto" )
09763         config.useColour = UseColour::Auto;
09764     else
09765         return clara::ParserResult::runtimeError( "colour mode must be one of: auto, yes or
no. '" + useColour + "' not recognised" );
09766     return ParserResult::ok( ParseResultType::Matched );
09767 };
09768 auto const setWaitForKeypress = [&]( std::string const& keypress ) {
09769     auto keypressLc = toLower( keypress );
09770     if( keypressLc == "never" )
09771         config.waitForKeypress = WaitForKeypress::Never;
09772     else if( keypressLc == "start" )
09773         config.waitForKeypress = WaitForKeypress::BeforeStart;
09774     else if( keypressLc == "exit" )
09775         config.waitForKeypress = WaitForKeypress::BeforeExit;
09776     else if( keypressLc == "both" )
09777         config.waitForKeypress = WaitForKeypress::BeforeStartAndExit;
09778     else
09779         return clara::ParserResult::runtimeError( "keypress argument must be one of: never,
start, exit or both. '" + keypress + "' not recognised" );
09780     return ParserResult::ok( ParseResultType::Matched );
09781 };
09782 auto const setVerbosity = [&]( std::string const& verbosity ) {
09783     auto lcVerbosity = toLower( verbosity );
09784     if( lcVerbosity == "quiet" )
09785         config.verbosity = Verbosity::Quiet;

```

```

09786         else if( lcVerbosity == "normal" )
09787             config.verbosity = Verbosity::Normal;
09788         else if( lcVerbosity == "high" )
09789             config.verbosity = Verbosity::High;
09790         else
09791             return ParserResult::runtimeError( "Unrecognised verbosity, '" + verbosity + "'" );
09792         return ParserResult::ok( ParseResultType::Matched );
09793     };
09794     auto const setReporter = [&]( std::string const& reporter ) {
09795         IReporterRegistry::FactoryMap const& factories =
09796             getRegistryHub().getReporterRegistry().getFactories();
09797         auto lcReporter = toLower( reporter );
09798         auto result = factories.find( lcReporter );
09799
09800         if( factories.end() != result )
09801             config.reporterName = lcReporter;
09802         else
09803             return ParserResult::runtimeError( "Unrecognized reporter, '" + reporter + "'. Check
available with --list-reporters" );
09804         return ParserResult::ok( ParseResultType::Matched );
09805     };
09806
09807     auto cli
09808         = ExeName( config.processName )
09809         | Help( config.showHelp )
09810         | Opt( config.listTests )
09811             [ "-l" ][ "--list-tests" ]
09812             ( "list all/matching test cases" )
09813         | Opt( config.listTags )
09814             [ "-t" ][ "--list-tags" ]
09815             ( "list all/matching tags" )
09816         | Opt( config.showSuccessfulTests )
09817             [ "-s" ][ "--success" ]
09818             ( "include successful tests in output" )
09819         | Opt( config.shouldDebugBreak )
09820             [ "-b" ][ "--break" ]
09821             ( "break into debugger on failure" )
09822         | Opt( config.noThrow )
09823             [ "-e" ][ "--nothrow" ]
09824             ( "skip exception tests" )
09825         | Opt( config.showInvisibles )
09826             [ "-i" ][ "--invisibles" ]
09827             ( "show invisibles (tabs, newlines)" )
09828         | Opt( config.outputFilename, "filename" )
09829             [ "-o" ][ "--out" ]
09830             ( "output filename" )
09831         | Opt( setReporter, "name" )
09832             [ "-r" ][ "--reporter" ]
09833             ( "reporter to use (defaults to console)" )
09834         | Opt( config.name, "name" )
09835             [ "-n" ][ "--name" ]
09836             ( "suite name" )
09837         | Opt( [&]( bool ){ config.abortAfter = 1; } )
09838             [ "-a" ][ "--abort" ]
09839             ( "abort at first failure" )
09840         | Opt( [&]( int x ){ config.abortAfter = x; }, "no. failures" )
09841             [ "-x" ][ "--abortx" ]
09842             ( "abort after x failures" )
09843         | Opt( setWarning, "warning name" )
09844             [ "-w" ][ "--warn" ]
09845             ( "enable warnings" )
09846         | Opt( [&]( bool flag ) { config.showDurations = flag ? ShowDurations::Always :
ShowDurations::Never; }, "yes|no" )
09847             [ "-d" ][ "--durations" ]
09848             ( "show test durations" )
09849         | Opt( config.minDuration, "seconds" )
09850             [ "-D" ][ "--min-duration" ]
09851             ( "show test durations for tests taking at least the given number of seconds" )
09852         | Opt( loadTestNamesFromFile, "filename" )
09853             [ "-f" ][ "--input-file" ]
09854             ( "load test names to run from a file" )
09855         | Opt( config.fileNamesAsTags )
09856             [ "-#" ][ "--filenames-as-tags" ]
09857             ( "adds a tag for the filename" )
09858         | Opt( config.sectionsToRun, "section name" )
09859             [ "-c" ][ "--section" ]
09860             ( "specify section to run" )
09861         | Opt( setVerbosity, "quiet|normal|high" )
09862             [ "-v" ][ "--verbosity" ]
09863             ( "set output verbosity" )
09864         | Opt( config.listTestNamesOnly )
09865             [ "--list-test-names-only" ]
09866             ( "list all/matching test cases names only" )
09867         | Opt( config.listReporters )
09868             [ "--list-reporters" ]
09869             ( "list all reporters" )

```

```

09870         | Opt( setTestOrder, "decl|lex|rand" )
09871         | [ "--order" ]
09872         | ( "test case order (defaults to decl)" )
09873         | Opt( setRngSeed, "'time'|number" )
09874         | [ "--rng-seed" ]
09875         | ( "set a specific seed for random numbers" )
09876         | Opt( setColourUsage, "yes|no" )
09877         | [ "--use-colour" ]
09878         | ( "should output be colourised" )
09879         | Opt( config.libIdentify )
09880         | [ "--libidentify" ]
09881         | ( "report name and version according to libidentify standard" )
09882         | Opt( setWaitForKeypress, "never|start|exit|both" )
09883         | [ "--wait-for-keypress" ]
09884         | ( "waits for a keypress before exiting" )
09885         | Opt( config.benchmarkSamples, "samples" )
09886         | [ "--benchmark-samples" ]
09887         | ( "number of samples to collect (default: 100)" )
09888         | Opt( config.benchmarkResamples, "resamples" )
09889         | [ "--benchmark-resamples" ]
09890         | ( "number of resamples for the bootstrap (default: 100000)" )
09891         | Opt( config.benchmarkConfidenceInterval, "confidence interval" )
09892         | [ "--benchmark-confidence-interval" ]
09893         | ( "confidence interval for the bootstrap (between 0 and 1, default: 0.95)" )
09894         | Opt( config.benchmarkNoAnalysis )
09895         | [ "--benchmark-no-analysis" ]
09896         | ( "perform only measurements; do not perform any analysis" )
09897         | Opt( config.benchmarkWarmupTime, "benchmarkWarmupTime" )
09898         | [ "--benchmark-warmup-time" ]
09899         | ( "amount of time in milliseconds spent on warming up each test (default: 100)" )
09900         | Arg( config.testsOrTags, "test name|pattern|tags" )
09901         | ( "which test or tests to use" );
09902
09903     return cli;
09904 }
09905
09906 } // end namespace Catch
09907 // end catch_commandline.cpp
09908 // start catch_common.cpp
09909
09910 #include <cstring>
09911 #include <ostream>
09912
09913 namespace Catch {
09914
09915     bool SourceLineInfo::operator == ( SourceLineInfo const& other ) const noexcept {
09916         return line == other.line && (file == other.file || std::strcmp(file, other.file) == 0);
09917     }
09918     bool SourceLineInfo::operator < ( SourceLineInfo const& other ) const noexcept {
09919         // We can assume that the same file will usually have the same pointer.
09920         // Thus, if the pointers are the same, there is no point in calling the strcmp
09921         return line < other.line || ( line == other.line && file != other.file && (std::strcmp(file,
09922         other.file) < 0));
09923     }
09924
09925     std::ostream& operator << ( std::ostream& os, SourceLineInfo const& info ) {
09926 #ifndef __GNUG__
09927         os << info.file << ' (' << info.line << ')';
09928 #else
09929         os << info.file << ':' << info.line;
09930 #endif
09931         return os;
09932     }
09933
09934     std::string StreamEndStop::operator+() const {
09935         return std::string();
09936     }
09937
09938     NonCopyable::NonCopyable() = default;
09939     NonCopyable::~NonCopyable() = default;
09940 }
09941 // end catch_common.cpp
09942 // start catch_config.cpp
09943
09944 namespace Catch {
09945
09946     Config::Config( ConfigData const& data )
09947     :   m_data( data ),
09948         m_stream( openStream() )
09949     {
09950         // We need to trim filter specs to avoid trouble with superfluous
09951         // whitespace (esp. important for bdd macros, as those are manually
09952         // aligned with whitespace).
09953         for (auto& elem : m_data.testsOrTags) {
09954             elem = trim(elem);
09955         }
09956     }

```

```

09956     }
09957     for (auto& elem : m_data.sectionsToRun) {
09958         elem = trim(elem);
09959     }
09960
09961     TestSpecParser parser(ITagAliasRegistry::get());
09962     if (!m_data.testsOrTags.empty()) {
09963         m_hasTestFilters = true;
09964         for (auto const& testOrTags : m_data.testsOrTags) {
09965             parser.parse(testOrTags);
09966         }
09967     }
09968     m_testSpec = parser.testSpec();
09969 }
09970
09971 std::string const& Config::getFilename() const {
09972     return m_data.outputFilename;
09973 }
09974
09975 bool Config::listTests() const { return m_data.listTests; }
09976 bool Config::listTestNamesOnly() const { return m_data.listTestNamesOnly; }
09977 bool Config::listTags() const { return m_data.listTags; }
09978 bool Config::listReporters() const { return m_data.listReporters; }
09979
09980 std::string Config::getProcessName() const { return m_data.processName; }
09981 std::string const& Config::getReporterName() const { return m_data.reporterName; }
09982
09983 std::vector<std::string> const& Config::getTestsOrTags() const { return m_data.testsOrTags; }
09984 std::vector<std::string> const& Config::getSectionsToRun() const { return m_data.sectionsToRun; }
09985
09986 TestSpec const& Config::testSpec() const { return m_testSpec; }
09987 bool Config::hasTestFilters() const { return m_hasTestFilters; }
09988
09989 bool Config::showHelp() const { return m_data.showHelp; }
09990
09991 // IConfig interface
09992 bool Config::allowThrows() const { return !m_data.noThrow; }
09993 std::ostream& Config::stream() const { return m_stream->stream(); }
09994 std::string Config::name() const { return m_data.name.empty() ?
m_data.processName : m_data.name; }
09995 bool Config::includeSuccessfulResults() const { return m_data.showSuccessfulTests; }
09996 bool Config::warnAboutMissingAssertions() const { return !(m_data.warnings &
WarnAbout::NoAssertions); }
09997 bool Config::warnAboutNoTests() const { return !(m_data.warnings &
WarnAbout::NoTests); }
09998 ShowDurations::OrNot Config::showDurations() const { return m_data.showDurations; }
09999 double Config::minDuration() const { return m_data.minDuration; }
10000 RunTests::InWhatOrder Config::runOrder() const { return m_data.runOrder; }
10001 unsigned int Config::rngSeed() const { return m_data.rngSeed; }
10002 UseColour::YesOrNo Config::useColour() const { return m_data.useColour; }
10003 bool Config::shouldDebugBreak() const { return m_data.shouldDebugBreak; }
10004 int Config::abortAfter() const { return m_data.abortAfter; }
10005 bool Config::showInvisibles() const { return m_data.showInvisibles; }
10006 Verbosity Config::verbosity() const { return m_data.verbosity; }
10007
10008 bool Config::benchmarkNoAnalysis() const { return m_data.benchmarkNoAnalysis; }
10009 int Config::benchmarkSamples() const { return m_data.benchmarkSamples; }
10010 double Config::benchmarkConfidenceInterval() const { return
m_data.benchmarkConfidenceInterval; }
10011 unsigned int Config::benchmarkResamples() const { return m_data.benchmarkResamples; }
10012
10013 std::chrono::milliseconds Config::benchmarkWarmupTime() const { return
std::chrono::milliseconds(m_data.benchmarkWarmupTime); }
10014
10015 IStream const* Config::openStream() {
10016     return Catch::makeStream(m_data.outputFilename);
10017 }
10018 } // end namespace Catch
10019 // end catch_config.cpp
10020 // start catch_console_colour.cpp
10021
10022 #if defined(__clang__)
10023 #    pragma clang diagnostic push
10024 #    pragma clang diagnostic ignored "-Wexit-time-destructors"
10025 #endif
10026
10027 // start catch_errno_guard.h
10028
10029 namespace Catch {
10030
10031     class ErrnoGuard {
10032     public:
10033         ErrnoGuard();
10034         ~ErrnoGuard();
10035     private:

```

```

10036         int m_oldErrno;
10037     };
10038
10039 }
10040
10041 // end catch_errno_guard.h
10042 // start catch_windows_h_proxy.h
10043
10044
10045 #if defined(CATCH_PLATFORM_WINDOWS)
10046
10047 #if !defined(NOMINMAX) && !defined(CATCH_CONFIG_NO_NOMINMAX)
10048 #    define CATCH_DEFINED_NOMINMAX
10049 #    define NOMINMAX
10050 #endif
10051 #if !defined(WIN32_LEAN_AND_MEAN) && !defined(CATCH_CONFIG_NO_WIN32_LEAN_AND_MEAN)
10052 #    define CATCH_DEFINED_WIN32_LEAN_AND_MEAN
10053 #    define WIN32_LEAN_AND_MEAN
10054 #endif
10055
10056 #ifdef __AFXDLL
10057 #include <AfxWin.h>
10058 #else
10059 #include <windows.h>
10060 #endif
10061
10062 #ifdef CATCH_DEFINED_NOMINMAX
10063 #    undef NOMINMAX
10064 #endif
10065 #ifdef CATCH_DEFINED_WIN32_LEAN_AND_MEAN
10066 #    undef WIN32_LEAN_AND_MEAN
10067 #endif
10068
10069 #endif // defined(CATCH_PLATFORM_WINDOWS)
10070
10071 // end catch_windows_h_proxy.h
10072 #include <sstream>
10073
10074 namespace Catch {
10075     namespace {
10076
10077         struct IColourImpl {
10078             virtual ~IColourImpl() = default;
10079             virtual void use( Colour::Code _colourCode ) = 0;
10080         };
10081
10082         struct NoColourImpl : IColourImpl {
10083             void use( Colour::Code ) override {}
10084
10085             static IColourImpl* instance() {
10086                 static NoColourImpl s_instance;
10087                 return &s_instance;
10088             }
10089         };
10090
10091     } // anon namespace
10092 } // namespace Catch
10093
10094 #if !defined( CATCH_CONFIG_COLOUR_NONE ) && !defined( CATCH_CONFIG_COLOUR_WINDOWS ) && !defined(
CATCH_CONFIG_COLOUR_ANSI )
10095 #    ifdef CATCH_PLATFORM_WINDOWS
10096 #        define CATCH_CONFIG_COLOUR_WINDOWS
10097 #    else
10098 #        define CATCH_CONFIG_COLOUR_ANSI
10099 #    endif
10100 #endif
10101
10102 #if defined( CATCH_CONFIG_COLOUR_WINDOWS )
10103
10104 namespace Catch {
10105     namespace {
10106
10107         class Win32ColourImpl : public IColourImpl {
10108         public:
10109             Win32ColourImpl() : stdoutHandle( GetStdHandle(STD_OUTPUT_HANDLE) )
10110             {
10111                 CONSOLE_SCREEN_BUFFER_INFO csbiInfo;
10112                 GetConsoleScreenBufferInfo( stdoutHandle, &csbiInfo );
10113                 originalForegroundAttributes = csbiInfo.wAttributes & ~( BACKGROUND_GREEN | BACKGROUND_RED
| BACKGROUND_BLUE | BACKGROUND_INTENSITY );
10114                 originalBackgroundAttributes = csbiInfo.wAttributes & ~( FOREGROUND_GREEN | FOREGROUND_RED
| FOREGROUND_BLUE | FOREGROUND_INTENSITY );
10115             }
10116
10117             void use( Colour::Code _colourCode ) override {
10118                 switch( _colourCode ) {
10119                     case Colour::None:         return setTextAttribute( originalForegroundAttributes );

```

```

10120         case Colour::White:      return setTextAttribute( FOREGROUND_GREEN | FOREGROUND_RED |
FOREGROUND_BLUE );
10121         case Colour::Red:         return setTextAttribute( FOREGROUND_RED );
10122         case Colour::Green:       return setTextAttribute( FOREGROUND_GREEN );
10123         case Colour::Blue:        return setTextAttribute( FOREGROUND_BLUE );
10124         case Colour::Cyan:        return setTextAttribute( FOREGROUND_BLUE | FOREGROUND_GREEN );
10125         case Colour::Yellow:      return setTextAttribute( FOREGROUND_RED | FOREGROUND_GREEN );
10126         case Colour::Grey:        return setTextAttribute( 0 );
10127
10128         case Colour::LightGrey:    return setTextAttribute( FOREGROUND_INTENSITY );
10129         case Colour::BrightRed:    return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_RED );
10130         case Colour::BrightGreen:  return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_GREEN );
10131         case Colour::BrightWhite:  return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_GREEN | FOREGROUND_RED | FOREGROUND_BLUE );
10132         case Colour::BrightYellow: return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_RED | FOREGROUND_GREEN );
10133
10134         case Colour::Bright: CATCH_INTERNAL_ERROR( "not a colour" );
10135
10136         default:
10137             CATCH_ERROR( "Unknown colour requested" );
10138     }
10139 }
10140
10141 private:
10142     void setTextAttribute( WORD _textAttribute ) {
10143         SetConsoleTextAttribute( stdoutHandle, _textAttribute | originalBackgroundAttributes );
10144     }
10145     HANDLE stdoutHandle;
10146     WORD originalForegroundAttributes;
10147     WORD originalBackgroundAttributes;
10148 };
10149
10150 IColourImpl* platformColourInstance() {
10151     static Win32ColourImpl s_instance;
10152
10153     IConfigPtr config = getCurrentContext().getConfig();
10154     UseColour::YesOrNo colourMode = config
? config->useColour()
: UseColour::Auto;
10155     if( colourMode == UseColour::Auto )
10156         colourMode = UseColour::Yes;
10157     return colourMode == UseColour::Yes
? &s_instance
: NoColourImpl::instance();
10158 }
10159 }
10160 } // end anon namespace
10161 } // end namespace Catch
10162
10163 #elif defined( CATCH_CONFIG_COLOUR_ANSI )
10164 #include <unistd.h>
10165
10166 namespace Catch {
10167     namespace {
10168         // use POSIX/ ANSI console terminal codes
10169         // Thanks to Adam Strzelecki for original contribution
10170         // (http://github.com/nanoant)
10171         // https://github.com/philsquared/Catch/pull/131
10172         class PosixColourImpl : public IColourImpl {
10173         public:
10174             void use( Colour::Code _colourCode ) override {
10175                 switch( _colourCode ) {
10176                     case Colour::None:
10177                     case Colour::White:      return setColour( "[0m" );
10178                     case Colour::Red:         return setColour( "[0;31m" );
10179                     case Colour::Green:       return setColour( "[0;32m" );
10180                     case Colour::Blue:        return setColour( "[0;34m" );
10181                     case Colour::Cyan:        return setColour( "[0;36m" );
10182                     case Colour::Yellow:      return setColour( "[0;33m" );
10183                     case Colour::Grey:        return setColour( "[1;30m" );
10184
10185                     case Colour::LightGrey:    return setColour( "[0;37m" );
10186                     case Colour::BrightRed:    return setColour( "[1;31m" );
10187                     case Colour::BrightGreen:  return setColour( "[1;32m" );
10188                     case Colour::BrightWhite:  return setColour( "[1;37m" );
10189                     case Colour::BrightYellow: return setColour( "[1;33m" );
10190
10191                     case Colour::Bright: CATCH_INTERNAL_ERROR( "not a colour" );
10192                     default: CATCH_INTERNAL_ERROR( "Unknown colour requested" );
10193                 }
10194             }
10195         };
10196     }
10197     static IColourImpl* instance() {

```

```

10202         static PosixColourImpl s_instance;
10203         return &s_instance;
10204     }
10205
10206     private:
10207         void setColour( const char* _escapeCode ) {
10208             getCurrentContext().getConfig()->stream()
10209                 << '\033' << _escapeCode;
10210         }
10211     };
10212
10213     bool useColourOnPlatform() {
10214         return
10215             #if defined(CATCH_PLATFORM_MAC) || defined(CATCH_PLATFORM_IPHONE)
10216                 !isDebuggerActive() &&
10217             #endif
10218             #if !(defined(__DJGPP__) && defined(__STRICT_ANSI__))
10219                 isatty(STDOUT_FILENO)
10220             #else
10221                 false
10222             #endif
10223         ;
10224     }
10225     IColourImpl* platformColourInstance() {
10226         ErrnoGuard guard;
10227         IConfigPtr config = getCurrentContext().getConfig();
10228         UseColour::YesOrNo colourMode = config
10229             ? config->useColour()
10230             : UseColour::Auto;
10231         if( colourMode == UseColour::Auto )
10232             colourMode = useColourOnPlatform()
10233                 ? UseColour::Yes
10234                 : UseColour::No;
10235         return colourMode == UseColour::Yes
10236             ? PosixColourImpl::instance()
10237             : NoColourImpl::instance();
10238     }
10239 } // end anon namespace
10240 } // end namespace Catch
10241
10242 #else // not Windows or ANSI //////////////////////////////////////
10243
10244 namespace Catch {
10245     static IColourImpl* platformColourInstance() { return NoColourImpl::instance(); }
10246 } // end namespace Catch
10247
10248 #endif // Windows/ ANSI/ None
10249
10250 namespace Catch {
10251     Colour::Colour( Code _colourCode ) { use( _colourCode ); }
10252     Colour::Colour( Colour&& other ) noexcept {
10253         m_moved = other.m_moved;
10254         other.m_moved = true;
10255     }
10256     Colour& Colour::operator=( Colour&& other ) noexcept {
10257         m_moved = other.m_moved;
10258         other.m_moved = true;
10259         return *this;
10260     }
10261     Colour::~Colour(){ if( !m_moved ) use( None ); }
10262
10263     void Colour::use( Code _colourCode ) {
10264         static IColourImpl* impl = platformColourInstance();
10265         // Strictly speaking, this cannot possibly happen.
10266         // However, under some conditions it does happen (see #1626),
10267         // and this change is small enough that we can let practicality
10268         // triumph over purity in this case.
10269         if (impl != nullptr) {
10270             impl->use( _colourCode );
10271         }
10272     }
10273
10274     std::ostream& operator << ( std::ostream& os, Colour const& ) {
10275         return os;
10276     }
10277 } // end namespace Catch
10278
10279 #if defined(__clang__)
10280 #pragma clang diagnostic pop
10281 #endif
10282

```



```

10289 // end catch_console_colour.cpp
10290 // start catch_context.cpp
10291
10292 namespace Catch {
10293
10294     class Context : public IMutableContext, NonCopyable {
10295     public: // IContext
10296         IResultCapture* getResultCapture() override {
10297             return m_resultCapture;
10298         }
10299         IRunner* getRunner() override {
10300             return m_runner;
10301         }
10302     }
10303
10304     IConfigPtr const& getConfig() const override {
10305         return m_config;
10306     }
10307
10308     ~Context() override;
10309
10310     public: // IMutableContext
10311         void setResultCapture( IResultCapture* resultCapture ) override {
10312             m_resultCapture = resultCapture;
10313         }
10314         void setRunner( IRunner* runner ) override {
10315             m_runner = runner;
10316         }
10317         void setConfig( IConfigPtr const& config ) override {
10318             m_config = config;
10319         }
10320
10321         friend IMutableContext& getCurrentMutableContext();
10322
10323     private:
10324         IConfigPtr m_config;
10325         IRunner* m_runner = nullptr;
10326         IResultCapture* m_resultCapture = nullptr;
10327     };
10328
10329     IMutableContext* IMutableContext::currentContext = nullptr;
10330
10331     void IMutableContext::createContext() {
10332     {
10333         currentContext = new Context();
10334     }
10335
10336     void cleanUpContext() {
10337         delete IMutableContext::currentContext;
10338         IMutableContext::currentContext = nullptr;
10339     }
10340     IContext::~IContext() = default;
10341     IMutableContext::~IMutableContext() = default;
10342     Context::~Context() = default;
10343
10344     SimplePcg32& rng() {
10345         static SimplePcg32 s_rng;
10346         return s_rng;
10347     }
10348
10349 }
10350 // end catch_context.cpp
10351 // start catch_debug_console.cpp
10352
10353 // start catch_debug_console.h
10354
10355 #include <string>
10356
10357 namespace Catch {
10358     void writeToDebugConsole( std::string const& text );
10359 }
10360
10361 // end catch_debug_console.h
10362 #if defined(CATCH_CONFIG_ANDROID_LOGWRITE)
10363 #include <android/log.h>
10364
10365     namespace Catch {
10366         void writeToDebugConsole( std::string const& text ) {
10367             __android_log_write( ANDROID_LOG_DEBUG, "Catch", text.c_str() );
10368         }
10369     }
10370
10371 #elif defined(CATCH_PLATFORM_WINDOWS)
10372
10373     namespace Catch {
10374         void writeToDebugConsole( std::string const& text ) {
10375             ::OutputDebugStringA( text.c_str() );

```

```

10376     }
10377 }
10378
10379 #else
10380
10381 namespace Catch {
10382     void writeToDebugConsole( std::string const& text ) {
10383         // !TBD: Need a version for Mac/ XCode and other IDEs
10384         Catch::cout() << text;
10385     }
10386 }
10387
10388 #endif // Platform
10389 // end catch_debug_console.cpp
10390 // start catch_debugger.cpp
10391
10392 #if defined(CATCH_PLATFORM_MAC) || defined(CATCH_PLATFORM_IPHONE)
10393
10394 # include <cassert>
10395 # include <sys/types.h>
10396 # include <unistd.h>
10397 # include <cstdint>
10398 # include <ostream>
10399
10400 #ifdef __apple_build_version__
10401     // These headers will only compile with AppleClang (XCode)
10402     // For other compilers (Clang, GCC, ... ) we need to exclude them
10403 # include <sys/sysctl.h>
10404 #endif
10405
10406 namespace Catch {
10407     #ifdef __apple_build_version__
10408         // The following function is taken directly from the following technical note:
10409         // https://developer.apple.com/library/archive/qa/qa1361/_index.html
10410
10411         // Returns true if the current process is being debugged (either
10412         // running under the debugger or has a debugger attached post facto).
10413         bool isDebuggerActive() {
10414             int mib[4];
10415             struct kinfo_proc info;
10416             std::size_t size;
10417
10418             // Initialize the flags so that, if sysctl fails for some bizarre
10419             // reason, we get a predictable result.
10420
10421             info.kp_proc.p_flag = 0;
10422
10423             // Initialize mib, which tells sysctl the info we want, in this case
10424             // we're looking for information about a specific process ID.
10425
10426             mib[0] = CTL_KERN;
10427             mib[1] = KERN_PROC;
10428             mib[2] = KERN_PROC_PID;
10429             mib[3] = getpid();
10430
10431             // Call sysctl.
10432
10433             size = sizeof(info);
10434             if( sysctl(mib, sizeof(mib) / sizeof(*mib), &info, &size, nullptr, 0) != 0 ) {
10435                 Catch::cerr() << "\n** Call to sysctl failed - unable to determine if debugger is
active **\n" << std::endl;
10436                 return false;
10437             }
10438
10439             // We're being debugged if the P_TRACED flag is set.
10440
10441             return ( (info.kp_proc.p_flag & P_TRACED) != 0 );
10442         }
10443     #else
10444         bool isDebuggerActive() {
10445             // We need to find another way to determine this for non-appleclang compilers on macOS
10446             return false;
10447         }
10448     #endif
10449 } // namespace Catch
10450
10451 #elif defined(CATCH_PLATFORM_LINUX)
10452 # include <fstream>
10453 # include <string>
10454
10455 namespace Catch {
10456     // The standard POSIX way of detecting a debugger is to attempt to
10457     // ptrace() the process, but this needs to be done from a child and not
10458     // this process itself to still allow attaching to this process later
10459     // if wanted, so is rather heavy. Under Linux we have the PID of the
10460     // "debugger" (which doesn't need to be gdb, of course, it could also
10461     // be strace, for example) in /proc/$PID/status, so just get it from

```

```

10462         // there instead.
10463         bool isDebuggerActive(){
10464             // Libstdc++ has a bug, where std::ifstream sets errno to 0
10465             // This way our users can properly assert over errno values
10466             ErrnoGuard guard;
10467             std::ifstream in("/proc/self/status");
10468             for( std::string line; std::getline(in, line); ) {
10469                 static const int PREFIX_LEN = 11;
10470                 if( line.compare(0, PREFIX_LEN, "TracerPid:") == 0 ) {
10471                     // We're traced if the PID is not 0 and no other PID starts
10472                     // with 0 digit, so it's enough to check for just a single
10473                     // character.
10474                     return line.length() > PREFIX_LEN && line[PREFIX_LEN] != '0';
10475                 }
10476             }
10477             return false;
10478         }
10479     } // namespace Catch
10480 #elif defined(_MSC_VER)
10481     extern "C" __declspec(dllimport) int __stdcall IsDebuggerPresent();
10482     namespace Catch {
10483         bool isDebuggerActive() {
10484             return IsDebuggerPresent() != 0;
10485         }
10486     }
10487 #elif defined(__MINGW32__)
10488     extern "C" __declspec(dllimport) int __stdcall IsDebuggerPresent();
10489     namespace Catch {
10490         bool isDebuggerActive() {
10491             return IsDebuggerPresent() != 0;
10492         }
10493     }
10494 #else
10495     namespace Catch {
10496         bool isDebuggerActive() { return false; }
10497     }
10498 #endif // Platform
10499 // end catch_debugger.cpp
10500 // start catch_decomposer.cpp
10501 namespace Catch {
10502     ITransientExpression::~ITransientExpression() = default;
10503     void formatReconstructedExpression( std::ostream &os, std::string const& lhs, StringRef op,
10504         std::string const& rhs ) {
10505         if( lhs.size() + rhs.size() < 40 &&
10506             lhs.find('\n') == std::string::npos &&
10507             rhs.find('\n') == std::string::npos )
10508             os << lhs << " " << op << " " << rhs;
10509         else
10510             os << lhs << "\n" << op << "\n" << rhs;
10511     }
10512 // end catch_decomposer.cpp
10513 // start catch_enforce.cpp
10514 #include <stdexcept>
10515 namespace Catch {
10516     #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS) &&
10517         !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS_CUSTOM_HANDLER)
10518     [[noreturn]]
10519     void throw_exception(std::exception const& e) {
10520         Catch::cerr() << "Catch will terminate because it needed to throw an exception.\n"
10521             << "The message was: " << e.what() << '\n';
10522         std::terminate();
10523     }
10524 #endif
10525 [[noreturn]]
10526 void throw_logic_error(std::string const& msg) {
10527     throw_exception(std::logic_error(msg));
10528 }
10529 [[noreturn]]
10530 void throw_domain_error(std::string const& msg) {
10531     throw_exception(std::domain_error(msg));
10532 }
10533 [[noreturn]]
10534 void throw_runtime_error(std::string const& msg) {
10535     throw_exception(std::runtime_error(msg));
10536 }
10537 } // namespace Catch;

```

```

10547 // end catch_enforce.cpp
10548 // start catch_enum_values_registry.cpp
10549 // start catch_enum_values_registry.h
10550
10551 #include <vector>
10552 #include <memory>
10553
10554 namespace Catch {
10555
10556     namespace Detail {
10557
10558         std::unique_ptr<EnumInfo> makeEnumInfo( StringRef enumName, StringRef allValueNames,
10559             std::vector<int> const& values );
10560
10561         class EnumValuesRegistry : public IMutableEnumValuesRegistry {
10562         public:
10563             std::vector<std::unique_ptr<EnumInfo> m_enumInfos;
10564
10565             EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums, std::vector<int>
10566                 const& values) override;
10567
10568             std::vector<StringRef> parseEnums( StringRef enums );
10569         }; // Detail
10570     } // Catch
10571 } // Catch
10572
10573 // end catch_enum_values_registry.h
10574
10575 #include <map>
10576 #include <cassert>
10577
10578 namespace Catch {
10579
10580     IMutableEnumValuesRegistry::~IMutableEnumValuesRegistry() {}
10581
10582     namespace Detail {
10583
10584         namespace {
10585             // Extracts the actual name part of an enum instance
10586             // In other words, it returns the Blue part of Bikeshed::Colour::Blue
10587             StringRef extractInstanceName(StringRef enumInstance) {
10588                 // Find last occurrence of ":"
10589                 size_t name_start = enumInstance.size();
10590                 while ( name_start > 0 && enumInstance[name_start - 1] != ':' ) {
10591                     --name_start;
10592                 }
10593                 return enumInstance.substr(name_start, enumInstance.size() - name_start);
10594             }
10595         }
10596
10597         std::vector<StringRef> parseEnums( StringRef enums ) {
10598             auto enumValues = splitStringRef( enums, ',' );
10599             std::vector<StringRef> parsed;
10600             parsed.reserve( enumValues.size() );
10601             for( auto const& enumValue : enumValues ) {
10602                 parsed.push_back(trim(extractInstanceName(enumValue)));
10603             }
10604             return parsed;
10605         }
10606
10607         EnumInfo::~EnumInfo() {}
10608
10609         StringRef EnumInfo::lookup( int value ) const {
10610             for( auto const& valueToName : m_values ) {
10611                 if( valueToName.first == value )
10612                     return valueToName.second;
10613             }
10614             return "{** unexpected enum value **}"_sr;
10615         }
10616
10617         std::unique_ptr<EnumInfo> makeEnumInfo( StringRef enumName, StringRef allValueNames,
10618             std::vector<int> const& values ) {
10619             std::unique_ptr<EnumInfo> enumInfo( new EnumInfo );
10620             enumInfo->m_name = enumName;
10621             enumInfo->m_values.reserve( values.size() );
10622
10623             const auto valueNames = Catch::Detail::parseEnums( allValueNames );
10624             assert( valueNames.size() == values.size() );
10625             std::size_t i = 0;
10626             for( auto value : values )
10627                 enumInfo->m_values.emplace_back(value, valueNames[i++]);
10628             return enumInfo;
10629         }
10630

```

```

10631         EnumInfo const& EnumValuesRegistry::registerEnum( StringRef enumName, StringRef allValueNames,
10632             std::vector<int> const& values ) {
10633             m_enumInfos.push_back(makeEnumInfo(enumName, allValueNames, values));
10634             return *m_enumInfos.back();
10635         }
10636     } // Detail
10637 } // Catch
10638
10639 // end catch_enum_values_registry.cpp
10640 // start catch_errno_guard.cpp
10641
10642 #include <cerrno>
10643
10644 namespace Catch {
10645     ErrnoGuard::ErrnoGuard():m_oldErrno(errno){}
10646     ErrnoGuard::~ErrnoGuard() { errno = m_oldErrno; }
10647 }
10648 // end catch_errno_guard.cpp
10649 // start catch_exception_translator_registry.cpp
10650
10651 // start catch_exception_translator_registry.h
10652
10653 #include <vector>
10654 #include <string>
10655 #include <memory>
10656
10657 namespace Catch {
10658     class ExceptionTranslatorRegistry : public IExceptionTranslatorRegistry {
10659     public:
10660         ~ExceptionTranslatorRegistry();
10661         virtual void registerTranslator( const IExceptionTranslator* translator );
10662         std::string translateActiveException() const override;
10663         std::string tryTranslators() const;
10664     private:
10665         std::vector<std::unique_ptr<IExceptionTranslator const> m_translators;
10666     };
10667 }
10668
10669 // end catch_exception_translator_registry.h
10670
10671 #ifdef __OBJC__
10672 #import "Foundation/Foundation.h"
10673 #endif
10674
10675 namespace Catch {
10676     ExceptionTranslatorRegistry::~ExceptionTranslatorRegistry() {
10677     }
10678
10679     void ExceptionTranslatorRegistry::registerTranslator( const IExceptionTranslator* translator ) {
10680         m_translators.push_back( std::unique_ptr<const IExceptionTranslator>( translator ) );
10681     }
10682
10683     #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
10684     std::string ExceptionTranslatorRegistry::translateActiveException() const {
10685         try {
10686             #ifdef __OBJC__
10687                 // In Objective-C try objective-c exceptions first
10688                 @try {
10689                     return tryTranslators();
10690                 } @catch (NSException *exception) {
10691                     return Catch::Detail::stringify( [exception description] );
10692                 }
10693             #else
10694                 // Compiling a mixed mode project with MSVC means that CLR
10695                 // exceptions will be caught in (...) as well. However, these
10696                 // do not fill-in std::current_exception and thus lead to crash
10697                 // when attempting rethrow.
10698                 // /EHa switch also causes structured exceptions to be caught
10699                 // here, but they fill-in current_exception properly, so
10700                 // at worst the output should be a little weird, instead of
10701                 // causing a crash.
10702                 if (std::current_exception() == nullptr) {
10703                     return "Non C++ exception. Possibly a CLR exception.";
10704                 }
10705                 return tryTranslators();
10706             #endif
10707         } catch (TestFailureException&) {
10708             std::rethrow_exception(std::current_exception());
10709         } catch (std::exception& ex) {
10710             return ex.what();
10711         }
10712     }
10713 }

```

```

10717         catch( std::string& msg ) {
10718             return msg;
10719         }
10720         catch( const char* msg ) {
10721             return msg;
10722         }
10723         catch(...) {
10724             return "Unknown exception";
10725         }
10726     }
10727
10728     std::string ExceptionTranslatorRegistry::tryTranslators() const {
10729         if (m_translators.empty()) {
10730             std::rethrow_exception(std::current_exception());
10731         } else {
10732             return m_translators[0]->translate(m_translators.begin() + 1, m_translators.end());
10733         }
10734     }
10735
10736 #else // ^^ Exceptions are enabled // Exceptions are disabled vv
10737     std::string ExceptionTranslatorRegistry::translateActiveException() const {
10738         CATCH_INTERNAL_ERROR("Attempted to translate active exception under
10739         CATCH_CONFIG_DISABLE_EXCEPTIONS!");
10740     }
10741
10742     std::string ExceptionTranslatorRegistry::tryTranslators() const {
10743         CATCH_INTERNAL_ERROR("Attempted to use exception translators under
10744         CATCH_CONFIG_DISABLE_EXCEPTIONS!");
10745     }
10746 #endif
10747 // end catch_exception_translator_registry.cpp
10748 // start catch_fatal_condition.cpp
10749
10750 #include <algorithm>
10751
10752 #if !defined( CATCH_CONFIG_WINDOWS_SEH ) && !defined( CATCH_CONFIG_POSIX_SIGNALS )
10753
10754 namespace Catch {
10755
10756     // If neither SEH nor signal handling is required, the handler impls
10757     // do not have to do anything, and can be empty.
10758     void FatalConditionHandler::engage_platform() {}
10759     void FatalConditionHandler::disengage_platform() {}
10760     FatalConditionHandler::FatalConditionHandler() = default;
10761     FatalConditionHandler::~FatalConditionHandler() = default;
10762
10763 } // end namespace Catch
10764
10765 #endif // !CATCH_CONFIG_WINDOWS_SEH && !CATCH_CONFIG_POSIX_SIGNALS
10766
10767 #if defined( CATCH_CONFIG_WINDOWS_SEH ) && defined( CATCH_CONFIG_POSIX_SIGNALS )
10768 #error "Inconsistent configuration: Windows' SEH handling and POSIX signals cannot be enabled at the
10769 same time"
10770 #endif // CATCH_CONFIG_WINDOWS_SEH && CATCH_CONFIG_POSIX_SIGNALS
10771
10772 #if defined( CATCH_CONFIG_WINDOWS_SEH ) || defined( CATCH_CONFIG_POSIX_SIGNALS )
10773 namespace {
10774     void reportFatal( char const * const message ) {
10775         Catch::getCurrentContext().getResultCapture()->handleFatalErrorCondition( message );
10776     }
10777 }
10778
10779 constexpr std::size_t minStackSizeForErrors = 32 * 1024;
10780 } // end unnamed namespace
10781
10782 #endif // CATCH_CONFIG_WINDOWS_SEH || CATCH_CONFIG_POSIX_SIGNALS
10783
10784 #if defined( CATCH_CONFIG_WINDOWS_SEH )
10785 namespace Catch {
10786
10787     struct SignalDefs { DWORD id; const char* name; };
10788
10789     // There is no 1-1 mapping between signals and windows exceptions.
10790     // Windows can easily distinguish between SO and SigSegV,
10791     // but SigInt, SigTerm, etc are handled differently.
10792     static SignalDefs signalDefs[] = {
10793         { static_cast<DWORD>(EXCEPTION_ILLEGAL_INSTRUCTION), "SIGILL - Illegal instruction signal" },
10794         { static_cast<DWORD>(EXCEPTION_STACK_OVERFLOW), "SIGSEGV - Stack overflow" },
10795         { static_cast<DWORD>(EXCEPTION_ACCESS_VIOLATION), "SIGSEGV - Segmentation violation signal" },
10796         { static_cast<DWORD>(EXCEPTION_INT_DIVIDE_BY_ZERO), "Divide by zero error" },
10797     };
10798
10799     static LONG CALLBACK handleVectoredException(PEXCEPTION_POINTERS ExceptionInfo) {
10800         for (auto const& def : signalDefs) {

```

```

10805         if (ExceptionInfo->ExceptionRecord->ExceptionCode == def.id) {
10806             reportFatal(def.name);
10807         }
10808     }
10809     // If its not an exception we care about, pass it along.
10810     // This stops us from eating debugger breaks etc.
10811     return EXCEPTION_CONTINUE_SEARCH;
10812 }
10813
10814 // Since we do not support multiple instantiations, we put these
10815 // into global variables and rely on cleaning them up in outlined
10816 // constructors/destructors
10817 static PVOID exceptionHandlerHandle = nullptr;
10818
10819 // For MSVC, we reserve part of the stack memory for handling
10820 // memory overflow structured exception.
10821 FatalConditionHandler::FatalConditionHandler() {
10822     ULONG guaranteeSize = static_cast<ULONG>(minStackSizeForErrors);
10823     if (!SetThreadStackGuarantee(&guaranteeSize)) {
10824         // We do not want to fully error out, because needing
10825         // the stack reserve should be rare enough anyway.
10826         Catch::cerr()
10827             << "Failed to reserve piece of stack."
10828             << " Stack overflows will not be reported successfully.";
10829     }
10830 }
10831
10832 // We do not attempt to unset the stack guarantee, because
10833 // Windows does not support lowering the stack size guarantee.
10834 FatalConditionHandler::~FatalConditionHandler() = default;
10835
10836 void FatalConditionHandler::engage_platform() {
10837     // Register as first handler in current chain
10838     exceptionHandlerHandle = AddVectoredExceptionHandler(1, handleVectoredException);
10839     if (!exceptionHandlerHandle) {
10840         CATCH_RUNTIME_ERROR("Could not register vectored exception handler");
10841     }
10842 }
10843
10844 void FatalConditionHandler::disengage_platform() {
10845     if (!RemoveVectoredExceptionHandler(exceptionHandlerHandle)) {
10846         CATCH_RUNTIME_ERROR("Could not unregister vectored exception handler");
10847     }
10848     exceptionHandlerHandle = nullptr;
10849 }
10850
10851 } // end namespace Catch
10852
10853 #endif // CATCH_CONFIG_WINDOWS_SEH
10854
10855 #if defined( CATCH_CONFIG_POSIX_SIGNALS )
10856
10857 #include <signal.h>
10858
10859 namespace Catch {
10860
10861     struct SignalDefs {
10862         int id;
10863         const char* name;
10864     };
10865
10866     static SignalDefs signalDefs[] = {
10867         { SIGINT, "SIGINT - Terminal interrupt signal" },
10868         { SIGILL, "SIGILL - Illegal instruction signal" },
10869         { SIGFPE, "SIGFPE - Floating point error signal" },
10870         { SIGSEGV, "SIGSEGV - Segmentation violation signal" },
10871         { SIGTERM, "SIGTERM - Termination request signal" },
10872         { SIGABRT, "SIGABRT - Abort (abnormal termination) signal" }
10873     };
10874
10875 // Older GCCs trigger -Wmissing-field-initializers for T foo = {}
10876 // which is zero initialization, but not explicit. We want to avoid
10877 // that.
10878 #if defined(__GNUC__)
10879 #    pragma GCC diagnostic push
10880 #    pragma GCC diagnostic ignored "-Wmissing-field-initializers"
10881 #endif
10882
10883     static char* altStackMem = nullptr;
10884     static std::size_t altStackSize = 0;
10885     static stack_t oldSigStack{};
10886     static struct sigaction oldSigActions[sizeof(signalDefs) / sizeof(SignalDefs)]{};
10887
10888     static void restorePreviousSignalHandlers() {
10889         // We set signal handlers back to the previous ones. Hopefully
10890         // nobody overwrote them in the meantime, and doesn't expect
10891         // their signal handlers to live past ours given that they

```

```

10892         // installed them after ours..
10893         for (std::size_t i = 0; i < sizeof(signalDefs) / sizeof(SignalDefs); ++i) {
10894             sigaction(signalDefs[i].id, &oldSigActions[i], nullptr);
10895         }
10896         // Return the old stack
10897         sigaltstack(&oldSigStack, nullptr);
10898     }
10899
10900     static void handleSignal( int sig ) {
10901         char const * name = "<unknown signal>";
10902         for (auto const& def : signalDefs) {
10903             if (sig == def.id) {
10904                 name = def.name;
10905                 break;
10906             }
10907         }
10908         // We need to restore previous signal handlers and let them do
10909         // their thing, so that the users can have the debugger break
10910         // when a signal is raised, and so on.
10911         restorePreviousSignalHandlers();
10912         reportFatal( name );
10913         raise( sig );
10914     }
10915
10916     FatalConditionHandler::FatalConditionHandler() {
10917         assert(!altStackMem && "Cannot initialize POSIX signal handler when one already exists");
10918         if (altStackSize == 0) {
10919             altStackSize = std::max(static_cast<size_t>(SIGSTKSZ), minStackSizeForErrors);
10920         }
10921         altStackMem = new char[altStackSize]();
10922     }
10923
10924     FatalConditionHandler::~FatalConditionHandler() {
10925         delete[] altStackMem;
10926         // We signal that another instance can be constructed by zeroing
10927         // out the pointer.
10928         altStackMem = nullptr;
10929     }
10930
10931     void FatalConditionHandler::engage_platform() {
10932         stack_t sigStack;
10933         sigStack.ss_sp = altStackMem;
10934         sigStack.ss_size = altStackSize;
10935         sigStack.ss_flags = 0;
10936         sigaltstack(&sigStack, &oldSigStack);
10937         struct sigaction sa = { };
10938
10939         sa.sa_handler = handleSignal;
10940         sa.sa_flags = SA_ONSTACK;
10941         for (std::size_t i = 0; i < sizeof(signalDefs)/sizeof(SignalDefs); ++i) {
10942             sigaction(signalDefs[i].id, &sa, &oldSigActions[i]);
10943         }
10944     }
10945
10946 #if defined(__GNUC__)
10947 #pragma GCC diagnostic pop
10948 #endif
10949
10950     void FatalConditionHandler::disengage_platform() {
10951         restorePreviousSignalHandlers();
10952     }
10953 } // end namespace Catch
10954
10955 #endif // CATCH_CONFIG_POSIX_SIGNALS
10956 // end catch_fatal_condition.cpp
10957 // start catch_generators.cpp
10958
10959 #include <limits>
10960 #include <set>
10961
10962 namespace Catch {
10963     IGeneratorTracker::~IGeneratorTracker() {}
10964
10965     const char* GeneratorException::what() const noexcept {
10966         return m_msg;
10967     }
10968
10969 namespace Generators {
10970     GeneratorUntypedBase::~GeneratorUntypedBase() {}
10971
10972     auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
10973     IGeneratorTracker& {
10974         return getResultCapture().acquireGeneratorTracker( generatorName, lineInfo );
10975     }
10976 }
10977

```



```

10978
10979 } // namespace Generators
10980 } // namespace Catch
10981 // end catch_generators.cpp
10982 // start catch_interfaces_capture.cpp
10983
10984 namespace Catch {
10985     IResultCapture::~IResultCapture() = default;
10986 }
10987 // end catch_interfaces_capture.cpp
10988 // start catch_interfaces_config.cpp
10989
10990 namespace Catch {
10991     IConfig::~IConfig() = default;
10992 }
10993 // end catch_interfaces_config.cpp
10994 // start catch_interfaces_exception.cpp
10995
10996 namespace Catch {
10997     IExceptionTranslator::~IExceptionTranslator() = default;
10998     IExceptionTranslatorRegistry::~IExceptionTranslatorRegistry() = default;
10999 }
11000 // end catch_interfaces_exception.cpp
11001 // start catch_interfaces_registry_hub.cpp
11002
11003 namespace Catch {
11004     IRegistryHub::~IRegistryHub() = default;
11005     IMutableRegistryHub::~IMutableRegistryHub() = default;
11006 }
11007 // end catch_interfaces_registry_hub.cpp
11008 // start catch_interfaces_reporter.cpp
11009
11010 // start catch_reporter_listening.h
11011
11012 namespace Catch {
11013
11014     class ListeningReporter : public IStreamingReporter {
11015     public:
11016         using Reporters = std::vector<IStreamingReporterPtr>;
11017         Reporters m_listeners;
11018         IStreamingReporterPtr m_reporter = nullptr;
11019         ReporterPreferences m_preferences;
11020
11021         ListeningReporter();
11022
11023         void addListener( IStreamingReporterPtr&& listener );
11024         void addReporter( IStreamingReporterPtr&& reporter );
11025
11026     public: // IStreamingReporter
11027         ReporterPreferences getPreferences() const override;
11028
11029         void noMatchingTestCases( std::string const& spec ) override;
11030
11031         void reportInvalidArguments( std::string const& arg ) override;
11032
11033         static std::set<Verbosity> getSupportedVerbsosities();
11034
11035     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
11036         void benchmarkPreparing( std::string const& name ) override;
11037         void benchmarkStarting( BenchmarkInfo const& benchmarkInfo ) override;
11038         void benchmarkEnded( BenchmarkStats<> const& benchmarkStats ) override;
11039         void benchmarkFailed( std::string const& ) override;
11040     #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
11041
11042         void testRunStarting( TestRunInfo const& testRunInfo ) override;
11043         void testGroupStarting( GroupInfo const& groupInfo ) override;
11044         void testCaseStarting( TestCaseInfo const& testInfo ) override;
11045         void sectionStarting( SectionInfo const& sectionInfo ) override;
11046         void assertionStarting( AssertionInfo const& assertionInfo ) override;
11047
11048         // The return value indicates if the messages buffer should be cleared:
11049         bool assertionEnded( AssertionStats const& assertionStats ) override;
11050         void sectionEnded( SectionStats const& sectionStats ) override;
11051         void testCaseEnded( TestCaseStats const& testCaseStats ) override;
11052         void testGroupEnded( TestGroupStats const& testGroupStats ) override;
11053         void testRunEnded( TestRunStats const& testRunStats ) override;
11054
11055         void skipTest( TestCaseInfo const& testInfo ) override;
11056         bool isMulti() const override;
11057     };
11058
11059 } // namespace Catch
11060
11061 } // end namespace Catch
11062
11063 // end catch_reporter_listening.h
11064 namespace Catch {

```

```

11065
11066 ReporterConfig::ReporterConfig( IConfigPtr const& _fullConfig )
11067 :   m_stream( &_fullConfig->stream() ), m_fullConfig( _fullConfig ) {}
11068
11069 ReporterConfig::ReporterConfig( IConfigPtr const& _fullConfig, std::ostream& _stream )
11070 :   m_stream( &_stream ), m_fullConfig( _fullConfig ) {}
11071
11072 std::ostream& ReporterConfig::stream() const { return *m_stream; }
11073 IConfigPtr ReporterConfig::fullConfig() const { return m_fullConfig; }
11074
11075 TestRunInfo::TestRunInfo( std::string const& _name ) : name( _name ) {}
11076
11077 GroupInfo::GroupInfo(   std::string const& _name,
11078                       std::size_t _groupIndex,
11079                       std::size_t _groupsCount )
11080 :   name( _name ),
11081   groupIndex( _groupIndex ),
11082   groupsCounts( _groupsCount )
11083 {}
11084
11085 AssertionStats::AssertionStats( AssertionResult const& _assertionResult,
11086                                std::vector<MessageInfo> const& _infoMessages,
11087                                Totals const& _totals )
11088 :   assertionResult( _assertionResult ),
11089   infoMessages( _infoMessages ),
11090   totals( _totals )
11091 {
11092     assertionResult.m_resultData.lazyExpression.m_transientExpression =
11093     _assertionResult.m_resultData.lazyExpression.m_transientExpression;
11094
11095     if( assertionResult.hasMessage() ) {
11096         // Copy message into messages list.
11097         // !TBD This should have been done earlier, somewhere
11098         MessageBuilder builder( assertionResult.getTestMacroName(),
11099                                assertionResult.getSourceInfo(), assertionResult.getResultType() );
11100         builder « assertionResult.getMessage();
11101         builder.m_info.message = builder.m_stream.str();
11102         infoMessages.push_back( builder.m_info );
11103     }
11104
11105     AssertionStats::~AssertionStats() = default;
11106
11107 SectionStats::SectionStats(   SectionInfo const& _sectionInfo,
11108                             Counts const& _assertions,
11109                             double _durationInSeconds,
11110                             bool _missingAssertions )
11111 :   sectionInfo( _sectionInfo ),
11112   assertions( _assertions ),
11113   durationInSeconds( _durationInSeconds ),
11114   missingAssertions( _missingAssertions )
11115 {}
11116
11117 SectionStats::~SectionStats() = default;
11118
11119 TestCaseStats::TestCaseStats(   TestCaseInfo const& _testInfo,
11120                               Totals const& _totals,
11121                               std::string const& _stdOut,
11122                               std::string const& _stdErr,
11123                               bool _aborting )
11124 :   testInfo( _testInfo ),
11125   totals( _totals ),
11126   stdOut( _stdOut ),
11127   stdErr( _stdErr ),
11128   aborting( _aborting )
11129 {}
11130
11131 TestCaseStats::~TestCaseStats() = default;
11132
11133 TestGroupStats::TestGroupStats(   GroupInfo const& _groupInfo,
11134                                 Totals const& _totals,
11135                                 bool _aborting )
11136 :   groupInfo( _groupInfo ),
11137   totals( _totals ),
11138   aborting( _aborting )
11139 {}
11140
11141 TestGroupStats::TestGroupStats(   GroupInfo const& _groupInfo )
11142 :   groupInfo( _groupInfo ),
11143   aborting( false )
11144 {}
11145
11146 TestGroupStats::~TestGroupStats() = default;
11147
11148 TestRunStats::TestRunStats(   TestRunInfo const& _runInfo,
11149                             Totals const& _totals,

```

```

11150         bool _aborting )
11151     :   runInfo( _runInfo ),
11152         totals( _totals ),
11153         aborting( _aborting )
11154     {}
11155
11156     TestRunStats::~TestRunStats() = default;
11157
11158     void IStreamingReporter::fatalErrorEncountered( StringRef ) {}
11159     bool IStreamingReporter::isMulti() const { return false; }
11160
11161     IReporterFactory::~IReporterFactory() = default;
11162     IReporterRegistry::~IReporterRegistry() = default;
11163
11164 } // end namespace Catch
11165 // end catch_interfaces_reporter.cpp
11166 // start catch_interfaces_runner.cpp
11167
11168 namespace Catch {
11169     IRunner::~IRunner() = default;
11170 }
11171 // end catch_interfaces_runner.cpp
11172 // start catch_interfaces_testcase.cpp
11173
11174 namespace Catch {
11175     ITestInvoker::~ITestInvoker() = default;
11176     ITestCaseRegistry::~ITestCaseRegistry() = default;
11177 }
11178 // end catch_interfaces_testcase.cpp
11179 // start catch_leak_detector.cpp
11180
11181 #ifdef CATCH_CONFIG_WINDOWS_CRTDBG
11182 #include <crtdbg.h>
11183
11184 namespace Catch {
11185     LeakDetector::LeakDetector() {
11186         int flag = _CrtSetDbgFlag(_CRTDBG_REPORT_FLAG);
11187         flag |= _CRTDBG_LEAK_CHECK_DF;
11188         flag |= _CRTDBG_ALLOC_MEM_DF;
11189         _CrtSetDbgFlag(flag);
11190         _CrtSetReportMode(_CRT_WARN, _CRTDBG_MODE_FILE | _CRTDBG_MODE_DEBUG);
11191         _CrtSetReportFile(_CRT_WARN, _CRTDBG_FILE_STDERR);
11192         // Change this to leaking allocation's number to break there
11193         _CrtSetBreakAlloc(-1);
11194     }
11195 }
11196 #else
11197
11198 #else
11199     Catch::LeakDetector::LeakDetector() {}
11200 #endif
11201
11202 #endif
11203
11204 Catch::LeakDetector::~LeakDetector() {
11205     Catch::cleanUp();
11206 }
11207 // end catch_leak_detector.cpp
11208 // start catch_list.cpp
11209
11210 // start catch_list.h
11211
11212 #include <set>
11213
11214 namespace Catch {
11215     std::size_t listTests( Config const& config );
11216
11217     std::size_t listTestsNamesOnly( Config const& config );
11218
11219     struct TagInfo {
11220         void add( std::string const& spelling );
11221         std::string all() const;
11222
11223         std::set<std::string> spellings;
11224         std::size_t count = 0;
11225     };
11226
11227     std::size_t listTags( Config const& config );
11228
11229     std::size_t listReporters();
11230
11231     Option<std::size_t> list( std::shared_ptr<Config> const& config );
11232 } // end namespace Catch
11233
11234 // end catch_list.h
11235
11236 // end catch_list.cpp

```

```

11237 // start catch_text.h
11238
11239 namespace Catch {
11240     using namespace clara::TextFlow;
11241 }
11242
11243 // end catch_text.h
11244 #include <limits>
11245 #include <algorithm>
11246 #include <iomanip>
11247
11248 namespace Catch {
11249
11250     std::size_t listTests( Config const& config ) {
11251         TestSpec const& testSpec = config.testSpec();
11252         if( config.hasTestFilters() )
11253             Catch::cout() << "Matching test cases:\n";
11254         else {
11255             Catch::cout() << "All available test cases:\n";
11256         }
11257
11258         auto matchedTestCases = filterTests( getAllTestCasesSorted( config ), testSpec, config );
11259         for( auto const& testCaseInfo : matchedTestCases ) {
11260             Colour::Code colour = testCaseInfo.isHidden()
11261                 ? Colour::SecondaryText
11262                 : Colour::None;
11263             Colour colourGuard( colour );
11264
11265             Catch::cout() << Column( testCaseInfo.name ).initialIndent( 2 ).indent( 4 ) << "\n";
11266             if( config.verbosity() >= Verbosity::High ) {
11267                 Catch::cout() << Column( Catch::Detail::stringify( testCaseInfo.lineInfo ) ).indent(4)
11268                 << std::endl;
11269                 std::string description = testCaseInfo.description;
11270                 if( description.empty() )
11271                     description = "(NO DESCRIPTION)";
11272                 Catch::cout() << Column( description ).indent(4) << std::endl;
11273                 if( !testCaseInfo.tags.empty() )
11274                     Catch::cout() << Column( testCaseInfo.tagsAsString() ).indent( 6 ) << "\n";
11275             }
11276
11277             if( !config.hasTestFilters() )
11278                 Catch::cout() << pluralise( matchedTestCases.size(), "test case" ) << '\n' << std::endl;
11279             else
11280                 Catch::cout() << pluralise( matchedTestCases.size(), "matching test case" ) << '\n' <<
11281                 std::endl;
11282             return matchedTestCases.size();
11283         }
11284
11285         std::size_t listTestsNamesOnly( Config const& config ) {
11286             TestSpec const& testSpec = config.testSpec();
11287             std::size_t matchedTests = 0;
11288             std::vector<TestCases> matchedTestCases = filterTests( getAllTestCasesSorted( config ),
11289             testSpec, config );
11290             for( auto const& testCaseInfo : matchedTestCases ) {
11291                 matchedTests++;
11292                 if( startsWith( testCaseInfo.name, '#' ) )
11293                     Catch::cout() << "' ' << testCaseInfo.name << "' '";
11294                 else
11295                     Catch::cout() << testCaseInfo.name;
11296                 if ( config.verbosity() >= Verbosity::High )
11297                     Catch::cout() << "\t@" << testCaseInfo.lineInfo;
11298                 Catch::cout() << std::endl;
11299             }
11300             return matchedTests;
11301         }
11302
11303         void TagInfo::add( std::string const& spelling ) {
11304             ++count;
11305             spellings.insert( spelling );
11306         }
11307
11308         std::string TagInfo::all() const {
11309             size_t size = 0;
11310             for (auto const& spelling : spellings) {
11311                 // Add 2 for the brackets
11312                 size += spelling.size() + 2;
11313             }
11314
11315             std::string out; out.reserve(size);
11316             for (auto const& spelling : spellings) {
11317                 out += '[';
11318                 out += spelling;
11319                 out += ']';
11320             }
11321             return out;
11322         }
11323     }

```

```

11321
11322     std::size_t listTags( Config const& config ) {
11323         TestSpec const& testSpec = config.testSpec();
11324         if( config.hasTestFilters() )
11325             Catch::cout() << "Tags for matching test cases:\n";
11326         else {
11327             Catch::cout() << "All available tags:\n";
11328         }
11329
11330         std::map<std::string, TagInfo> tagCounts;
11331
11332         std::vector<TestCase> matchedTestCases = filterTests( getAllTestCasesSorted( config ),
testSpec, config );
11333         for( auto const& testCase : matchedTestCases ) {
11334             for( auto const& tagName : testCase.getTestCaseInfo().tags ) {
11335                 std::string lcaseTagName = toLower( tagName );
11336                 auto countIt = tagCounts.find( lcaseTagName );
11337                 if( countIt == tagCounts.end() )
11338                     countIt = tagCounts.insert( std::make_pair( lcaseTagName, TagInfo() ) ).first;
11339                 countIt->second.add( tagName );
11340             }
11341         }
11342
11343         for( auto const& tagCount : tagCounts ) {
11344             ReusableStringStream rss;
11345             rss << " " << std::setw(2) << tagCount.second.count << " ";
11346             auto str = rss.str();
11347             auto wrapper = Column( tagCount.second.all() )
11348                             .initialIndent( 0 )
11349                             .indent( str.size() )
11350                             .width( CATCH_CONFIG_CONSOLE_WIDTH-10 );
11351             Catch::cout() << str << wrapper << '\n';
11352         }
11353         Catch::cout() << pluralise( tagCounts.size(), "tag" ) << '\n' << std::endl;
11354         return tagCounts.size();
11355     }
11356
11357     std::size_t listReporters() {
11358         Catch::cout() << "Available reporters:\n";
11359         IReporterRegistry::FactoryMap const& factories =
getRegistryHub().getReporterRegistry().getFactories();
11360         std::size_t maxNameLen = 0;
11361         for( auto const& factoryKvp : factories )
11362             maxNameLen = (std::max)( maxNameLen, factoryKvp.first.size() );
11363
11364         for( auto const& factoryKvp : factories ) {
11365             Catch::cout()
11366                 << Column( factoryKvp.first + ":" )
11367                     .indent(2)
11368                     .width( 5+maxNameLen )
11369                 + Column( factoryKvp.second->getDescription() )
11370                     .initialIndent(0)
11371                     .indent(2)
11372                     .width( CATCH_CONFIG_CONSOLE_WIDTH - maxNameLen-8 )
11373                 << "\n";
11374         }
11375         Catch::cout() << std::endl;
11376         return factories.size();
11377     }
11378
11379     Option<std::size_t> list( std::shared_ptr<Config> const& config ) {
11380         Option<std::size_t> listedCount;
11381         getCurrentMutableContext().setConfig( config );
11382         if( config->listTests() )
11383             listedCount = listedCount.valueOr(0) + listTests( *config );
11384         if( config->listTestNamesOnly() )
11385             listedCount = listedCount.valueOr(0) + listTestsNamesOnly( *config );
11386         if( config->listTags() )
11387             listedCount = listedCount.valueOr(0) + listTags( *config );
11388         if( config->listReporters() )
11389             listedCount = listedCount.valueOr(0) + listReporters();
11390         return listedCount;
11391     }
11392
11393 } // end namespace Catch
11394 // end catch_list.cpp
11395 // start catch_matchers.cpp
11396
11397 namespace Catch {
11398     namespace Matchers {
11399         namespace Impl {
11400
11401             std::string MatcherUntypedBase::toString() const {
11402                 if( m_cachedToString.empty() )
11403                     m_cachedToString = describe();
11404                 return m_cachedToString;
11405             }

```

```

11406
11407     MatcherUntypedBase::~MatcherUntypedBase() = default;
11408
11409     } // namespace Impl
11410 } // namespace Matchers
11411
11412 using namespace Matchers;
11413 using Matchers::Impl::MatcherBase;
11414
11415 } // namespace Catch
11416 // end catch_matchers.cpp
11417 // start catch_matchers_exception.cpp
11418
11419 namespace Catch {
11420 namespace Matchers {
11421 namespace Exception {
11422
11423 bool ExceptionMessageMatcher::match(std::exception const& ex) const {
11424     return ex.what() == m_message;
11425 }
11426
11427 std::string ExceptionMessageMatcher::describe() const {
11428     return "exception message matches \"" + m_message + "\"";
11429 }
11430
11431 }
11432 Exception::ExceptionMessageMatcher Message(std::string const& message) {
11433     return Exception::ExceptionMessageMatcher(message);
11434 }
11435
11436 // namespace Exception
11437 } // namespace Matchers
11438 } // namespace Catch
11439 // end catch_matchers_exception.cpp
11440 // start catch_matchers_floating.cpp
11441
11442 // start catch_polyfills.hpp
11443
11444 namespace Catch {
11445     bool isnan(float f);
11446     bool isnan(double d);
11447 }
11448
11449 // end catch_polyfills.hpp
11450 // start catch_to_string.hpp
11451
11452 #include <string>
11453
11454 namespace Catch {
11455     template <typename T>
11456     std::string to_string(T const& t) {
11457 #if defined(CATCH_CONFIG_CPP11_TO_STRING)
11458         return std::to_string(t);
11459 #else
11460         ReusableStringStream rss;
11461         rss << t;
11462         return rss.str();
11463 #endif
11464     }
11465 } // end namespace Catch
11466
11467 // end catch_to_string.hpp
11468 #include <algorithm>
11469 #include <cmath>
11470 #include <cstdlib>
11471 #include <cstdint>
11472 #include <cstring>
11473 #include <sstream>
11474 #include <type_traits>
11475 #include <iomanip>
11476 #include <limits>
11477
11478 namespace Catch {
11479 namespace {
11480
11481     int32_t convert(float f) {
11482         static_assert(sizeof(float) == sizeof(int32_t), "Important ULP matcher assumption violated");
11483         int32_t i;
11484         std::memcpy(&i, &f, sizeof(f));
11485         return i;
11486     }
11487
11488     int64_t convert(double d) {
11489         static_assert(sizeof(double) == sizeof(int64_t), "Important ULP matcher assumption violated");
11490         int64_t i;
11491         std::memcpy(&i, &d, sizeof(d));
11492         return i;

```

```

11493     }
11494
11495     template <typename FP>
11496     bool almostEqualUlp(FP lhs, FP rhs, uint64_t maxUlpDiff) {
11497         // Comparison with NaN should always be false.
11498         // This way we can rule it out before getting into the ugly details
11499         if (Catch::isnan(lhs) || Catch::isnan(rhs)) {
11500             return false;
11501         }
11502
11503         auto lc = convert(lhs);
11504         auto rc = convert(rhs);
11505
11506         if ((lc < 0) != (rc < 0)) {
11507             // Potentially we can have +0 and -0
11508             return lhs == rhs;
11509         }
11510
11511         // static cast as a workaround for IBM XLC
11512         auto ulpDiff = std::abs(static_cast<FP>(lc - rc));
11513         return static_cast<uint64_t>(ulpDiff) <= maxUlpDiff;
11514     }
11515
11516 #if defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
11517
11518     float nextafter(float x, float y) {
11519         return ::nextafterf(x, y);
11520     }
11521
11522     double nextafter(double x, double y) {
11523         return ::nextafter(x, y);
11524     }
11525
11526 #endif // ^^^ CATCH_CONFIG_GLOBAL_NEXTAFTER ^^^
11527
11528     template <typename FP>
11529     FP step(FP start, FP direction, uint64_t steps) {
11530         for (uint64_t i = 0; i < steps; ++i) {
11531             #if defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
11532                 start = Catch::nextafter(start, direction);
11533             #else
11534                 start = std::nextafter(start, direction);
11535             #endif
11536         }
11537         return start;
11538     }
11539
11540     // Performs equivalent check of std::fabs(lhs - rhs) <= margin
11541     // But without the subtraction to allow for INFINITY in comparison
11542     bool marginComparison(double lhs, double rhs, double margin) {
11543         return (lhs + margin >= rhs) && (rhs + margin >= lhs);
11544     }
11545
11546     template <typename FloatingPoint>
11547     void write(std::ostream& out, FloatingPoint num) {
11548         out << std::scientific
11549             << std::setprecision(std::numeric_limits<FloatingPoint>::max_digits10 - 1)
11550             << num;
11551     }
11552
11553 } // end anonymous namespace
11554
11555 namespace Matchers {
11556 namespace Floating {
11557
11558     enum class FloatingPointKind : uint8_t {
11559         Float,
11560         Double
11561     };
11562
11563     WithinAbsMatcher::WithinAbsMatcher(double target, double margin)
11564         : m_target{ target }, m_margin{ margin } {
11565         CATCH_ENFORCE(margin >= 0, "Invalid margin: " << margin << ".");
11566         << " Margin has to be non-negative.");
11567     }
11568
11569     // Performs equivalent check of std::fabs(lhs - rhs) <= margin
11570     // But without the subtraction to allow for INFINITY in comparison
11571     bool WithinAbsMatcher::match(double const& matchee) const {
11572         return (matchee + m_margin >= m_target) && (m_target + m_margin >= matchee);
11573     }
11574
11575     std::string WithinAbsMatcher::describe() const {
11576         return "is within " + ::Catch::Detail::stringify(m_margin) + " of " +
11577             ::Catch::Detail::stringify(m_target);
11578     }
11579
11580 }

```

```

11579     WithinUlpMatcher::WithinUlpMatcher(double target, uint64_t ulps, FloatingPointKind baseType)
11580     :m_target{ target }, m_ulps{ ulps }, m_type{ baseType } {
11581         CATCH_ENFORCE(m_type == FloatingPointKind::Double
11582             || m_ulps < (std::numeric_limits<uint32_t>::max)(),
11583             "Provided ULP is impossibly large for a float comparison.");
11584     }
11585
11586     #if defined(__clang__)
11587     #pragma clang diagnostic push
11588     // Clang <3.5 reports on the default branch in the switch below
11589     #pragma clang diagnostic ignored "-Wunreachable-code"
11590     #endif
11591
11592     bool WithinUlpMatcher::match(double const& matchee) const {
11593         switch (m_type) {
11594             case FloatingPointKind::Float:
11595                 return almostEqualUlp<float>(static_cast<float>(matchee), static_cast<float>(m_target),
11596                     m_ulps);
11597             case FloatingPointKind::Double:
11598                 return almostEqualUlp<double>(matchee, m_target, m_ulps);
11599             default:
11600                 CATCH_INTERNAL_ERROR( "Unknown FloatingPointKind value" );
11601         }
11602     }
11603
11604     #if defined(__clang__)
11605     #pragma clang diagnostic pop
11606     #endif
11607
11608     std::string WithinUlpMatcher::describe() const {
11609         std::stringstream ret;
11610
11611         ret << "is within " << m_ulps << " ULPs of ";
11612
11613         if (m_type == FloatingPointKind::Float) {
11614             write(ret, static_cast<float>(m_target));
11615             ret << 'f';
11616         } else {
11617             write(ret, m_target);
11618         }
11619
11620         ret << " [";
11621         if (m_type == FloatingPointKind::Double) {
11622             write(ret, step(m_target, static_cast<double>(-INFINITY), m_ulps));
11623             ret << ", ";
11624             write(ret, step(m_target, static_cast<double>( INFINITY), m_ulps));
11625         } else {
11626             // We have to cast INFINITY to float because of MinGW, see #1782
11627             write(ret, step(static_cast<float>(m_target), static_cast<float>(-INFINITY), m_ulps));
11628             ret << ", ";
11629             write(ret, step(static_cast<float>(m_target), static_cast<float>( INFINITY), m_ulps));
11630         }
11631         ret << "]";
11632
11633         return ret.str();
11634     }
11635
11636     WithinRelMatcher::WithinRelMatcher(double target, double epsilon):
11637         m_target(target),
11638         m_epsilon(epsilon){
11639         CATCH_ENFORCE(m_epsilon >= 0., "Relative comparison with epsilon < 0 does not make sense.");
11640         CATCH_ENFORCE(m_epsilon < 1., "Relative comparison with epsilon >= 1 does not make sense.");
11641     }
11642
11643     bool WithinRelMatcher::match(double const& matchee) const {
11644         const auto relMargin = m_epsilon * (std::max)(std::fabs(matchee), std::fabs(m_target));
11645         return marginComparison(matchee, m_target,
11646             std::isinf(relMargin)? 0 : relMargin);
11647     }
11648
11649     std::string WithinRelMatcher::describe() const {
11650         Catch::ReusableStringStream sstr;
11651         sstr << "and " << m_target << " are within " << m_epsilon * 100. << "% of each other";
11652         return sstr.str();
11653     }
11654 } // namespace Floating
11655
11656 Floating::WithinUlpMatcher WithinULP(double target, uint64_t maxUlpDiff) {
11657     return Floating::WithinUlpMatcher(target, maxUlpDiff, Floating::FloatingPointKind::Double);
11658 }
11659
11660 Floating::WithinUlpMatcher WithinULP(float target, uint64_t maxUlpDiff) {
11661     return Floating::WithinUlpMatcher(target, maxUlpDiff, Floating::FloatingPointKind::Float);
11662 }
11663
11664 Floating::WithinAbsMatcher WithinAbs(double target, double margin) {

```



```

11665     return Floating::WithinAbsMatcher(target, margin);
11666 }
11667
11668 Floating::WithinRelMatcher WithinRel(double target, double eps) {
11669     return Floating::WithinRelMatcher(target, eps);
11670 }
11671
11672 Floating::WithinRelMatcher WithinRel(double target) {
11673     return Floating::WithinRelMatcher(target, std::numeric_limits<double>::epsilon() * 100);
11674 }
11675
11676 Floating::WithinRelMatcher WithinRel(float target, float eps) {
11677     return Floating::WithinRelMatcher(target, eps);
11678 }
11679
11680 Floating::WithinRelMatcher WithinRel(float target) {
11681     return Floating::WithinRelMatcher(target, std::numeric_limits<float>::epsilon() * 100);
11682 }
11683
11684 } // namespace Matchers
11685 } // namespace Catch
11686 // end catch_matchers_floating.cpp
11687 // start catch_matchers_generic.cpp
11688
11689 std::string Catch::Matchers::Generic::Detail::finalizeDescription(const std::string& desc) {
11690     if (desc.empty()) {
11691         return "matches undescribed predicate";
11692     } else {
11693         return "matches predicate: \"" + desc + "\"";
11694     }
11695 }
11696 // end catch_matchers_generic.cpp
11697 // start catch_matchers_string.cpp
11698
11699 #include <regex>
11700
11701 namespace Catch {
11702 namespace Matchers {
11703
11704     namespace StdString {
11705
11706         CasedString::CasedString( std::string const& str, CaseSensitive::Choice caseSensitivity )
11707         :   m_caseSensitivity( caseSensitivity ),
11708             m_str( adjustString( str ) )
11709         {}
11710         std::string CasedString::adjustString( std::string const& str ) const {
11711             return m_caseSensitivity == CaseSensitive::No
11712                 ? toLower( str )
11713                 : str;
11714         }
11715         std::string CasedString::caseSensitivitySuffix() const {
11716             return m_caseSensitivity == CaseSensitive::No
11717                 ? " (case insensitive)"
11718                 : std::string();
11719         }
11720
11721         StringMatcherBase::StringMatcherBase( std::string const& operation, CasedString const&
comparator )
11722         : m_comparator( comparator ),
11723           m_operation( operation ) {}
11724
11725         std::string StringMatcherBase::describe() const {
11726             std::string description;
11727             description.reserve(5 + m_operation.size() + m_comparator.m_str.size() +
m_comparator.caseSensitivitySuffix().size());
11728             description += m_operation;
11729             description += ": ";
11730             description += m_comparator.m_str;
11731             description += " ";
11732             description += m_comparator.caseSensitivitySuffix();
11733             return description;
11734         }
11735
11736         EqualsMatcher::EqualsMatcher( CasedString const& comparator ) : StringMatcherBase( "equals",
comparator ) {}
11737
11738         bool EqualsMatcher::match( std::string const& source ) const {
11739             return m_comparator.adjustString( source ) == m_comparator.m_str;
11740         }
11741
11742         ContainsMatcher::ContainsMatcher( CasedString const& comparator ) : StringMatcherBase(
"contains", comparator ) {}
11743
11744         bool ContainsMatcher::match( std::string const& source ) const {
11745             return contains( m_comparator.adjustString( source ), m_comparator.m_str );
11746         }
11747
11748     }

```

```

11749
11750     StartsWithMatcher::StartsWithMatcher( CasedString const& comparator ) : StringMatcherBase(
11751 "starts with", comparator ) {}
11752
11753     bool StartsWithMatcher::match( std::string const& source ) const {
11754         return startsWith( m_comparator.adjustString( source ), m_comparator.m_str );
11755     }
11756
11757     EndsWithMatcher::EndsWithMatcher( CasedString const& comparator ) : StringMatcherBase( "ends
11758 with", comparator ) {}
11759
11760     bool EndsWithMatcher::match( std::string const& source ) const {
11761         return endsWith( m_comparator.adjustString( source ), m_comparator.m_str );
11762     }
11763
11764     RegexMatcher::RegexMatcher( std::string regex, CaseSensitive::Choice caseSensitivity ):
11765 m_regex( std::move( regex ), m_caseSensitivity( caseSensitivity ) ) {}
11766
11767     bool RegexMatcher::match( std::string const& matchee ) const {
11768         auto flags = std::regex::ECMAScript; // ECMAScript is the default syntax option anyway
11769         if ( m_caseSensitivity == CaseSensitive::Choice::No ) {
11770             flags |= std::regex::icase;
11771         }
11772         auto reg = std::regex( m_regex, flags );
11773         return std::regex_match( matchee, reg );
11774     }
11775
11776     std::string RegexMatcher::describe() const {
11777         return "matches " + ::Catch::Detail::stringify( m_regex ) + ( ( m_caseSensitivity ==
11778 CaseSensitive::Choice::Yes ) ? " case sensitively" : " case insensitively" );
11779     }
11780
11781 } // namespace StdString
11782
11783 StdString::EqualsMatcher Equals( std::string const& str, CaseSensitive::Choice caseSensitivity ) {
11784     return StdString::EqualsMatcher( StdString::CasedString( str, caseSensitivity ) );
11785 }
11786
11787 StdString::ContainsMatcher Contains( std::string const& str, CaseSensitive::Choice caseSensitivity
11788 ) {
11789     return StdString::ContainsMatcher( StdString::CasedString( str, caseSensitivity ) );
11790 }
11791
11792 StdString::EndsWithMatcher EndsWith( std::string const& str, CaseSensitive::Choice caseSensitivity
11793 ) {
11794     return StdString::EndsWithMatcher( StdString::CasedString( str, caseSensitivity ) );
11795 }
11796
11797 StdString::StartsWithMatcher StartsWith( std::string const& str, CaseSensitive::Choice
11798 caseSensitivity ) {
11799     return StdString::StartsWithMatcher( StdString::CasedString( str, caseSensitivity ) );
11800 }
11801
11802 StdString::RegexMatcher Matches( std::string const& regex, CaseSensitive::Choice caseSensitivity ) {
11803     return StdString::RegexMatcher( regex, caseSensitivity );
11804 }
11805
11806 } // namespace Matchers
11807 } // namespace Catch
11808 // end catch_matchers_string.cpp
11809 // start catch_message.cpp
11810
11811 // start catch_uncaught_exceptions.h
11812
11813 namespace Catch {
11814     bool uncaught_exceptions();
11815 } // end namespace Catch
11816
11817 // end catch_uncaught_exceptions.h
11818 #include <cassert>
11819 #include <stack>
11820
11821 namespace Catch {
11822
11823     MessageInfo::MessageInfo( StringRef const& _macroName,
11824                               SourceLineInfo const& _lineInfo,
11825                               ResultWas::OfType _type )
11826     :   macroName( _macroName ),
11827         lineInfo( _lineInfo ),
11828         type( _type ),
11829         sequence( ++globalCount )
11830     {}
11831
11832     bool MessageInfo::operator==( MessageInfo const& other ) const {
11833         return sequence == other.sequence;
11834     }
11835
11836     bool MessageInfo::operator<( MessageInfo const& other ) const {
11837         return sequence < other.sequence;
11838     }

```

```

11829
11830 // This may need protecting if threading support is added
11831 unsigned int MessageInfo::globalCount = 0;
11832
11833
11834
11835 Catch::MessageBuilder::MessageBuilder( StringRef const& macroName,
11836                                         SourceLineInfo const& lineInfo,
11837                                         ResultWas::OfType type )
11838     :m_info(macroName, lineInfo, type) {}
11839
11840
11841
11842 ScopedMessage::ScopedMessage( MessageBuilder const& builder )
11843 : m_info( builder.m_info ), m_moved()
11844 {
11845     m_info.message = builder.m_stream.str();
11846     getResultCapture().pushScopedMessage( m_info );
11847 }
11848
11849 ScopedMessage::ScopedMessage( ScopedMessage&& old )
11850 : m_info( old.m_info ), m_moved()
11851 {
11852     old.m_moved = true;
11853 }
11854
11855 ScopedMessage::~ScopedMessage() {
11856     if ( !uncaught_exceptions() && !m_moved ) {
11857         getResultCapture().popScopedMessage(m_info);
11858     }
11859 }
11860
11861 Capturer::Capturer( StringRef macroName, SourceLineInfo const& lineInfo, ResultWas::OfType
resultType, StringRef names ) {
11862     auto trimmed = [&] (size_t start, size_t end) {
11863         while (names[start] == ',' || isspace(static_cast<unsigned char>(names[start]))) {
11864             ++start;
11865         }
11866         while (names[end] == ',' || isspace(static_cast<unsigned char>(names[end]))) {
11867             --end;
11868         }
11869         return names.substr(start, end - start + 1);
11870     };
11871     auto skipq = [&] (size_t start, char quote) {
11872         for (auto i = start + 1; i < names.size(); ++i) {
11873             if (names[i] == quote)
11874                 return i;
11875             if (names[i] == '\\')
11876                 ++i;
11877         }
11878         CATCH_INTERNAL_ERROR("CAPTURE parsing encountered unmatched quote");
11879     };
11880
11881     size_t start = 0;
11882     std::stack<char> openings;
11883     for (size_t pos = 0; pos < names.size(); ++pos) {
11884         char c = names[pos];
11885         switch (c) {
11886             case '[':
11887             case '{':
11888             case '(':
11889                 // It is basically impossible to disambiguate between
11890                 // comparison and start of template args in this context
11891                 case '<':
11892                     openings.push(c);
11893                     break;
11894             case ']':
11895             case '}':
11896             case ')':
11897                 case '>':
11898                     openings.pop();
11899                     break;
11900             case '"':
11901             case '\\':
11902                 pos = skipq(pos, c);
11903                 break;
11904             case ',':
11905                 if (start != pos && openings.empty()) {
11906                     m_messages.emplace_back(macroName, lineInfo, resultType);
11907                     m_messages.back().message = static_cast<std::string>(trimmed(start, pos));
11908                     m_messages.back().message += " := ";
11909                     start = pos;
11910                 }
11911             }
11912     }
11913     assert(openings.empty() && "Mismatched openings");
11914     m_messages.emplace_back(macroName, lineInfo, resultType);
11915     m_messages.back().message = static_cast<std::string>(trimmed(start, names.size() - 1));
11916     m_messages.back().message += " := ";

```

```

11917     }
11918     Capturer::~Capturer() {
11919         if ( !uncaught_exceptions() ){
11920             assert( m_captured == m_messages.size() );
11921             for( size_t i = 0; i < m_captured; ++i )
11922                 m_resultCapture.popScopedMessage( m_messages[i] );
11923         }
11924     }
11925
11926     void Capturer::captureValue( size_t index, std::string const& value ) {
11927         assert( index < m_messages.size() );
11928         m_messages[index].message += value;
11929         m_resultCapture.pushScopedMessage( m_messages[index] );
11930         m_captured++;
11931     }
11932
11933 } // end namespace Catch
11934 // end catch_message.cpp
11935 // start catch_output_redirect.cpp
11936
11937 // start catch_output_redirect.h
11938 #ifndef TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
11939 #define TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
11940
11941 #include <cstdio>
11942 #include <iosfwd>
11943 #include <string>
11944
11945 namespace Catch {
11946
11947     class RedirectedStream {
11948     public:
11949         RedirectedStream( std::ostream& originalStream, std::ostream& redirectionStream );
11950         ~RedirectedStream();
11951
11952     public:
11953         class RedirectedStdOut {
11954         public:
11955             RedirectedStdOut();
11956             auto str() const -> std::string;
11957         };
11958
11959         class RedirectedStdErr {
11960         public:
11961             RedirectedStdErr();
11962             auto str() const -> std::string;
11963         };
11964
11965         // StdErr has two constituent streams in C++, std::cerr and std::clog
11966         // This means that we need to redirect 2 streams into 1 to keep proper
11967         // order of writes
11968         class RedirectedStreams {
11969         public:
11970             RedirectedStreams( RedirectedStreams const& ) = delete;
11971             RedirectedStreams& operator=( RedirectedStreams const& ) = delete;
11972             RedirectedStreams( RedirectedStreams&& ) = delete;
11973             RedirectedStreams& operator=( RedirectedStreams&& ) = delete;
11974
11975             RedirectedStreams( std::string& redirectedCout, std::string& redirectedCerr );
11976             ~RedirectedStreams();
11977
11978         private:
11979             std::string& m_redirectedCout;
11980             std::string& m_redirectedCerr;
11981             RedirectedStdOut m_redirectedStdOut;
11982             RedirectedStdErr m_redirectedStdErr;
11983         };
11984
11985 #if defined( CATCH_CONFIG_NEW_CAPTURE )
11986
11987         // Windows's implementation of std::tmpfile is terrible (it tries
11988         // to create a file inside system folder, thus requiring elevated
11989         // privileges for the binary), so we have to use tmpnam(_s) and
11990         // create the file ourselves there.
11991         class TempFile {
11992         public:
11993             TempFile( TempFile const& ) = delete;
11994             TempFile& operator=( TempFile const& ) = delete;
11995             TempFile( TempFile&& ) = delete;

```

```

12004         TempFile& operator=(TempFile&&) = delete;
12005
12006         TempFile();
12007         ~TempFile();
12008
12009         std::FILE* getFile();
12010         std::string getContents();
12011
12012     private:
12013         std::FILE* m_file = nullptr;
12014         #if defined(_MSC_VER)
12015             char m_buffer[L_tmpnam] = { 0 };
12016         #endif
12017     };
12018
12019     class OutputRedirect {
12020     public:
12021         OutputRedirect(OutputRedirect const&) = delete;
12022         OutputRedirect& operator=(OutputRedirect const&) = delete;
12023         OutputRedirect(OutputRedirect&&) = delete;
12024         OutputRedirect& operator=(OutputRedirect&&) = delete;
12025
12026         OutputRedirect(std::string& stdout_dest, std::string& stderr_dest);
12027         ~OutputRedirect();
12028
12029     private:
12030         int m_originalStdout = -1;
12031         int m_originalStderr = -1;
12032         TempFile m_stdoutFile;
12033         TempFile m_stderrFile;
12034         std::string& m_stdoutDest;
12035         std::string& m_stderrDest;
12036     };
12037
12038 #endif
12039
12040 } // end namespace Catch
12041
12042 #endif // TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
12043 // end catch_output_redirect.h
12044 #include <cstdio>
12045 #include <cstring>
12046 #include <fstream>
12047 #include <sstream>
12048 #include <stdexcept>
12049
12050 #if defined(CATCH_CONFIG_NEW_CAPTURE)
12051     #if defined(_MSC_VER)
12052         #include <io.h> // _dup and _dup2
12053         #define dup _dup
12054         #define dup2 _dup2
12055         #define fileno _fileno
12056     #else
12057         #include <unistd.h> // dup and dup2
12058     #endif
12059 #endif
12060
12061 namespace Catch {
12062
12063     RedirectedStream::RedirectedStream( std::ostream& originalStream, std::ostream& redirectionStream
12064 )
12065     :   m_originalStream( originalStream ),
12066         m_redirectionStream( redirectionStream ),
12067         m_prevBuf( m_originalStream.rdbuf() )
12068     {
12069         m_originalStream.rdbuf( m_redirectionStream.rdbuf() );
12070     }
12071
12072     RedirectedStream::~RedirectedStream() {
12073         m_originalStream.rdbuf( m_prevBuf );
12074     }
12075
12076     RedirectedStdOut::RedirectedStdOut() : m_cout( Catch::cout(), m_rss.get() ) {}
12077     auto RedirectedStdOut::str() const -> std::string { return m_rss.str(); }
12078
12079     RedirectedStdErr::RedirectedStdErr()
12080     :   m_cerr( Catch::cerr(), m_rss.get() ),
12081         m_clog( Catch::clog(), m_rss.get() )
12082     {}
12083     auto RedirectedStdErr::str() const -> std::string { return m_rss.str(); }
12084
12085     RedirectedStreams::RedirectedStreams(std::string& redirectedCout, std::string& redirectedCerr)
12086     :   m_redirectedCout(redirectedCout),
12087         m_redirectedCerr(redirectedCerr)
12088     {}
12089
12090     RedirectedStreams::~RedirectedStreams() {

```

```

12090         m_redirectedCout += m_redirectedStdOut.str();
12091         m_redirectedCerr += m_redirectedStdErr.str();
12092     }
12093
12094     #if defined(CATCH_CONFIG_NEW_CAPTURE)
12095
12096     #if defined(_MSC_VER)
12097         TempFile::TempFile() {
12098             if (tmpnam_s(m_buffer)) {
12099                 CATCH_RUNTIME_ERROR("Could not get a temp filename");
12100             }
12101             if (fopen_s(&m_file, m_buffer, "w+")) {
12102                 char buffer[100];
12103                 if (strerror_s(buffer, errno)) {
12104                     CATCH_RUNTIME_ERROR("Could not translate errno to a string");
12105                 }
12106                 CATCH_RUNTIME_ERROR("Could not open the temp file: '" < m_buffer < "' because: " <
buffer);
12107             }
12108         }
12109     #else
12110         TempFile::TempFile() {
12111             m_file = std::tmpfile();
12112             if (!m_file) {
12113                 CATCH_RUNTIME_ERROR("Could not create a temp file.");
12114             }
12115         }
12116     #endif
12117
12118     TempFile::~TempFile() {
12119         // TBD: What to do about errors here?
12120         std::fclose(m_file);
12121         // We manually create the file on Windows only, on Linux
12122         // it will be autodeleted
12123         #if defined(_MSC_VER)
12124             std::remove(m_buffer);
12125         #endif
12126     }
12127
12128     FILE* TempFile::getFile() {
12129         return m_file;
12130     }
12131
12132     std::string TempFile::getContents() {
12133         std::stringstream sstr;
12134         char buffer[100] = {};
12135         std::rewind(m_file);
12136         while (std::fgets(buffer, sizeof(buffer), m_file)) {
12137             sstr << buffer;
12138         }
12139         return sstr.str();
12140     }
12141
12142     OutputRedirect::OutputRedirect(std::string& stdout_dest, std::string& stderr_dest) :
12143         m_originalStdout(dup(1)),
12144         m_originalStderr(dup(2)),
12145         m_stdoutDest(stdout_dest),
12146         m_stderrDest(stderr_dest) {
12147         dup2(fileno(m_stdoutFile.getFile()), 1);
12148         dup2(fileno(m_stderrFile.getFile()), 2);
12149     }
12150
12151     OutputRedirect::~OutputRedirect() {
12152         Catch::cout() << std::flush;
12153         fflush(stdout);
12154         // Since we support overriding these streams, we flush cerr
12155         // even though std::cerr is unbuffered
12156         Catch::cerr() << std::flush;
12157         Catch::clog() << std::flush;
12158         fflush(stderr);
12159
12160         dup2(m_originalStdout, 1);
12161         dup2(m_originalStderr, 2);
12162
12163         m_stdoutDest += m_stdoutFile.getContents();
12164         m_stderrDest += m_stderrFile.getContents();
12165     }
12166
12167 #endif // CATCH_CONFIG_NEW_CAPTURE
12168
12169 } // namespace Catch
12170
12171 #if defined(CATCH_CONFIG_NEW_CAPTURE)
12172     #if defined(_MSC_VER)
12173         #undef dup
12174         #undef dup2
12175     #endif

```

```

12176     #undef fileno
12177     #endif
12178 #endif
12179 // end catch_output_redirect.cpp
12180 // start catch_polyfills.cpp
12181
12182 #include <cmath>
12183
12184 namespace Catch {
12185
12186 #if !defined(CATCH_CONFIG_POLYFILL_ISNAN)
12187     bool isnan(float f) {
12188         return std::isnan(f);
12189     }
12190     bool isnan(double d) {
12191         return std::isnan(d);
12192     }
12193 #else
12194     // For now we only use this for embarcadero
12195     bool isnan(float f) {
12196         return std::_isnan(f);
12197     }
12198     bool isnan(double d) {
12199         return std::_isnan(d);
12200     }
12201 #endif
12202
12203 } // end namespace Catch
12204 // end catch_polyfills.cpp
12205 // start catch_random_number_generator.cpp
12206
12207 namespace Catch {
12208
12209     namespace {
12210
12211         #if defined(_MSC_VER)
12212         #pragma warning(push)
12213         #pragma warning(disable:4146) // we negate uint32 during the rotate
12214         #endif
12215         // Safe rotr implementation thanks to John Regehr
12216         uint32_t rotate_right(uint32_t val, uint32_t count) {
12217             const uint32_t mask = 31;
12218             count &= mask;
12219             return (val >> count) | (val << (-count & mask));
12220         }
12221
12222         #if defined(_MSC_VER)
12223         #pragma warning(pop)
12224         #endif
12225     }
12226 }
12227
12228     SimplePcg32::SimplePcg32(result_type seed_) {
12229         seed(seed_);
12230     }
12231
12232     void SimplePcg32::seed(result_type seed_) {
12233         m_state = 0;
12234         (*this)();
12235         m_state += seed_;
12236         (*this)();
12237     }
12238
12239     void SimplePcg32::discard(uint64_t skip) {
12240         // We could implement this to run in O(log n) steps, but this
12241         // should suffice for our use case.
12242         for (uint64_t s = 0; s < skip; ++s) {
12243             static_cast<void>((*this)());
12244         }
12245     }
12246
12247     SimplePcg32::result_type SimplePcg32::operator()() {
12248         // prepare the output value
12249         const uint32_t xorshifted = static_cast<uint32_t>(((m_state >> 18u) ^ m_state) >> 27u);
12250         const auto output = rotate_right(xorshifted, m_state >> 59u);
12251
12252         // advance state
12253         m_state = m_state * 6364136223846793005ULL + s_inc;
12254
12255         return output;
12256     }
12257
12258     bool operator==(SimplePcg32 const& lhs, SimplePcg32 const& rhs) {
12259         return lhs.m_state == rhs.m_state;
12260     }
12261
12262     bool operator!=(SimplePcg32 const& lhs, SimplePcg32 const& rhs) {

```

```

12263         return lhs.m_state != rhs.m_state;
12264     }
12265 }
12266 // end catch_random_number_generator.cpp
12267 // start catch_registry_hub.cpp
12268
12269 // start catch_test_case_registry_impl.h
12270
12271 #include <vector>
12272 #include <set>
12273 #include <algorithm>
12274 #include <ios>
12275
12276 namespace Catch {
12277
12278     class TestCase;
12279     struct IConfig;
12280
12281     std::vector<TestCase> sortTests( IConfig const& config, std::vector<TestCase> const&
12282     unsortedTestCases );
12283
12284     bool isThrowSafe( TestCase const& testCase, IConfig const& config );
12285     bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config );
12286
12287     void enforceNoDuplicateTestCases( std::vector<TestCase> const& functions );
12288
12289     std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
12290     testSpec, IConfig const& config );
12291     std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config );
12292
12293     class TestRegistry : public ITestRegistry {
12294     public:
12295         virtual ~TestRegistry() = default;
12296
12297         virtual void registerTest( TestCase const& testCase );
12298
12299         std::vector<TestCase> const& getAllTests() const override;
12300         std::vector<TestCase> const& getAllTestsSorted( IConfig const& config ) const override;
12301
12302     private:
12303         std::vector<TestCase> m_functions;
12304         mutable RunTests::InWhatOrder m_currentSortOrder = RunTests::InDeclarationOrder;
12305         mutable std::vector<TestCase> m_sortedFunctions;
12306         std::size_t m_unnamedCount = 0;
12307         std::ios_base::Init m_ostreamInit; // Forces cout/ cerr to be initialised
12308     };
12309
12310     class TestInvokerAsFunction : public ITestInvoker {
12311     public:
12312         TestInvokerAsFunction( void(*testAsFunction)() ) noexcept;
12313
12314         void invoke() const override;
12315     };
12316
12317     std::string extractClassName( StringRef const& classOrQualifiedMethodName );
12318
12319 } // end namespace Catch
12320
12321 // end catch_test_case_registry_impl.h
12322 // start catch_reporter_registry.h
12323
12324 #include <map>
12325
12326 namespace Catch {
12327
12328     class ReporterRegistry : public IReporterRegistry {
12329     public:
12330         ~ReporterRegistry() override;
12331
12332         IStreamingReporterPtr create( std::string const& name, IConfigPtr const& config ) const
12333         override;
12334
12335         void registerReporter( std::string const& name, IReporterFactoryPtr const& factory );
12336         void registerListener( IReporterFactoryPtr const& factory );
12337
12338         FactoryMap const& getFactories() const override;
12339         Listeners const& getListeners() const override;
12340
12341     private:
12342         FactoryMap m_factories;
12343         Listeners m_listeners;
12344     };
12345
12346 }

```



```

12349 }
12350
12351 // end catch_reporter_registry.h
12352 // start catch_tag_alias_registry.h
12353
12354 // start catch_tag_alias.h
12355
12356 #include <string>
12357
12358 namespace Catch {
12359
12360     struct TagAlias {
12361         TagAlias(std::string const& _tag, SourceLineInfo _lineInfo);
12362
12363         std::string tag;
12364         SourceLineInfo lineInfo;
12365     };
12366
12367 } // end namespace Catch
12368
12369 // end catch_tag_alias.h
12370 #include <map>
12371
12372 namespace Catch {
12373
12374     class TagAliasRegistry : public ITagAliasRegistry {
12375     public:
12376         ~TagAliasRegistry() override;
12377         TagAlias const* find( std::string const& alias ) const override;
12378         std::string expandAliases( std::string const& unexpandedTestSpec ) const override;
12379         void add( std::string const& alias, std::string const& tag, SourceLineInfo const& lineInfo );
12380
12381     private:
12382         std::map<std::string, TagAlias> m_registry;
12383     };
12384
12385 } // end namespace Catch
12386
12387 // end catch_tag_alias_registry.h
12388 // start catch_startup_exception_registry.h
12389
12390 #include <vector>
12391 #include <exception>
12392
12393 namespace Catch {
12394
12395     class StartupExceptionRegistry {
12396     #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
12397     public:
12398         void add(std::exception_ptr const& exception) noexcept;
12399         std::vector<std::exception_ptr> const& getExceptions() const noexcept;
12400     private:
12401         std::vector<std::exception_ptr> m_exceptions;
12402     #endif
12403     };
12404
12405 } // end namespace Catch
12406
12407 // end catch_startup_exception_registry.h
12408 // start catch_singletons.hpp
12409
12410 namespace Catch {
12411
12412     struct ISingleton {
12413         virtual ~ISingleton();
12414     };
12415
12416     void addSingleton( ISingleton* singleton );
12417     void cleanupSingletons();
12418
12419     template<typename SingletonImplT, typename InterfaceT = SingletonImplT, typename MutableInterfaceT
= InterfaceT>
12420     class Singleton : SingletonImplT, public ISingleton {
12421
12422     public:
12423         static auto getInternal() -> Singleton* {
12424             static Singleton* s_instance = nullptr;
12425             if( !s_instance ) {
12426                 s_instance = new Singleton;
12427                 addSingleton( s_instance );
12428             }
12429             return s_instance;
12430         }
12431
12432     public:
12433         static auto get() -> InterfaceT const& {
12434             return *getInternal();
12435         }
12436     };
12437
12438 }

```

```

12435         static auto getMutable() -> MutableInterfaceT& {
12436             return *getInternal();
12437         }
12438     };
12439
12440 } // namespace Catch
12441
12442 // end catch_singletons.hpp
12443 namespace Catch {
12444     namespace {
12445         class RegistryHub : public IRegistryHub, public IMutableRegistryHub,
12446                             private NonCopyable {
12447         public: // IRegistryHub
12448             RegistryHub() = default;
12449             IReporterRegistry const& getReporterRegistry() const override {
12450                 return m_reporterRegistry;
12451             }
12452             ITestCaseRegistry const& getTestCaseRegistry() const override {
12453                 return m_testCaseRegistry;
12454             }
12455             IExceptionTranslatorRegistry const& getExceptionTranslatorRegistry() const override {
12456                 return m_exceptionTranslatorRegistry;
12457             }
12458             ITagAliasRegistry const& getTagAliasRegistry() const override {
12459                 return m_tagAliasRegistry;
12460             }
12461             StartupExceptionRegistry const& getStartupExceptionRegistry() const override {
12462                 return m_exceptionRegistry;
12463             }
12464         public: // IMutableRegistryHub
12465             void registerReporter( std::string const& name, IReporterFactoryPtr const& factory )
12466             override {
12467                 m_reporterRegistry.registerReporter( name, factory );
12468             }
12469             void registerListener( IReporterFactoryPtr const& factory ) override {
12470                 m_reporterRegistry.registerListener( factory );
12471             }
12472             void registerTest( TestCase const& testInfo ) override {
12473                 m_testCaseRegistry.registerTest( testInfo );
12474             }
12475             void registerTranslator( const IExceptionTranslator* translator ) override {
12476                 m_exceptionTranslatorRegistry.registerTranslator( translator );
12477             }
12478             void registerTagAlias( std::string const& alias, std::string const& tag, SourceLineInfo
12479             const& lineInfo ) override {
12480                 m_tagAliasRegistry.add( alias, tag, lineInfo );
12481             }
12482             void registerStartupException() noexcept override {
12483                 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
12484                     m_exceptionRegistry.add(std::current_exception());
12485                 #else
12486                     CATCH_INTERNAL_ERROR("Attempted to register active exception under
12487                     CATCH_CONFIG_DISABLE_EXCEPTIONS!");
12488                 #endif
12489             }
12490             IMutableEnumValuesRegistry& getMutableEnumValuesRegistry() override {
12491                 return m_enumValuesRegistry;
12492             }
12493         private:
12494             TestRegistry m_testCaseRegistry;
12495             ReporterRegistry m_reporterRegistry;
12496             ExceptionTranslatorRegistry m_exceptionTranslatorRegistry;
12497             TagAliasRegistry m_tagAliasRegistry;
12498             StartupExceptionRegistry m_exceptionRegistry;
12499             Detail::EnumValuesRegistry m_enumValuesRegistry;
12500     };
12501
12502 using RegistryHubSingleton = Singleton<RegistryHub, IRegistryHub, IMutableRegistryHub>;
12503
12504 IRegistryHub const& getRegistryHub() {
12505     return RegistryHubSingleton::get();
12506 }
12507 IMutableRegistryHub& getMutableRegistryHub() {
12508     return RegistryHubSingleton::getMutable();
12509 }
12510 void cleanup() {
12511     cleanupSingletons();
12512     cleanupContext();
12513 }
12514 std::string translateActiveException() {
12515     return getRegistryHub().getExceptionTranslatorRegistry().translateActiveException();
12516 }

```

```

12519     }
12520
12521 } // end namespace Catch
12522 // end catch_registry_hub.cpp
12523 // start catch_reporter_registry.cpp
12524
12525 namespace Catch {
12526
12527     ReporterRegistry::~ReporterRegistry() = default;
12528
12529     IStreamingReporterPtr ReporterRegistry::create( std::string const& name, IConfigPtr const& config
12530 ) const {
12531         auto it = m_factories.find( name );
12532         if( it == m_factories.end() )
12533             return nullptr;
12534         return it->second->create( ReporterConfig( config ) );
12535     }
12536
12537     void ReporterRegistry::registerReporter( std::string const& name, IReporterFactoryPtr const&
12538     factory ) {
12539         m_factories.emplace(name, factory);
12540     }
12541
12542     void ReporterRegistry::registerListener( IReporterFactoryPtr const& factory ) {
12543         m_listeners.push_back( factory );
12544     }
12545
12546     IReporterRegistry::FactoryMap const& ReporterRegistry::getFactories() const {
12547         return m_factories;
12548     }
12549
12550     IReporterRegistry::Listeners const& ReporterRegistry::getListeners() const {
12551         return m_listeners;
12552     }
12553 }
12554 // end catch_reporter_registry.cpp
12555 // start catch_result_type.cpp
12556
12557 namespace Catch {
12558
12559     bool isOk( ResultWas::OfType resultType ) {
12560         return ( resultType & ResultWas::FailureBit ) == 0;
12561     }
12562
12563     bool isJustInfo( int flags ) {
12564         return flags == ResultWas::Info;
12565     }
12566
12567     ResultDisposition::Flags operator | ( ResultDisposition::Flags lhs, ResultDisposition::Flags rhs )
12568     {
12569         return static_cast<ResultDisposition::Flags>( static_cast<int>( lhs ) | static_cast<int>( rhs )
12570 );
12571     }
12572
12573     bool shouldContinueOnFailure( int flags ) { return ( flags &
12574     ResultDisposition::ContinueOnFailure ) != 0; }
12575
12576     bool shouldSuppressFailure( int flags ) { return ( flags & ResultDisposition::SuppressFail )
12577     != 0; }
12578
12579 } // end namespace Catch
12580 // end catch_result_type.cpp
12581 // start catch_run_context.cpp
12582
12583 #include <cassert>
12584 #include <algorithm>
12585 #include <sstream>
12586
12587 namespace Catch {
12588
12589     namespace Generators {
12590
12591         struct GeneratorTracker : TestCaseTracking::TrackerBase, IGeneratorTracker {
12592             GeneratorBasePtr m_generator;
12593
12594             GeneratorTracker( TestCaseTracking::NameAndLocation const& nameAndLocation,
12595             TrackerContext& ctx, ITracker* parent )
12596             : TrackerBase( nameAndLocation, ctx, parent )
12597             {}
12598             ~GeneratorTracker();
12599
12600             static GeneratorTracker& acquire( TrackerContext& ctx, TestCaseTracking::NameAndLocation
12601             const& nameAndLocation ) {
12602                 std::shared_ptr<GeneratorTracker> tracker;
12603
12604                 ITracker& currentTracker = ctx.currentTracker();
12605                 // Under specific circumstances, the generator we want
12606                 // to acquire is also the current tracker. If this is
12607                 // the case, we have to avoid looking through current
12608                 // tracker's children, and instead return the current
12609                 // tracker.
12610
12611

```

```

12598         // A case where this check is important is e.g.
12599         //     for (int i = 0; i < 5; ++i) {
12600         //         int n = GENERATE(1, 2);
12601         //     }
12602         //
12603         // without it, the code above creates 5 nested generators.
12604         if (currentTracker.nameAndLocation() == nameAndLocation) {
12605             auto thisTracker = currentTracker.parent().findChild(nameAndLocation);
12606             assert(thisTracker);
12607             assert(thisTracker->isGeneratorTracker());
12608             tracker = std::static_pointer_cast<GeneratorTracker>(thisTracker);
12609         } else if ( TestCaseTracking::ITrackerPtr childTracker = currentTracker.findChild(
nameAndLocation ) ) {
12610             assert( childTracker );
12611             assert( childTracker->isGeneratorTracker() );
12612             tracker = std::static_pointer_cast<GeneratorTracker>( childTracker );
12613         } else {
12614             tracker = std::make_shared<GeneratorTracker>( nameAndLocation, ctx,
&currentTracker );
12615             currentTracker.addChild( tracker );
12616         }
12617
12618         if( !tracker->isComplete() ) {
12619             tracker->open();
12620         }
12621
12622         return *tracker;
12623     }
12624
12625     // TrackerBase interface
12626     bool isGeneratorTracker() const override { return true; }
12627     auto hasGenerator() const -> bool override {
12628         return !!m_generator;
12629     }
12630     void close() override {
12631         TrackerBase::close();
12632         // If a generator has a child (it is followed by a section)
12633         // and none of its children have started, then we must wait
12634         // until later to start consuming its values.
12635         // This catches cases where `GENERATE` is placed between two
12636         // `SECTION`'s.
12637         // **The check for m_children.empty cannot be removed**.
12638         // doing so would break `GENERATE` `_not_` followed by `SECTION`'s.
12639         const bool should_wait_for_child = [&]() {
12640             // No children -> nobody to wait for
12641             if ( m_children.empty() ) {
12642                 return false;
12643             }
12644             // If at least one child started executing, don't wait
12645             if ( std::find_if(
12646                 m_children.begin(),
12647                 m_children.end(),
12648                 []( TestCaseTracking::ITrackerPtr tracker ) {
12649                     return tracker->hasStarted();
12650                 } ) != m_children.end() ) {
12651                 return false;
12652             }
12653
12654             // No children have started. We need to check if they _can_
12655             // start, and thus we should wait for them, or they cannot
12656             // start (due to filters), and we shouldn't wait for them
12657             auto* parent = m_parent;
12658             // This is safe: there is always at least one section
12659             // tracker in a test case tracking tree
12660             while ( !parent->isSectionTracker() ) {
12661                 parent = &( parent->parent() );
12662             }
12663             assert( parent &&
12664                 "Missing root (test case) level section" );
12665
12666             auto const& parentSection =
12667                 static_cast<SectionTracker&>( *parent );
12668             auto const& filters = parentSection.getFilters();
12669             // No filters -> no restrictions on running sections
12670             if ( filters.empty() ) {
12671                 return true;
12672             }
12673
12674             for ( auto const& child : m_children ) {
12675                 if ( child->isSectionTracker() &&
12676                     std::find( filters.begin(),
12677                         filters.end(),
12678                         static_cast<SectionTracker&>( *child )
12679                             .trimmedName() ) !=
12680                     filters.end() ) {
12681                     return true;
12682                 }

```

```

12683         }
12684         return false;
12685     }();
12686
12687     // This check is a bit tricky, because m_generator->next()
12688     // has a side-effect, where it consumes generator's current
12689     // value, but we do not want to invoke the side-effect if
12690     // this generator is still waiting for any child to start.
12691     if ( should_wait_for_child ||
12692         ( m_runState == CompletedSuccessfully &&
12693           m_generator->next() ) ) {
12694         m_children.clear();
12695         m_runState = Executing;
12696     }
12697 }
12698
12699 // IGeneratorTracker interface
12700 auto getGenerator() const -> GeneratorBasePtr const& override {
12701     return m_generator;
12702 }
12703 void setGenerator( GeneratorBasePtr&& generator ) override {
12704     m_generator = std::move( generator );
12705 }
12706 };
12707 GeneratorTracker::~GeneratorTracker() {}
12708 }
12709
12710 RunContext::RunContext( IConfigPtr const& _config, IStreamingReporterPtr&& reporter)
12711 :   m_runInfo(_config->name()),
12712     m_context(getCurrentMutableContext()),
12713     m_config(_config),
12714     m_reporter(std::move(reporter)),
12715     m_lastAssertionInfo{ StringRef(), SourceLineInfo("",0), StringRef(), ResultDisposition::Normal
12716 },
12717     m_includeSuccessfulResults( m_config->includeSuccessfulResults() ||
12718 m_reporter->getPreferences().shouldReportAllAssertions )
12719 {
12720     m_context.setRunner( this );
12721     m_context.setConfig( m_config );
12722     m_context.setResultCapture( this );
12723     m_reporter->testRunStarting( m_runInfo );
12724 }
12725
12726 RunContext::~RunContext() {
12727     m_reporter->testRunEnded( TestRunStats( m_runInfo, m_totals, aborting() ) );
12728 }
12729
12730 void RunContext::testGroupStarting( std::string const& testSpec, std::size_t groupIndex,
12731 std::size_t groupsCount ) {
12732     m_reporter->testGroupStarting( GroupInfo( testSpec, groupIndex, groupsCount ) );
12733 }
12734
12735 void RunContext::testGroupEnded( std::string const& testSpec, Totals const& totals, std::size_t
12736 groupIndex, std::size_t groupsCount ) {
12737     m_reporter->testGroupEnded( TestGroupStats( GroupInfo( testSpec, groupIndex, groupsCount ),
12738 totals, aborting() ) );
12739 }
12740
12741 Totals RunContext::runTest( TestCase const& testCase ) {
12742     Totals prevTotals = m_totals;
12743
12744     std::string redirectedCout;
12745     std::string redirectedCerr;
12746
12747     auto const& testInfo = testCase.getTestCaseInfo();
12748
12749     m_reporter->testCaseStarting( testInfo );
12750
12751     m_activeTestCase = &testCase;
12752
12753     ITracker& rootTracker = m_trackerContext.startRun();
12754     assert( rootTracker.isSectionTracker() );
12755     static_cast<SectionTracker&>(rootTracker).addInitialFilters( m_config->getSectionsToRun() );
12756     do {
12757         m_trackerContext.startCycle();
12758         m_testCaseTracker = &SectionTracker::acquire( m_trackerContext,
12759 TestCaseTracking::NameAndLocation( testInfo.name, testInfo.lineInfo ) );
12760         runCurrentTest( redirectedCout, redirectedCerr );
12761     } while ( !m_testCaseTracker->isSuccessfullyCompleted() && !aborting() );
12762
12763     Totals deltaTotals = m_totals.delta( prevTotals );
12764     if ( testInfo.expectedToFail() && deltaTotals.testCases.passed > 0 ) {
12765         deltaTotals.assertions.failed++;
12766         deltaTotals.testCases.passed--;
12767         deltaTotals.testCases.failed++;
12768     }
12769     m_totals.testCases += deltaTotals.testCases;

```

```

12764         m_reporter->testCaseEnded(TestCaseStats(testInfo,
12765             deltaTotals,
12766             redirectedCout,
12767             redirectedCerr,
12768             aborting()));
12769
12770         m_activeTestCase = nullptr;
12771         m_testCaseTracker = nullptr;
12772
12773         return deltaTotals;
12774     }
12775
12776     IConfigPtr RunContext::config() const {
12777         return m_config;
12778     }
12779
12780     IStreamingReporter& RunContext::reporter() const {
12781         return *m_reporter;
12782     }
12783
12784     void RunContext::assertionEnded(AssertionResult const & result) {
12785         if (result.getResultType() == ResultWas::Ok) {
12786             m_totals.assertions.passed++;
12787             m_lastAssertionPassed = true;
12788         } else if (!result.isOk()) {
12789             m_lastAssertionPassed = false;
12790             if( m_activeTestCase->getTestCaseInfo().okToFail() )
12791                 m_totals.assertions.failedButOk++;
12792             else
12793                 m_totals.assertions.failed++;
12794         }
12795         else {
12796             m_lastAssertionPassed = true;
12797         }
12798
12799         // We have no use for the return value (whether messages should be cleared), because messages
12800         // were made scoped
12801         // and should be let to clear themselves out.
12802         static_cast<void>(m_reporter->assertionEnded(AssertionStats(result, m_messages, m_totals)));
12803
12804         if (result.getResultType() != ResultWas::Warning)
12805             m_messageScopes.clear();
12806
12807         // Reset working state
12808         resetAssertionInfo();
12809         m_lastResult = result;
12810     }
12811
12812     void RunContext::resetAssertionInfo() {
12813         m_lastAssertionInfo.macroName = StringRef();
12814         m_lastAssertionInfo.capturedExpression = "{Unknown expression after the reported line}"_sr;
12815     }
12816
12817     bool RunContext::sectionStarted(SectionInfo const & sectionInfo, Counts & assertions) {
12818         ITracker& sectionTracker = SectionTracker::acquire(m_trackerContext,
12819             TestCaseTracking::NameAndLocation(sectionInfo.name, sectionInfo.lineInfo));
12820         if (!sectionTracker.isOpen())
12821             return false;
12822         m_activeSections.push_back(&sectionTracker);
12823
12824         m_lastAssertionInfo.lineInfo = sectionInfo.lineInfo;
12825
12826         m_reporter->sectionStarting(sectionInfo);
12827
12828         assertions = m_totals.assertions;
12829
12830         return true;
12831     }
12832
12833     auto RunContext::acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo
12834 ) -> IGeneratorTracker& {
12835         using namespace Generators;
12836         GeneratorTracker& tracker = GeneratorTracker::acquire(m_trackerContext,
12837             TestCaseTracking::NameAndLocation(
12838                 static_cast<std::string>(generatorName), lineInfo ) );
12839         m_lastAssertionInfo.lineInfo = lineInfo;
12840         return tracker;
12841     }
12842
12843     bool RunContext::testForMissingAssertions(Counts& assertions) {
12844         if (assertions.total() != 0)
12845             return false;
12846         if (!m_config->warnAboutMissingAssertions())
12847             return false;
12848         if (m_trackerContext.currentTracker().hasChildren())
12849             return false;
12850         m_totals.assertions.failed++;
12851         assertions.failed++;
12852         return true;
12853     }

```

```

12847     }
12848
12849     void RunContext::sectionEnded(SectionEndInfo const & endInfo) {
12850         Counts assertions = m_totals.assertions - endInfo.prevAssertions;
12851         bool missingAssertions = testForMissingAssertions(assertions);
12852
12853         if (!m_activeSections.empty()) {
12854             m_activeSections.back()->close();
12855             m_activeSections.pop_back();
12856         }
12857
12858         m_reporter->sectionEnded(SectionStats(endInfo.sectionInfo, assertions,
12859 endInfo.durationInSeconds, missingAssertions));
12860         m_messages.clear();
12861         m_messageScopes.clear();
12862     }
12863
12864     void RunContext::sectionEndedEarly(SectionEndInfo const & endInfo) {
12865         if (m_unfinishedSections.empty())
12866             m_activeSections.back()->fail();
12867         else
12868             m_activeSections.back()->close();
12869         m_activeSections.pop_back();
12870         m_unfinishedSections.push_back(endInfo);
12871     }
12872
12873 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
12874     void RunContext::benchmarkPreparing(std::string const& name) {
12875         m_reporter->benchmarkPreparing(name);
12876     }
12877     void RunContext::benchmarkStarting( BenchmarkInfo const& info ) {
12878         m_reporter->benchmarkStarting( info );
12879     }
12880     void RunContext::benchmarkEnded( BenchmarkStats<> const& stats ) {
12881         m_reporter->benchmarkEnded( stats );
12882     }
12883     void RunContext::benchmarkFailed(std::string const & error) {
12884         m_reporter->benchmarkFailed(error);
12885     }
12886 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
12887
12888     void RunContext::pushScopedMessage(MessageInfo const & message) {
12889         m_messages.push_back(message);
12890     }
12891
12892     void RunContext::popScopedMessage(MessageInfo const & message) {
12893         m_messages.erase(std::remove(m_messages.begin(), m_messages.end(), message),
12894 m_messages.end());
12895     }
12896
12897     void RunContext::emplaceUnscopedMessage( MessageBuilder const& builder ) {
12898         m_messageScopes.emplace_back( builder );
12899     }
12900
12901     std::string RunContext::getCurrentTestName() const {
12902         return m_activeTestCase
12903             ? m_activeTestCase->getTestCaseInfo().name
12904             : std::string();
12905     }
12906
12907     const AssertionResult * RunContext::getLastResult() const {
12908         return &(*m_lastResult);
12909     }
12910
12911     void RunContext::exceptionEarlyReported() {
12912         m_shouldReportUnexpected = false;
12913     }
12914
12915     void RunContext::handleFatalErrorCondition( StringRef message ) {
12916         // First notify reporter that bad things happened
12917         m_reporter->fatalErrorEncountered(message);
12918
12919         // Don't rebuild the result -- the stringification itself can cause more fatal errors
12920         // Instead, fake a result data.
12921         AssertionResultData tempResult( ResultWas::FatalErrorCondition, { false } );
12922         tempResult.message = static_cast<std::string>(message);
12923         AssertionResult result(m_lastAssertionInfo, tempResult);
12924
12925         assertionEnded(result);
12926
12927         handleUnfinishedSections();
12928
12929         // Recreate section for test case (as we will lose the one that was in scope)
12930         auto const& testCaseInfo = m_activeTestCase->getTestCaseInfo();
12931         SectionInfo testCaseSection(testCaseInfo.lineInfo, testCaseInfo.name);

```

```

12932         Counts assertions;
12933         assertions.failed = 1;
12934         SectionStats testCaseSectionStats(testCaseSection, assertions, 0, false);
12935         m_reporter->sectionEnded(testCaseSectionStats);
12936
12937         auto const& testInfo = m_activeTestCase->getTestCaseInfo();
12938
12939         Totals deltaTotals;
12940         deltaTotals.testCases.failed = 1;
12941         deltaTotals.assertions.failed = 1;
12942         m_reporter->testCaseEnded(TestCaseStats(testInfo,
12943             deltaTotals,
12944             std::string(),
12945             std::string(),
12946             false));
12947         m_totals.testCases.failed++;
12948         testGroupEnded(std::string(), m_totals, 1, 1);
12949         m_reporter->testRunEnded(TestRunStats(m_runInfo, m_totals, false));
12950     }
12951
12952     bool RunContext::lastAssertionPassed() {
12953         return m_lastAssertionPassed;
12954     }
12955
12956     void RunContext::assertionPassed() {
12957         m_lastAssertionPassed = true;
12958         ++m_totals.assertions.passed;
12959         resetAssertionInfo();
12960         m_messageScopes.clear();
12961     }
12962
12963     bool RunContext::aborting() const {
12964         return m_totals.assertions.failed >= static_cast<std::size_t>(m_config->abortAfter());
12965     }
12966
12967     void RunContext::runCurrentTest(std::string & redirectedCout, std::string & redirectedCerr) {
12968         auto const& testCaseInfo = m_activeTestCase->getTestCaseInfo();
12969         SectionInfo testCaseSection(testCaseInfo.lineInfo, testCaseInfo.name);
12970         m_reporter->sectionStarting(testCaseSection);
12971         Counts prevAssertions = m_totals.assertions;
12972         double duration = 0;
12973         m_shouldReportUnexpected = true;
12974         m_lastAssertionInfo = { "TEST_CASE"_sr, testCaseInfo.lineInfo, StringRef(),
ResultDisposition::Normal };
12975
12976         seedRng(*m_config);
12977
12978         Timer timer;
12979         CATCH_TRY {
12980             if (m_reporter->getPreferences().shouldRedirectStdOut) {
12981 #if !defined(CATCH_CONFIG_EXPERIMENTAL_REDIRECT)
12982                 RedirectedStreams redirectedStreams(redirectedCout, redirectedCerr);
12983
12984                 timer.start();
12985                 invokeActiveTestCase();
12986 #else
12987                 OutputRedirect r(redirectedCout, redirectedCerr);
12988                 timer.start();
12989                 invokeActiveTestCase();
12990 #endif
12991             } else {
12992                 timer.start();
12993                 invokeActiveTestCase();
12994             }
12995             duration = timer.getElapsedSeconds();
12996         } CATCH_CATCH_ANON (TestFailureException&) {
12997             // This just means the test was aborted due to failure
12998         } CATCH_CATCH_ALL {
12999             // Under CATCH_CONFIG_FAST_COMPILE, unexpected exceptions under REQUIRE assertions
13000             // are reported without translation at the point of origin.
13001             if( m_shouldReportUnexpected ) {
13002                 AssertionReaction dummyReaction;
13003                 handleUnexpectedInflightException( m_lastAssertionInfo, translateActiveException(),
dummyReaction );
13004             }
13005         }
13006         Counts assertions = m_totals.assertions - prevAssertions;
13007         bool missingAssertions = testForMissingAssertions(assertions);
13008
13009         m_testCaseTracker->close();
13010         handleUnfinishedSections();
13011         m_messages.clear();
13012         m_messageScopes.clear();
13013
13014         SectionStats testCaseSectionStats(testCaseSection, assertions, duration, missingAssertions);
13015         m_reporter->sectionEnded(testCaseSectionStats);
13016     }

```



```

13017
13018 void RunContext::invokeActiveTestCase() {
13019     FatalConditionHandlerGuard _(&m_fatalConditionhandler);
13020     m_activeTestCase->invoke();
13021 }
13022
13023 void RunContext::handleUnfinishedSections() {
13024     // If sections ended prematurely due to an exception we stored their
13025     // infos here so we can tear them down outside the unwind process.
13026     for (auto it = m_unfinishedSections.rbegin(),
13027          itEnd = m_unfinishedSections.rend();
13028          it != itEnd;
13029          ++it)
13030         sectionEnded(*it);
13031     m_unfinishedSections.clear();
13032 }
13033
13034 void RunContext::handleExpr(
13035     AssertionInfo const& info,
13036     ITransientExpression const& expr,
13037     AssertionReaction& reaction
13038 ) {
13039     m_reporter->assertionStarting( info );
13040
13041     bool negated = isFalseTest( info.resultDisposition );
13042     bool result = expr.getResult() != negated;
13043
13044     if( result ) {
13045         if (!m_includeSuccessfulResults) {
13046             assertionPassed();
13047         }
13048         else {
13049             reportExpr(info, ResultWas::Ok, &expr, negated);
13050         }
13051     }
13052     else {
13053         reportExpr(info, ResultWas::ExpressionFailed, &expr, negated );
13054         populateReaction( reaction );
13055     }
13056 }
13057 void RunContext::reportExpr(
13058     AssertionInfo const& info,
13059     ResultWas::OfType resultType,
13060     ITransientExpression const& *expr,
13061     bool negated ) {
13062
13063     m_lastAssertionInfo = info;
13064     AssertionResultData data( resultType, LazyExpression( negated ) );
13065
13066     AssertionResult assertionResult{ info, data };
13067     assertionResult.m_resultData.lazyExpression.m_transientExpression = expr;
13068
13069     assertionEnded( assertionResult );
13070 }
13071
13072 void RunContext::handleMessage(
13073     AssertionInfo const& info,
13074     ResultWas::OfType resultType,
13075     StringRef const& message,
13076     AssertionReaction& reaction
13077 ) {
13078     m_reporter->assertionStarting( info );
13079
13080     m_lastAssertionInfo = info;
13081
13082     AssertionResultData data( resultType, LazyExpression( false ) );
13083     data.message = static_cast<std::string>(message);
13084     AssertionResult assertionResult{ m_lastAssertionInfo, data };
13085     assertionEnded( assertionResult );
13086     if( !assertionResult.isOk() )
13087         populateReaction( reaction );
13088 }
13089 void RunContext::handleUnexpectedExceptionNotThrown(
13090     AssertionInfo const& info,
13091     AssertionReaction& reaction
13092 ) {
13093     handleNonExpr(info, Catch::ResultWas::DidntThrowException, reaction);
13094 }
13095
13096 void RunContext::handleUnexpectedInflightException(
13097     AssertionInfo const& info,
13098     std::string const& message,
13099     AssertionReaction& reaction
13100 ) {
13101     m_lastAssertionInfo = info;
13102
13103     AssertionResultData data( ResultWas::ThrewException, LazyExpression( false ) );

```

```

13104         data.message = message;
13105         AssertionResult assertionResult{ info, data };
13106         assertionEnded( assertionResult );
13107         populateReaction( reaction );
13108     }
13109
13110     void RunContext::populateReaction( AssertionReaction& reaction ) {
13111         reaction.shouldDebugBreak = m_config->shouldDebugBreak();
13112         reaction.shouldThrow = aborting() || (m_lastAssertionInfo.resultDisposition &
ResultDisposition::Normal);
13113     }
13114
13115     void RunContext::handleIncomplete(
13116         AssertionInfo const& info
13117     ) {
13118         m_lastAssertionInfo = info;
13119
13120         AssertionResultData data( ResultWas::ThrowException, LazyExpression( false ) );
13121         data.message = "Exception translation was disabled by CATCH_CONFIG_FAST_COMPILE";
13122         AssertionResult assertionResult{ info, data };
13123         assertionEnded( assertionResult );
13124     }
13125     void RunContext::handleNonExpr(
13126         AssertionInfo const& info,
13127         ResultWas::OfType resultType,
13128         AssertionReaction &reaction
13129     ) {
13130         m_lastAssertionInfo = info;
13131
13132         AssertionResultData data( resultType, LazyExpression( false ) );
13133         AssertionResult assertionResult{ info, data };
13134         assertionEnded( assertionResult );
13135
13136         if( !assertionResult.isOk() )
13137             populateReaction( reaction );
13138     }
13139
13140     IResultCapture& getResultCapture() {
13141         if (auto* capture = getCurrentContext().getResultCapture())
13142             return *capture;
13143         else
13144             CATCH_INTERNAL_ERROR("No result capture instance");
13145     }
13146
13147     void seedRng(IConfig const& config) {
13148         if (config.rngSeed() != 0) {
13149             std::srand(config.rngSeed());
13150             rng().seed(config.rngSeed());
13151         }
13152     }
13153
13154     unsigned int rngSeed() {
13155         return getCurrentContext().getConfig()->rngSeed();
13156     }
13157
13158 }
13159 // end catch_run_context.cpp
13160 // start catch_section.cpp
13161
13162 namespace Catch {
13163
13164     Section::Section( SectionInfo const& info )
13165     :   m_info( info ),
13166         m_sectionIncluded( getResultCapture().sectionStarted( m_info, m_assertions ) )
13167     {
13168         m_timer.start();
13169     }
13170
13171     Section::~Section() {
13172         if( m_sectionIncluded ) {
13173             SectionEndInfo endInfo{ m_info, m_assertions, m_timer.getElapsedSeconds() };
13174             if( uncaught_exceptions() )
13175                 getResultCapture().sectionEndedEarly( endInfo );
13176             else
13177                 getResultCapture().sectionEnded( endInfo );
13178         }
13179     }
13180
13181     // This indicates whether the section should be executed or not
13182     Section::operator bool() const {
13183         return m_sectionIncluded;
13184     }
13185
13186 } // end namespace Catch
13187 // end catch_section.cpp
13188 // start catch_section_info.cpp
13189

```

```

13190 namespace Catch {
13191
13192     SectionInfo::SectionInfo
13193     (   SourceLineInfo const& _lineInfo,
13194         std::string const& _name )
13195     :   name( _name ),
13196         lineInfo( _lineInfo )
13197     {}
13198
13199 } // end namespace Catch
13200 // end catch_section_info.cpp
13201 // start catch_session.cpp
13202
13203 // start catch_session.h
13204
13205 #include <memory>
13206
13207 namespace Catch {
13208
13209     class Session : NonCopyable {
13210     public:
13211
13212         Session();
13213         ~Session() override;
13214
13215         void showHelp() const;
13216         void libIdentify();
13217
13218         int applyCommandLine( int argc, char const * const * argv );
13219         #if defined(CATCH_CONFIG_WCHAR) && defined(_WIN32) && defined(UNICODE)
13220         int applyCommandLine( int argc, wchar_t const * const * argv );
13221         #endif
13222
13223         void useConfigData( ConfigData const& configData );
13224
13225         template<typename CharT>
13226         int run(int argc, CharT const * const argv[]) {
13227             if (m_startupExceptions)
13228                 return 1;
13229             int returnCode = applyCommandLine(argc, argv);
13230             if (returnCode == 0)
13231                 returnCode = run();
13232             return returnCode;
13233         }
13234
13235         int run();
13236
13237         clara::Parser const& cli() const;
13238         void cli( clara::Parser const& newParser );
13239         ConfigData& configData();
13240         Config& config();
13241     private:
13242         int runInternal();
13243
13244         clara::Parser m_cli;
13245         ConfigData m_configData;
13246         std::shared_ptr<Config> m_config;
13247         bool m_startupExceptions = false;
13248     };
13249
13250 } // end namespace Catch
13251
13252 // end catch_session.h
13253 // start catch_version.h
13254
13255 #include <iosfwd>
13256
13257 namespace Catch {
13258
13259     // Versioning information
13260     struct Version {
13261         Version( Version const& ) = delete;
13262         Version& operator=( Version const& ) = delete;
13263         Version(      unsigned int _majorVersion,
13264                     unsigned int _minorVersion,
13265                     unsigned int _patchNumber,
13266                     char const * _branchName,
13267                     unsigned int _buildNumber );
13268
13269         unsigned int const majorVersion;
13270         unsigned int const minorVersion;
13271         unsigned int const patchNumber;
13272
13273         // buildNumber is only used if branchName is not null
13274         char const * const branchName;
13275         unsigned int const buildNumber;
13276

```

```

13277         friend std::ostream& operator « ( std::ostream& os, Version const& version );
13278     };
13279
13280     Version const& libraryVersion();
13281 }
13282
13283 // end catch_version.h
13284 #include <cstdlib>
13285 #include <iomanip>
13286 #include <set>
13287 #include <iterator>
13288
13289 namespace Catch {
13290
13291     namespace {
13292         const int MaxExitCode = 255;
13293
13294         IStreamingReporterPtr createReporter(std::string const& reporterName, IConfigPtr const&
13295 config) {
13296             auto reporter = Catch::getRegistryHub().getReporterRegistry().create(reporterName,
13297 config);
13298             CATCH_ENFORCE(reporter, "No reporter registered with name: '" « reporterName « "'");
13299             return reporter;
13300         }
13301
13302         IStreamingReporterPtr makeReporter(std::shared_ptr<Config> const& config) {
13303             if (Catch::getRegistryHub().getReporterRegistry().getListeners().empty()) {
13304                 return createReporter(config->getReporterName(), config);
13305             }
13306             // On older platforms, returning std::unique_ptr<ListeningReporter>
13307             // when the return type is std::unique_ptr<IStreamingReporter>
13308             // doesn't compile without a std::move call. However, this causes
13309             // a warning on newer platforms. Thus, we have to work around
13310             // it a bit and downcast the pointer manually.
13311             auto ret = std::unique_ptr<IStreamingReporter>(new ListeningReporter);
13312             auto& multi = static_cast<ListeningReporter&>(*ret);
13313             auto const& listeners = Catch::getRegistryHub().getReporterRegistry().getListeners();
13314             for (auto const& listener : listeners) {
13315                 multi.addListener(listener->create(Catch::ReporterConfig(config)));
13316             }
13317             multi.addReporter(createReporter(config->getReporterName(), config));
13318             return ret;
13319         }
13320
13321         class TestGroup {
13322         public:
13323             explicit TestGroup(std::shared_ptr<Config> const& config)
13324                 : m_config{config}
13325                 , m_context{config, makeReporter(config)}
13326             {
13327                 auto const& allTestCases = getAllTestCasesSorted(*m_config);
13328                 m_matches = m_config->testSpec().matchesByFilter(allTestCases, *m_config);
13329                 auto const& invalidArgs = m_config->testSpec().getInvalidArgs();
13330
13331                 if (m_matches.empty() && invalidArgs.empty()) {
13332                     for (auto const& test : allTestCases)
13333                         if (!test.isHidden())
13334                             m_tests.emplace(&test);
13335                 } else {
13336                     for (auto const& match : m_matches)
13337                         m_tests.insert(match.tests.begin(), match.tests.end());
13338                 }
13339             }
13340
13341             Totals execute() {
13342                 auto const& invalidArgs = m_config->testSpec().getInvalidArgs();
13343                 Totals totals;
13344                 m_context.testGroupStarting(m_config->name(), 1, 1);
13345                 for (auto const& testCase : m_tests) {
13346                     if (!m_context.aborting())
13347                         totals += m_context.runTest(*testCase);
13348                     else
13349                         m_context.reporter().skipTest(*testCase);
13350                 }
13351
13352                 for (auto const& match : m_matches) {
13353                     if (match.tests.empty()) {
13354                         m_context.reporter().noMatchingTestCases(match.name);
13355                         totals.error = -1;
13356                     }
13357                 }
13358
13359                 if (!invalidArgs.empty()) {
13360                     for (auto const& invalidArg : invalidArgs)
13361                         m_context.reporter().reportInvalidArguments(invalidArg);
13362                 }
13363             }
13364         };

```

```

13362         }
13363     }
13364     m_context.testGroupEnded(m_config->name(), totals, 1, 1);
13365     return totals;
13366 }
13367
13368 private:
13369     using Tests = std::set<TestCase const*>;
13370
13371     std::shared_ptr<Config> m_config;
13372     RunContext m_context;
13373     Tests m_tests;
13374     TestSpec::Matches m_matches;
13375 };
13376
13377 void applyFileNamesAsTags(Catch::IConfig const& config) {
13378     auto& tests = const_cast<std::vector<TestCase*>>(getAllTestCasesSorted(config));
13379     for (auto& testCase : tests) {
13380         auto tags = testCase.tags;
13381
13382         std::string filename = testCase.lineInfo.file;
13383         auto lastSlash = filename.find_last_of("\\/");
13384         if (lastSlash != std::string::npos) {
13385             filename.erase(0, lastSlash);
13386             filename[0] = '#';
13387         }
13388
13389         auto lastDot = filename.find_last_of('.');
13390         if (lastDot != std::string::npos) {
13391             filename.erase(lastDot);
13392         }
13393
13394         tags.push_back(std::move(filename));
13395         setTags(testCase, tags);
13396     }
13397 }
13398
13399 } // anon namespace
13400
13401 Session::Session() {
13402     static bool alreadyInstantiated = false;
13403     if (alreadyInstantiated) {
13404         CATCH_TRY { CATCH_INTERNAL_ERROR( "Only one instance of Catch::Session can ever be used"
13405 ); }
13406         CATCH_CATCH_ALL { getMutableRegistryHub().registerStartupException(); }
13407     }
13408
13409     // There cannot be exceptions at startup in no-exception mode.
13410     #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
13411     const auto& exceptions = getRegistryHub().getStartupExceptionRegistry().getExceptions();
13412     if ( !exceptions.empty() ) {
13413         config();
13414         getCurrentMutableContext().setConfig(m_config);
13415
13416         m_startupExceptions = true;
13417         Colour colourGuard( Colour::Red );
13418         Catch::cerr() << "Errors occurred during startup!" << '\n';
13419         // iterate over all exceptions and notify user
13420         for ( const auto& ex_ptr : exceptions ) {
13421             try {
13422                 std::rethrow_exception(ex_ptr);
13423             } catch ( std::exception const& ex ) {
13424                 Catch::cerr() << Column( ex.what() ).indent(2) << '\n';
13425             }
13426         }
13427     }
13428     #endif
13429
13430     alreadyInstantiated = true;
13431     m_cli = makeCommandLineParser( m_configData );
13432 }
13433
13434 Session::~~Session() {
13435     Catch::cleanUp();
13436 }
13437
13438 void Session::showHelp() const {
13439     Catch::cout()
13440         << "\nCatch v" << libraryVersion() << "\n"
13441         << m_cli << std::endl
13442         << "For more detailed usage please see the project docs\n" << std::endl;
13443 }
13444
13445 void Session::libIdentify() {
13446     Catch::cout()
13447         << std::left << std::setw(16) << "description: " << "A Catch2 test executable\n"
13448         << std::left << std::setw(16) << "category: " << "testframework\n"
13449         << std::left << std::setw(16) << "framework: " << "Catch Test\n"
13450         << std::left << std::setw(16) << "version: " << libraryVersion() << std::endl;

```

```

13448     }
13449
13450     int Session::applyCommandLine( int argc, char const * const * argv ) {
13451         if( m_startupExceptions )
13452             return 1;
13453
13454         auto result = m_cli.parse( clara::Args( argc, argv ) );
13455         if( !result ) {
13456             config();
13457             getCurrentMutableContext().setConfig(m_config);
13458             Catch::cerr()
13459                 « Colour( Colour::Red )
13460                 « "\nError(s) in input:\n"
13461                 « Column( result.errorMessage() ).indent( 2 )
13462                 « "\n\n";
13463             Catch::cerr() « "Run with -? for usage\n" « std::endl;
13464             return MaxExitCode;
13465         }
13466
13467         if( m_configData.showHelp )
13468             showHelp();
13469         if( m_configData.libIdentify )
13470             libIdentify();
13471         m_config.reset();
13472         return 0;
13473     }
13474
13475 #if defined(CATCH_CONFIG_WCHAR) && defined(_WIN32) && defined(UNICODE)
13476     int Session::applyCommandLine( int argc, wchar_t const * const * argv ) {
13477
13478         char **utf8Argv = new char * [ argc ];
13479
13480         for ( int i = 0; i < argc; ++i ) {
13481             int bufSize = WideCharToMultiByte( CP_UTF8, 0, argv[i], -1, nullptr, 0, nullptr, nullptr
13482 );
13483             utf8Argv[ i ] = new char[ bufSize ];
13484
13485             WideCharToMultiByte( CP_UTF8, 0, argv[i], -1, utf8Argv[i], bufSize, nullptr, nullptr );
13486         }
13487
13488         int returnCode = applyCommandLine( argc, utf8Argv );
13489
13490         for ( int i = 0; i < argc; ++i )
13491             delete [] utf8Argv[ i ];
13492
13493         delete [] utf8Argv;
13494
13495         return returnCode;
13496     }
13497 #endif
13498
13499     void Session::useConfigData( ConfigData const& configData ) {
13500         m_configData = configData;
13501         m_config.reset();
13502     }
13503
13504     int Session::run() {
13505         if( ( m_configData.waitForKeypress & WaitForKeypress::BeforeStart ) != 0 ) {
13506             Catch::cout() « "...waiting for enter/ return before starting" « std::endl;
13507             static_cast<void>(std::getchar());
13508         }
13509         int exitCode = runInternal();
13510         if( ( m_configData.waitForKeypress & WaitForKeypress::BeforeExit ) != 0 ) {
13511             Catch::cout() « "...waiting for enter/ return before exiting, with code: " « exitCode «
std::endl;
13512             static_cast<void>(std::getchar());
13513         }
13514         return exitCode;
13515     }
13516
13517     clara::Parser const& Session::cli() const {
13518         return m_cli;
13519     }
13520     void Session::cli( clara::Parser const& newParser ) {
13521         m_cli = newParser;
13522     }
13523     ConfigData& Session::configData() {
13524         return m_configData;
13525     }
13526     Config& Session::config() {
13527         if( !m_config )
13528             m_config = std::make_shared<Config>( m_configData );
13529         return *m_config;
13530     }
13531
13532     int Session::runInternal() {

```

```

13533         if( m_startupExceptions )
13534             return 1;
13535
13536         if (m_configData.showHelp || m_configData.libIdentify) {
13537             return 0;
13538         }
13539
13540         CATCH_TRY {
13541             config(); // Force config to be constructed
13542
13543             seedRng( *m_config );
13544
13545             if( m_configData_filenamesAsTags )
13546                 applyFilenamesAsTags( *m_config );
13547
13548             // Handle list request
13549             if( Option<std::size_t> listed = list( m_config ) )
13550                 return static_cast<int>( *listed );
13551
13552             TestGroup tests { m_config };
13553             auto const totals = tests.execute();
13554
13555             if( m_config->warnAboutNoTests() && totals.error == -1 )
13556                 return 2;
13557
13558             // Note that on unices only the lower 8 bits are usually used, clamping
13559             // the return value to 255 prevents false negative when some multiple
13560             // of 256 tests has failed
13561             return (std::min)(MaxExitCode, (std::max)(totals.error,
13562                 static_cast<int>(totals.assertions.failed)));
13563         }
13564         #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
13565         catch( std::exception& ex ) {
13566             Catch::cerr() << ex.what() << std::endl;
13567             return MaxExitCode;
13568         }
13569         #endif
13570     } // end namespace Catch
13571 } // end catch_session.cpp
13572 // start catch_singletons.cpp
13573 // start catch_singletons.cpp
13574
13575 #include <vector>
13576
13577 namespace Catch {
13578
13579     namespace {
13580         static auto getSingletons() -> std::vector<ISingleton*>* & {
13581             static std::vector<ISingleton*>* g_singletons = nullptr;
13582             if( !g_singletons )
13583                 g_singletons = new std::vector<ISingleton*>();
13584             return g_singletons;
13585         }
13586     }
13587
13588     ISingleton::~ISingleton() {}
13589
13590     void addSingleton(ISingleton* singleton) {
13591         getSingletons()->push_back( singleton );
13592     }
13593
13594     void cleanupSingletons() {
13595         auto& singletons = getSingletons();
13596         for( auto singleton : *singletons )
13597             delete singleton;
13598         singletons = nullptr;
13599     }
13600 } // namespace Catch
13601 // end catch_singletons.cpp
13602 // start catch_startup_exception_registry.cpp
13603
13604 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
13605 namespace Catch {
13606 void StartupExceptionRegistry::add( std::exception_ptr const& exception ) noexcept {
13607     CATCH_TRY {
13608         m_exceptions.push_back(exception);
13609     } CATCH_CATCH_ALL {
13610         // If we run out of memory during start-up there's really not a lot more we can do about
13611         it
13612         std::terminate();
13613     }
13614 }
13615
13616 std::vector<std::exception_ptr> const& StartupExceptionRegistry::getExceptions() const noexcept {
13617     return m_exceptions;

```

```

13618     }
13619
13620 } // end namespace Catch
13621 #endif
13622 // end catch_startup_exception_registry.cpp
13623 // start catch_stream.cpp
13624
13625 #include <cstdio>
13626 #include <iostream>
13627 #include <fstream>
13628 #include <sstream>
13629 #include <vector>
13630 #include <memory>
13631
13632 namespace Catch {
13633
13634     Catch::IStream::~IStream() = default;
13635
13636     namespace Detail { namespace {
13637         template<typename WriterF, std::size_t bufferSize=256>
13638         class StreamBufImpl : public std::streambuf {
13639             char data[bufferSize];
13640             WriterF m_writer;
13641
13642         public:
13643             StreamBufImpl() {
13644                 setp( data, data + sizeof(data) );
13645             }
13646
13647             ~StreamBufImpl() noexcept {
13648                 StreamBufImpl::sync();
13649             }
13650
13651         private:
13652             int overflow( int c ) override {
13653                 sync();
13654
13655                 if( c != EOF ) {
13656                     if( pbase() == epptr() )
13657                         m_writer( std::string( 1, static_cast<char>( c ) ) );
13658                     else
13659                         sputc( static_cast<char>( c ) );
13660                 }
13661                 return 0;
13662             }
13663
13664             int sync() override {
13665                 if( pbase() != pptr() ) {
13666                     m_writer( std::string( pbase(), static_cast<std::string::size_type>( pptr() -
13667 pbase() ) ) );
13667                     setp( pbase(), epptr() );
13668                 }
13669                 return 0;
13670             }
13671         };
13672
13673         struct OutputDebugWriter {
13674
13675             void operator()( std::string const&str ) {
13676                 writeToDebugConsole( str );
13677             }
13678         };
13679     };
13680
13681     class FileStream : public IStream {
13682     mutable std::ofstream m_ofs;
13683     public:
13684         FileStream( StringRef filename ) {
13685             m_ofs.open( filename.c_str() );
13686             CATCH_ENFORCE( !m_ofs.fail(), "Unable to open file: '" << filename << "'" );
13687         }
13688         ~FileStream() override = default;
13689     public: // IStream
13690         std::ostream& stream() const override {
13691             return m_ofs;
13692         }
13693     };
13694
13695     class CoutStream : public IStream {
13696     mutable std::ostream m_os;
13697     public:
13698         // Store the streambuf from cout up-front because
13699         // cout may get redirected when running tests
13700         CoutStream() : m_os( Catch::cout().rdbuf() ) {}
13701         ~CoutStream() override = default;
13702     public:

```



```

13707
13708     public: // IStream
13709         std::ostream& stream() const override { return m_os; }
13710     };
13711
13712
13713
13714     class DebugOutputStream : public IStream {
13715     public:
13716         std::unique_ptr<StreamBufImpl<OutputDebugWriter> m_streamBuf;
13717         mutable std::ostream m_os;
13718     public:
13719         DebugOutputStream()
13720         : m_streamBuf( new StreamBufImpl<OutputDebugWriter>() ),
13721           m_os( m_streamBuf.get() )
13722         {}
13723
13724         ~DebugOutputStream() override = default;
13725
13726     public: // IStream
13727         std::ostream& stream() const override { return m_os; }
13728     };
13729
13730 } // namespace anon::detail
13731
13732
13733 auto makeStream( StringRef const &filename ) -> IStream const* {
13734     if( filename.empty() )
13735         return new Detail::CoutStream();
13736     else if( filename[0] == '%' ) {
13737         if( filename == "%debug" )
13738             return new Detail::DebugOutputStream();
13739         else
13740             CATCH_ERROR( "Unrecognised stream: '" < filename < "'" );
13741     }
13742     else
13743         return new Detail::FileStream( filename );
13744 }
13745
13746 // This class encapsulates the idea of a pool of ostringstreams that can be reused.
13747 struct StringStreams {
13748     std::vector<std::unique_ptr<std::ostringstream> m_streams;
13749     std::vector<std::size_t> m_unused;
13750     std::ostringstream m_referenceStream; // Used for copy state/ flags from
13751
13752     auto add() -> std::size_t {
13753         if( m_unused.empty() ) {
13754             m_streams.push_back( std::unique_ptr<std::ostringstream>( new std::ostringstream ) );
13755             return m_streams.size()-1;
13756         }
13757         else {
13758             auto index = m_unused.back();
13759             m_unused.pop_back();
13760             return index;
13761         }
13762     }
13763
13764     void release( std::size_t index ) {
13765         m_streams[index]->copyfmt( m_referenceStream ); // Restore initial flags and other state
13766         m_unused.push_back(index);
13767     }
13768 };
13769
13770 ReusableStringStream::ReusableStringStream()
13771 : m_index( Singleton<StringStreams>::getMutable().add() ),
13772   m_oss( Singleton<StringStreams>::getMutable().m_streams[m_index].get() )
13773 {}
13774
13775 ReusableStringStream::~ReusableStringStream() {
13776     static_cast<std::ostringstream*>( m_oss )->str("");
13777     m_oss->clear();
13778     Singleton<StringStreams>::getMutable().release( m_index );
13779 }
13780
13781 auto ReusableStringStream::str() const -> std::string {
13782     return static_cast<std::ostringstream*>( m_oss )->str();
13783 }
13784
13785
13786 #ifndef CATCH_CONFIG_NOSTDOUT // If you #define this you must implement these functions
13787     std::ostream& cout() { return std::cout; }
13788     std::ostream& cerr() { return std::cerr; }
13789     std::ostream& clog() { return std::clog; }
13790 #endif
13791
13792 // end catch_stream.cpp
13793 // start catch_string_manip.cpp
13794
13795 #include <algorithm>

```

```

13797 #include <ostream>
13798 #include <cstring>
13799 #include <cctype>
13800 #include <vector>
13801
13802 namespace Catch {
13803
13804     namespace {
13805         char toLowerCh(char c) {
13806             return static_cast<char>( std::tolower( static_cast<unsigned char>(c) ) );
13807         }
13808     }
13809
13810     bool startsWith( std::string const& s, std::string const& prefix ) {
13811         return s.size() >= prefix.size() && std::equal(prefix.begin(), prefix.end(), s.begin());
13812     }
13813     bool startsWith( std::string const& s, char prefix ) {
13814         return !s.empty() && s[0] == prefix;
13815     }
13816     bool endsWith( std::string const& s, std::string const& suffix ) {
13817         return s.size() >= suffix.size() && std::equal(suffix.rbegin(), suffix.rend(), s.rbegin());
13818     }
13819     bool endsWith( std::string const& s, char suffix ) {
13820         return !s.empty() && s[s.size()-1] == suffix;
13821     }
13822     bool contains( std::string const& s, std::string const& infix ) {
13823         return s.find( infix ) != std::string::npos;
13824     }
13825     void toLowerInPlace( std::string& s ) {
13826         std::transform( s.begin(), s.end(), s.begin(), toLowerCh );
13827     }
13828     std::string toLower( std::string const& s ) {
13829         std::string lc = s;
13830         toLowerInPlace( lc );
13831         return lc;
13832     }
13833     std::string trim( std::string const& str ) {
13834         static char const* whitespaceChars = "\n\r\t ";
13835         std::string::size_type start = str.find_first_not_of( whitespaceChars );
13836         std::string::size_type end = str.find_last_not_of( whitespaceChars );
13837
13838         return start != std::string::npos ? str.substr( start, 1+end-start ) : std::string();
13839     }
13840
13841     StringRef trim(StringRef ref) {
13842         const auto is_ws = [](char c) {
13843             return c == ' ' || c == '\t' || c == '\n' || c == '\r';
13844         };
13845         size_t real_begin = 0;
13846         while (real_begin < ref.size() && is_ws(ref[real_begin])) { ++real_begin; }
13847         size_t real_end = ref.size();
13848         while (real_end > real_begin && is_ws(ref[real_end - 1])) { --real_end; }
13849
13850         return ref.substr(real_begin, real_end - real_begin);
13851     }
13852
13853     bool replaceInPlace( std::string& str, std::string const& replaceThis, std::string const& withThis
13854 ) {
13855         bool replaced = false;
13856         std::size_t i = str.find( replaceThis );
13857         while( i != std::string::npos ) {
13858             replaced = true;
13859             str = str.substr( 0, i ) + withThis + str.substr( i+replaceThis.size() );
13860             if( i < str.size()-withThis.size() )
13861                 i = str.find( replaceThis, i+withThis.size() );
13862             else
13863                 i = std::string::npos;
13864         }
13865         return replaced;
13866     }
13867
13868     std::vector<StringRef> splitStringRef( StringRef str, char delimiter ) {
13869         std::vector<StringRef> subStrings;
13870         std::size_t start = 0;
13871         for( std::size_t pos = 0; pos < str.size(); ++pos ) {
13872             if( str[pos] == delimiter ) {
13873                 if( pos - start > 1 )
13874                     subStrings.push_back( str.substr( start, pos-start ) );
13875                 start = pos+1;
13876             }
13877         }
13878         if( start < str.size() )
13879             subStrings.push_back( str.substr( start, str.size()-start ) );
13880         return subStrings;
13881     }
13882
13883     pluralise::pluralise( std::size_t count, std::string const& label )

```

```

13883     :   m_count( count ),
13884       m_label( label )
13885     {}
13886
13887     std::ostream& operator << ( std::ostream& os, pluralise const& pluraliser ) {
13888         os << pluraliser.m_count << ' ' << pluraliser.m_label;
13889         if( pluraliser.m_count != 1 )
13890             os << 's';
13891         return os;
13892     }
13893
13894 }
13895 // end catch_string_manip.cpp
13896 // start catch_stringref.cpp
13897
13898 #include <algorithm>
13899 #include <ostream>
13900 #include <cstring>
13901 #include <cstdint>
13902
13903 namespace Catch {
13904     StringRef::StringRef( char const* rawChars ) noexcept
13905     : StringRef( rawChars, static_cast<StringRef::size_type>(std::strlen(rawChars)) )
13906     {}
13907
13908     auto StringRef::c_str() const -> char const* {
13909         CATCH_ENFORCE(isNullTerminated(), "Called StringRef::c_str() on a non-null-terminated
instance");
13910         return m_start;
13911     }
13912     auto StringRef::data() const noexcept -> char const* {
13913         return m_start;
13914     }
13915
13916     auto StringRef::substr( size_type start, size_type size ) const noexcept -> StringRef {
13917         if (start < m_size) {
13918             return StringRef(m_start + start, (std::min)(m_size - start, size));
13919         } else {
13920             return StringRef();
13921         }
13922     }
13923     auto StringRef::operator == ( StringRef const& other ) const noexcept -> bool {
13924         return m_size == other.m_size
13925             && (std::memcmp( m_start, other.m_start, m_size ) == 0);
13926     }
13927
13928     auto operator << ( std::ostream& os, StringRef const& str ) -> std::ostream& {
13929         return os.write(str.data(), str.size());
13930     }
13931
13932     auto operator+=( std::string& lhs, StringRef const& rhs ) -> std::string& {
13933         lhs.append(rhs.data(), rhs.size());
13934         return lhs;
13935     }
13936
13937 } // namespace Catch
13938 // end catch_stringref.cpp
13939 // start catch_tag_alias.cpp
13940
13941 namespace Catch {
13942     TagAlias::TagAlias(std::string const & _tag, SourceLineInfo _lineInfo): tag(_tag),
lineInfo(_lineInfo) {}
13943 }
13944 // end catch_tag_alias.cpp
13945 // start catch_tag_alias_autoregistrar.cpp
13946
13947 namespace Catch {
13948
13949     RegistrarForTagAliases::RegistrarForTagAliases(char const* alias, char const* tag, SourceLineInfo
const& lineInfo) {
13950         CATCH_TRY {
13951             getMutableRegistryHub().registerTagAlias(alias, tag, lineInfo);
13952         } CATCH_CATCH_ALL {
13953             // Do not throw when constructing global objects, instead register the exception to be
processed later
13954             getMutableRegistryHub().registerStartupException();
13955         }
13956     }
13957
13958 }
13959 // end catch_tag_alias_autoregistrar.cpp
13960 // start catch_tag_alias_registry.cpp
13961
13962 #include <sstream>
13963
13964 namespace Catch {
13965

```

```

13966     TagAliasRegistry::~TagAliasRegistry() {}
13967
13968     TagAlias const* TagAliasRegistry::find( std::string const& alias ) const {
13969         auto it = m_registry.find( alias );
13970         if( it != m_registry.end() )
13971             return &(it->second);
13972         else
13973             return nullptr;
13974     }
13975
13976     std::string TagAliasRegistry::expandAliases( std::string const& unexpandedTestSpec ) const {
13977         std::string expandedTestSpec = unexpandedTestSpec;
13978         for( auto const& registryKvp : m_registry ) {
13979             std::size_t pos = expandedTestSpec.find( registryKvp.first );
13980             if( pos != std::string::npos ) {
13981                 expandedTestSpec = expandedTestSpec.substr( 0, pos ) +
13982                                     registryKvp.second.tag +
13983                                     expandedTestSpec.substr( pos + registryKvp.first.size() );
13984             }
13985         }
13986         return expandedTestSpec;
13987     }
13988
13989     void TagAliasRegistry::add( std::string const& alias, std::string const& tag, SourceLineInfo
const& lineInfo ) {
13990         CATCH_ENFORCE( startsWith(alias, "[@") && endsWith(alias, ']'),
13991             "error: tag alias, '" < alias < "' is not of the form [@alias name].\n" <
lineInfo );
13992
13993         CATCH_ENFORCE( m_registry.insert(std::make_pair(alias, TagAlias(tag, lineInfo))).second,
13994             "error: tag alias, '" < alias < "' already registered.\n"
13995             < "\tFirst seen at: " < find(alias)->lineInfo < "\n"
13996             < "\tRedefined at: " < lineInfo );
13997     }
13998
13999     ITagAliasRegistry::~ITagAliasRegistry() {}
14000
14001     ITagAliasRegistry const& ITagAliasRegistry::get() {
14002         return getRegistryHub().getTagAliasRegistry();
14003     }
14004
14005 } // end namespace Catch
14006 // end catch_tag_alias_registry.cpp
14007 // start catch_test_case_info.cpp
14008
14009 #include <cctype>
14010 #include <exception>
14011 #include <algorithm>
14012 #include <sstream>
14013
14014 namespace Catch {
14015     namespace {
14016         TestCaseInfo::SpecialProperties parseSpecialTag( std::string const& tag ) {
14017             if( startsWith( tag, '.' ) ||
14018                 tag == "!hide" )
14019                 return TestCaseInfo::IsHidden;
14020             else if( tag == "!throws" )
14021                 return TestCaseInfo::Throws;
14022             else if( tag == "!shouldfail" )
14023                 return TestCaseInfo::ShouldFail;
14024             else if( tag == "!mayfail" )
14025                 return TestCaseInfo::MayFail;
14026             else if( tag == "!nonportable" )
14027                 return TestCaseInfo::NonPortable;
14028             else if( tag == "!benchmark" )
14029                 return TestCaseInfo::Benchmark;
14030             return static_cast<TestCaseInfo::SpecialProperties>( TestCaseInfo::Benchmark |
14031                 TestCaseInfo::IsHidden );
14032             else
14033                 return TestCaseInfo::None;
14034         }
14035         bool isReservedTag( std::string const& tag ) {
14036             return parseSpecialTag( tag ) == TestCaseInfo::None && tag.size() > 0 && !std::isalnum(
14037                 static_cast<unsigned char>(tag[0]) );
14038         }
14039         void enforceNotReservedTag( std::string const& tag, SourceLineInfo const& _lineInfo ) {
14040             CATCH_ENFORCE( !isReservedTag(tag),
14041                 "Tag name: [" < tag < "] is not allowed.\n"
14042                 < "Tag names starting with non alphanumeric characters are reserved\n"
14043                 < _lineInfo );
14044         }
14045         TestCase makeTestCase( ITestInvoker* _testCase,
14046             std::string const& _className,
14047             NameAndTags const& nameAndTags,
14048             SourceLineInfo const& _lineInfo )

```

```

14049     {
14050         bool isHidden = false;
14051
14052         // Parse out tags
14053         std::vector<std::string> tags;
14054         std::string desc, tag;
14055         bool inTag = false;
14056         for (char c : nameAndTags.tags) {
14057             if( !inTag ) {
14058                 if( c == '[' )
14059                     inTag = true;
14060                 else
14061                     desc += c;
14062             }
14063             else {
14064                 if( c == ']' ) {
14065                     TestCaseInfo::SpecialProperties prop = parseSpecialTag( tag );
14066                     if( ( prop & TestCaseInfo::IsHidden ) != 0 )
14067                         isHidden = true;
14068                     else if( prop == TestCaseInfo::None )
14069                         enforceNotReservedTag( tag, _lineInfo );
14070
14071                     // Merged hide tags like `[.approvals]` should be added as
14072                     // `[.][approvals]`. The `[.]` is added at later point, so
14073                     // we only strip the prefix
14074                     if (startsWith(tag, '.') && tag.size() > 1) {
14075                         tag.erase(0, 1);
14076                     }
14077                     tags.push_back( tag );
14078                     tag.clear();
14079                     inTag = false;
14080                 }
14081                 else
14082                     tag += c;
14083             }
14084         }
14085         if( isHidden ) {
14086             // Add all "hidden" tags to make them behave identically
14087             tags.insert( tags.end(), { ".", "!hide" } );
14088         }
14089
14090         TestCaseInfo info( static_cast<std::string>(nameAndTags.name), _className, desc, tags,
14091 _lineInfo );
14092         return TestCase( _testCase, std::move(info) );
14093     }
14094
14095     void setTags( TestCaseInfo& testCaseInfo, std::vector<std::string> tags ) {
14096         std::sort(begin(tags), end(tags));
14097         tags.erase(std::unique(begin(tags), end(tags)), end(tags));
14098         testCaseInfo.lcaseTags.clear();
14099
14100         for( auto const& tag : tags ) {
14101             std::string lcaseTag = toLower( tag );
14102             testCaseInfo.properties = static_cast<TestCaseInfo::SpecialProperties>(
14103                 testCaseInfo.properties | parseSpecialTag( lcaseTag ) );
14104             testCaseInfo.lcaseTags.push_back( lcaseTag );
14105         }
14106         testCaseInfo.tags = std::move(tags);
14107     }
14108
14109     TestCaseInfo::TestCaseInfo( std::string const& _name,
14110                                std::string const& _className,
14111                                std::string const& _description,
14112                                std::vector<std::string> const& _tags,
14113                                SourceLineInfo const& _lineInfo )
14114     :   name( _name ),
14115         className( _className ),
14116         description( _description ),
14117         lineInfo( _lineInfo ),
14118         properties( None )
14119     {
14120         setTags( *this, _tags );
14121     }
14122
14123     bool TestCaseInfo::isHidden() const {
14124         return ( properties & IsHidden ) != 0;
14125     }
14126
14127     bool TestCaseInfo::throws() const {
14128         return ( properties & Throws ) != 0;
14129     }
14130
14131     bool TestCaseInfo::okToFail() const {
14132         return ( properties & (ShouldFail | MayFail) ) != 0;
14133     }
14134
14135     bool TestCaseInfo::expectedToFail() const {
14136         return ( properties & (ShouldFail) ) != 0;
14137     }

```

```

14134     std::string TestCaseInfo::tagsAsString() const {
14135         std::string ret;
14136         // '[' and ']' per tag
14137         std::size_t full_size = 2 * tags.size();
14138         for (const auto& tag : tags) {
14139             full_size += tag.size();
14140         }
14141         ret.reserve(full_size);
14142         for (const auto& tag : tags) {
14143             ret.push_back('[');
14144             ret.append(tag);
14145             ret.push_back(']');
14146         }
14147
14148         return ret;
14149     }
14150
14151     TestCase::TestCase( ITestInvoker* testCase, TestCaseInfo&& info ) : TestCaseInfo( std::move(info)
), test( testCase ) {}
14152
14153     TestCase TestCase::withName( std::string const& _newName ) const {
14154         TestCase other( *this );
14155         other.name = _newName;
14156         return other;
14157     }
14158
14159     void TestCase::invoke() const {
14160         test->invoke();
14161     }
14162
14163     bool TestCase::operator == ( TestCase const& other ) const {
14164         return test.get() == other.test.get() &&
14165             name == other.name &&
14166             className == other.className;
14167     }
14168
14169     bool TestCase::operator < ( TestCase const& other ) const {
14170         return name < other.name;
14171     }
14172
14173     TestCaseInfo const& TestCase::getTestCaseInfo() const
14174     {
14175         return *this;
14176     }
14177
14178 } // end namespace Catch
14179 // end catch_test_case_info.cpp
14180 // start catch_test_case_registry_impl.cpp
14181
14182 #include <algorithm>
14183 #include <sstream>
14184
14185 namespace Catch {
14186
14187     namespace {
14188         struct TestHasher {
14189             using hash_t = uint64_t;
14190
14191             explicit TestHasher( hash_t hashSuffix ):
14192                 m_hashSuffix{ hashSuffix } {}
14193
14194             uint32_t operator()( TestCase const& t ) const {
14195                 // FNV-1a hash with multiplication fold.
14196                 const hash_t prime = 1099511628211u;
14197                 hash_t hash = 14695981039346656037u;
14198                 for ( const char c : t.name ) {
14199                     hash ^= c;
14200                     hash *= prime;
14201                 }
14202                 hash ^= m_hashSuffix;
14203                 hash *= prime;
14204                 const uint32_t low{ static_cast<uint32_t>( hash ) };
14205                 const uint32_t high{ static_cast<uint32_t>( hash >> 32 ) };
14206                 return low * high;
14207             }
14208
14209             private:
14210                 hash_t m_hashSuffix;
14211         };
14212     } // end unnamed namespace
14213
14214     std::vector<TestCase> sortTests( IConfig const& config, std::vector<TestCase> const&
unsortedTestCases ) {
14215         switch( config.runOrder() ) {
14216             case RunTests::InDeclarationOrder:
14217                 // already in declaration order
14218                 break;

```

```

14219
14220         case RunTests::InLexicographicalOrder: {
14221             std::vector<TestCase> sorted = unsortedTestCases;
14222             std::sort( sorted.begin(), sorted.end() );
14223             return sorted;
14224         }
14225
14226         case RunTests::InRandomOrder: {
14227             seedRng( config );
14228             TestHasher h{ config.rngSeed() };
14229
14230             using hashedTest = std::pair<TestHasher::hash_t, TestCase const*>;
14231             std::vector<hashedTest> indexed_tests;
14232             indexed_tests.reserve( unsortedTestCases.size() );
14233
14234             for (auto const& testCase : unsortedTestCases) {
14235                 indexed_tests.emplace_back(h(testCase), &testCase);
14236             }
14237
14238             std::sort(indexed_tests.begin(), indexed_tests.end(),
14239                 [](hashedTest const& lhs, hashedTest const& rhs) {
14240                     if (lhs.first == rhs.first) {
14241                         return lhs.second->name < rhs.second->name;
14242                     }
14243                     return lhs.first < rhs.first;
14244                 });
14245
14246             std::vector<TestCase> sorted;
14247             sorted.reserve( indexed_tests.size() );
14248
14249             for (auto const& hashed : indexed_tests) {
14250                 sorted.emplace_back(*hashed.second);
14251             }
14252
14253             return sorted;
14254         }
14255     }
14256     return unsortedTestCases;
14257 }
14258
14259 bool isThrowSafe( TestCase const& testCase, IConfig const& config ) {
14260     return !testCase.throw() || config.allowThrows();
14261 }
14262
14263 bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config ) {
14264     return testSpec.matches( testCase ) && isThrowSafe( testCase, config );
14265 }
14266
14267 void enforceNoDuplicateTestCases( std::vector<TestCase> const& functions ) {
14268     std::set<TestCase> seenFunctions;
14269     for( auto const& function : functions ) {
14270         auto prev = seenFunctions.insert( function );
14271         CATCH_ENFORCE( prev.second,
14272             "error: TEST_CASE( \"" << function.name << "\" ) already defined.\n"
14273             << "\tFirst seen at " << prev.first->getTestCaseInfo().lineInfo << "\n"
14274             << "\tRedefined at " << function.getTestCaseInfo().lineInfo );
14275     }
14276 }
14277
14278 std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
testSpec, IConfig const& config ) {
14279     std::vector<TestCase> filtered;
14280     filtered.reserve( testCases.size() );
14281     for (auto const& testCase : testCases) {
14282         if ( (!testSpec.hasFilters() && !testCase.isHidden()) ||
14283             (testSpec.hasFilters() && matchTest(testCase, testSpec, config)) ) {
14284             filtered.push_back(testCase);
14285         }
14286     }
14287     return filtered;
14288 }
14289
14290 std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config ) {
14291     return getRegistryHub().getTestCaseRegistry().getAllTestsSorted( config );
14292 }
14293
14294 void TestRegistry::registerTest( TestCase const& testCase ) {
14295     std::string name = testCase.getTestCaseInfo().name;
14296     if( name.empty() ) {
14297         ReusableStringStream rss;
14298         rss << "Anonymous test case " << ++m_unnamedCount;
14299         return registerTest( testCase.withName( rss.str() ) );
14300     }
14301     m_functions.push_back( testCase );
14302 }
14303
14304 std::vector<TestCase> const& TestRegistry::getAllTests() const {
14305     return m_functions;
14306 }

```

```

14305     }
14306     std::vector<TestCase> const& TestRegistry::getAllTestsSorted( IConfig const& config ) const {
14307         if( m_sortedFunctions.empty() )
14308             enforceNoDuplicateTestCases( m_functions );
14309
14310         if( m_currentSortOrder != config.runOrder() || m_sortedFunctions.empty() ) {
14311             m_sortedFunctions = sortTests( config, m_functions );
14312             m_currentSortOrder = config.runOrder();
14313         }
14314         return m_sortedFunctions;
14315     }
14316
14318     TestInvokerAsFunction::TestInvokerAsFunction( void(*testAsFunction)() ) noexcept :
14319     m_testAsFunction( testAsFunction ) {}
14320
14320     void TestInvokerAsFunction::invoke() const {
14321         m_testAsFunction();
14322     }
14323
14324     std::string extractClassName( StringRef const& classOrQualifiedMethodName ) {
14325         std::string className(classOrQualifiedMethodName);
14326         if( startsWith( className, '&' ) )
14327         {
14328             std::size_t lastColons = className.rfind( "::" );
14329             std::size_t penultimateColons = className.rfind( ":", lastColons-1 );
14330             if( penultimateColons == std::string::npos )
14331                 penultimateColons = 1;
14332             className = className.substr( penultimateColons, lastColons-penultimateColons );
14333         }
14334         return className;
14335     }
14336
14337 } // end namespace Catch
14338 // end catch_test_case_registry_impl.cpp
14339 // start catch_test_case_tracker.cpp
14340
14341 #include <algorithm>
14342 #include <cassert>
14343 #include <stdexcept>
14344 #include <memory>
14345 #include <sstream>
14346
14347 #if defined(__clang__)
14348 #   pragma clang diagnostic push
14349 #   pragma clang diagnostic ignored "-Wexit-time-destructors"
14350 #endif
14351
14352 namespace Catch {
14353 namespace TestCaseTracking {
14354
14355     NameAndLocation::NameAndLocation( std::string const& _name, SourceLineInfo const& _location )
14356     :   name( _name ),
14357         location( _location )
14358     {}
14359
14360     ITracker::~ITracker() = default;
14361
14362     ITracker& TrackerContext::startRun() {
14363         m_rootTracker = std::make_shared<SectionTracker>( NameAndLocation( "{root}",
14364             CATCH_INTERNAL_LINEINFO ), *this, nullptr );
14365         m_currentTracker = nullptr;
14366         m_runState = Executing;
14367         return *m_rootTracker;
14368     }
14369
14370     void TrackerContext::endRun() {
14371         m_rootTracker.reset();
14372         m_currentTracker = nullptr;
14373         m_runState = NotStarted;
14374     }
14375
14376     void TrackerContext::startCycle() {
14377         m_currentTracker = m_rootTracker.get();
14378         m_runState = Executing;
14379     }
14380
14381     void TrackerContext::completeCycle() {
14382         m_runState = CompletedCycle;
14383     }
14384
14385     bool TrackerContext::completedCycle() const {
14386         return m_runState == CompletedCycle;
14387     }
14388
14389     ITracker& TrackerContext::currentTracker() {
14390         return *m_currentTracker;
14391     }
14392
14393     void TrackerContext::setCurrentTracker( ITracker* tracker ) {
14394         m_currentTracker = tracker;
14395     }

```



```

14391     }
14392
14393     TrackerBase::TrackerBase( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker*
parent ):
14394         ITracker(nameAndLocation),
14395         m_ctx( ctx ),
14396         m_parent( parent )
14397     {}
14398
14399     bool TrackerBase::isComplete() const {
14400         return m_runState == CompletedSuccessfully || m_runState == Failed;
14401     }
14402     bool TrackerBase::isSuccessfullyCompleted() const {
14403         return m_runState == CompletedSuccessfully;
14404     }
14405     bool TrackerBase::isOpen() const {
14406         return m_runState != NotStarted && !isComplete();
14407     }
14408     bool TrackerBase::hasChildren() const {
14409         return !m_children.empty();
14410     }
14411
14412     void TrackerBase::addChild( ITrackerPtr const& child ) {
14413         m_children.push_back( child );
14414     }
14415
14416     ITrackerPtr TrackerBase::findChild( NameAndLocation const& nameAndLocation ) {
14417         auto it = std::find_if( m_children.begin(), m_children.end(),
14418             [&nameAndLocation]( ITrackerPtr const& tracker ) {
14419             return
14420                 tracker->nameAndLocation().location == nameAndLocation.location &&
14421                 tracker->nameAndLocation().name == nameAndLocation.name;
14422             } );
14423         return( it != m_children.end() )
14424             ? *it
14425             : nullptr;
14426     }
14427     ITracker& TrackerBase::parent() {
14428         assert( m_parent ); // Should always be non-null except for root
14429         return *m_parent;
14430     }
14431
14432     void TrackerBase::openChild() {
14433         if( m_runState != ExecutingChildren ) {
14434             m_runState = ExecutingChildren;
14435             if( m_parent )
14436                 m_parent->openChild();
14437         }
14438     }
14439
14440     bool TrackerBase::isSectionTracker() const { return false; }
14441     bool TrackerBase::isGeneratorTracker() const { return false; }
14442
14443     void TrackerBase::open() {
14444         m_runState = Executing;
14445         moveToThis();
14446         if( m_parent )
14447             m_parent->openChild();
14448     }
14449
14450     void TrackerBase::close() {
14451
14452         // Close any still open children (e.g. generators)
14453         while( &m_ctx.currentTracker() != this )
14454             m_ctx.currentTracker().close();
14455
14456         switch( m_runState ) {
14457             case NeedsAnotherRun:
14458                 break;
14459
14460             case Executing:
14461                 m_runState = CompletedSuccessfully;
14462                 break;
14463             case ExecutingChildren:
14464                 if( std::all_of(m_children.begin(), m_children.end(), [](ITrackerPtr const& t){ return
t->isComplete(); }) )
14465                     m_runState = CompletedSuccessfully;
14466                 break;
14467
14468             case NotStarted:
14469             case CompletedSuccessfully:
14470             case Failed:
14471                 CATCH_INTERNAL_ERROR( "Illogical state: " << m_runState );
14472
14473             default:
14474                 CATCH_INTERNAL_ERROR( "Unknown state: " << m_runState );
14475         }

```

```

14476         moveToParent();
14477         m_ctx.completeCycle();
14478     }
14479     void TrackerBase::fail() {
14480         m_runState = Failed;
14481         if( m_parent )
14482             m_parent->markAsNeedingAnotherRun();
14483         moveToParent();
14484         m_ctx.completeCycle();
14485     }
14486     void TrackerBase::markAsNeedingAnotherRun() {
14487         m_runState = NeedsAnotherRun;
14488     }
14489
14490     void TrackerBase::moveToParent() {
14491         assert( m_parent );
14492         m_ctx.setCurrentTracker( m_parent );
14493     }
14494     void TrackerBase::moveToThis() {
14495         m_ctx.setCurrentTracker( this );
14496     }
14497
14498     SectionTracker::SectionTracker( NameAndLocation const& nameAndLocation, TrackerContext& ctx,
ITracker* parent )
14499     :   TrackerBase( nameAndLocation, ctx, parent ),
14500         m_trimmed_name(trim(nameAndLocation.name))
14501     {
14502         if( parent ) {
14503             while( !parent->isSectionTracker() )
14504                 parent = &parent->parent();
14505
14506             SectionTracker& parentSection = static_cast<SectionTracker&>( *parent );
14507             addNextFilters( parentSection.m_filters );
14508         }
14509     }
14510
14511     bool SectionTracker::isComplete() const {
14512         bool complete = true;
14513
14514         if (m_filters.empty())
14515             || m_filters[0] == ""
14516             || std::find(m_filters.begin(), m_filters.end(), m_trimmed_name) != m_filters.end()) {
14517             complete = TrackerBase::isComplete();
14518         }
14519         return complete;
14520     }
14521
14522     bool SectionTracker::isSectionTracker() const { return true; }
14523
14524     SectionTracker& SectionTracker::acquire( TrackerContext& ctx, NameAndLocation const&
nameAndLocation ) {
14525         std::shared_ptr<SectionTracker> section;
14526
14527         ITracker& currentTracker = ctx.currentTracker();
14528         if( ITrackerPtr childTracker = currentTracker.findChild( nameAndLocation ) ) {
14529             assert( childTracker );
14530             assert( childTracker->isSectionTracker() );
14531             section = std::static_pointer_cast<SectionTracker>( childTracker );
14532         }
14533         else {
14534             section = std::make_shared<SectionTracker>( nameAndLocation, ctx, &currentTracker );
14535             currentTracker.addChild( section );
14536         }
14537         if( !ctx.completedCycle() )
14538             section->tryOpen();
14539         return *section;
14540     }
14541
14542     void SectionTracker::tryOpen() {
14543         if( !isComplete() )
14544             open();
14545     }
14546
14547     void SectionTracker::addInitialFilters( std::vector<std::string> const& filters ) {
14548         if( !filters.empty() ) {
14549             m_filters.reserve( m_filters.size() + filters.size() + 2 );
14550             m_filters.emplace_back(""); // Root - should never be consulted
14551             m_filters.emplace_back(""); // Test Case - not a section filter
14552             m_filters.insert( m_filters.end(), filters.begin(), filters.end() );
14553         }
14554     }
14555     void SectionTracker::addNextFilters( std::vector<std::string> const& filters ) {
14556         if( filters.size() > 1 )
14557             m_filters.insert( m_filters.end(), filters.begin()+1, filters.end() );
14558     }
14559
14560     std::vector<std::string> const& SectionTracker::getFilters() const {

```

```

14561         return m_filters;
14562     }
14563
14564     std::string const& SectionTracker::trimmedName() const {
14565         return m_trimmed_name;
14566     }
14567
14568 } // namespace TestCaseTracking
14569
14570 using TestCaseTracking::ITracker;
14571 using TestCaseTracking::TrackerContext;
14572 using TestCaseTracking::SectionTracker;
14573
14574 } // namespace Catch
14575
14576 #if defined(__clang__)
14577 #    pragma clang diagnostic pop
14578 #endif
14579 // end catch_test_case_tracker.cpp
14580 // start catch_test_registry.cpp
14581
14582 namespace Catch {
14583
14584     auto makeTestInvoker( void(*testAsFunction)() ) noexcept -> ITestInvoker* {
14585         return new(std::nothrow) TestInvokerAsFunction( testAsFunction );
14586     }
14587
14588     NameAndTags::NameAndTags( StringRef const& name_ , StringRef const& tags_ ) noexcept : name( name_
), tags( tags_ ) {}
14589
14590     AutoReg::AutoReg( ITestInvoker* invoker, SourceLineInfo const& lineInfo, StringRef const&
classOrMethod, NameAndTags const& nameAndTags ) noexcept {
14591         CATCH_TRY {
14592             getMutableRegistryHub()
14593                 .registerTest(
14594                     makeTestCase(
14595                         invoker,
14596                         extractClassName( classOrMethod ),
14597                         nameAndTags,
14598                         lineInfo));
14599         } CATCH_CATCH_ALL {
14600             // Do not throw when constructing global objects, instead register the exception to be
            processed later
14601             getMutableRegistryHub().registerStartupException();
14602         }
14603     }
14604
14605     AutoReg::~AutoReg() = default;
14606 }
14607 // end catch_test_registry.cpp
14608 // start catch_test_spec.cpp
14609
14610 #include <algorithm>
14611 #include <string>
14612 #include <vector>
14613 #include <memory>
14614
14615 namespace Catch {
14616
14617     TestSpec::Pattern::Pattern( std::string const& name )
14618     : m_name( name )
14619     {}
14620
14621     TestSpec::Pattern::~Pattern() = default;
14622
14623     std::string const& TestSpec::Pattern::name() const {
14624         return m_name;
14625     }
14626
14627     TestSpec::NamePattern::NamePattern( std::string const& name, std::string const& filterString )
14628     : Pattern( filterString )
14629     , m_wildcardPattern( toLower( name ), CaseSensitive::No )
14630     {}
14631
14632     bool TestSpec::NamePattern::matches( TestCaseInfo const& testCase ) const {
14633         return m_wildcardPattern.matches( testCase.name );
14634     }
14635
14636     TestSpec::TagPattern::TagPattern( std::string const& tag, std::string const& filterString )
14637     : Pattern( filterString )
14638     , m_tag( toLower( tag ) )
14639     {}
14640
14641     bool TestSpec::TagPattern::matches( TestCaseInfo const& testCase ) const {
14642         return std::find( begin( testCase.caseTags ),
14643             end( testCase.caseTags ),
14644             m_tag ) != end( testCase.caseTags );

```

```

14645     }
14646
14647     TestSpec::ExcludedPattern::ExcludedPattern( PatternPtr const& underlyingPattern )
14648     : Pattern( underlyingPattern->name() )
14649     , m_underlyingPattern( underlyingPattern )
14650     {}
14651
14652     bool TestSpec::ExcludedPattern::matches( TestCaseInfo const& testCase ) const {
14653         return !m_underlyingPattern->matches( testCase );
14654     }
14655
14656     bool TestSpec::Filter::matches( TestCaseInfo const& testCase ) const {
14657         return std::all_of( m_patterns.begin(), m_patterns.end(), [&]( PatternPtr const& p ){ return
14658 p->matches( testCase ); } );
14659     }
14660
14661     std::string TestSpec::Filter::name() const {
14662         std::string name;
14663         for( auto const& p : m_patterns )
14664             name += p->name();
14665         return name;
14666     }
14667
14668     bool TestSpec::hasFilters() const {
14669         return !m_filters.empty();
14670     }
14671
14672     bool TestSpec::matches( TestCaseInfo const& testCase ) const {
14673         return std::any_of( m_filters.begin(), m_filters.end(), [&]( Filter const& f ){ return
14674 f.matches( testCase ); } );
14675     }
14676
14677     TestSpec::Matches TestSpec::matchesByFilter( std::vector<TestCase> const& testCases, IConfig
14678 const& config ) const
14679     {
14680         Matches matches( m_filters.size() );
14681         std::transform( m_filters.begin(), m_filters.end(), matches.begin(), [&]( Filter const& filter
14682 ) {
14683             std::vector<TestCase const*> currentMatches;
14684             for( auto const& test : testCases )
14685                 if( isThrowSafe( test, config ) && filter.matches( test ) )
14686                     currentMatches.emplace_back( &test );
14687             return FilterMatch{ filter.name(), currentMatches };
14688         } );
14689         return matches;
14690     }
14691
14692     const TestSpec::vectorStrings& TestSpec::getInvalidArgs() const {
14693         return (m_invalidArgs);
14694     }
14695
14696 // end catch_test_spec.cpp
14697 // start catch_test_spec_parser.cpp
14698 namespace Catch {
14699
14700     TestSpecParser::TestSpecParser( ITagAliasRegistry const& tagAliases ) : m_tagAliases( &tagAliases
14701 ) {}
14702
14703     TestSpecParser& TestSpecParser::parse( std::string const& arg ) {
14704         m_mode = None;
14705         m_exclusion = false;
14706         m_arg = m_tagAliases->expandAliases( arg );
14707         m_escapeChars.clear();
14708         m_substring.reserve(m_arg.size());
14709         m_patternName.reserve(m_arg.size());
14710         m_realPatternPos = 0;
14711
14712         for( m_pos = 0; m_pos < m_arg.size(); ++m_pos )
14713             //if visitChar fails
14714             if( !visitChar( m_arg[m_pos] ) ) {
14715                 m_testSpec.m_invalidArgs.push_back(arg);
14716                 break;
14717             }
14718         endMode();
14719         return *this;
14720     }
14721
14722     TestSpec TestSpecParser::testSpec() {
14723         addFilter();
14724         return m_testSpec;
14725     }
14726
14727     bool TestSpecParser::visitChar( char c ) {
14728         if( (m_mode != EscapedName) && (c == '\\') ) {
14729             escape();
14730             addCharToPattern(c);
14731             return true;
14732         }

```

```

14727         }else if( (m_mode != EscapedName) && (c == ',' ) ) {
14728             return separate();
14729         }
14730
14731         switch( m_mode ) {
14732             case None:
14733                 if( processNoneChar( c ) )
14734                     return true;
14735                 break;
14736             case Name:
14737                 processNameChar( c );
14738                 break;
14739             case EscapedName:
14740                 endMode();
14741                 addCharToPattern(c);
14742                 return true;
14743             default:
14744             case Tag:
14745             case QuotedName:
14746                 if( processOtherChar( c ) )
14747                     return true;
14748                 break;
14749         }
14750
14751         m_substring += c;
14752         if( !isControlChar( c ) ) {
14753             m_patternName += c;
14754             m_realPatternPos++;
14755         }
14756         return true;
14757     }
14758     // Two of the processing methods return true to signal the caller to return
14759     // without adding the given character to the current pattern strings
14760     bool TestSpecParser::processNoneChar( char c ) {
14761         switch( c ) {
14762             case ' ':
14763                 return true;
14764             case '~':
14765                 m_exclusion = true;
14766                 return false;
14767             case '[':
14768                 startNewMode( Tag );
14769                 return false;
14770             case '"':
14771                 startNewMode( QuotedName );
14772                 return false;
14773             default:
14774                 startNewMode( Name );
14775                 return false;
14776         }
14777     }
14778     void TestSpecParser::processNameChar( char c ) {
14779         if( c == '[' ) {
14780             if( m_substring == "exclude:" )
14781                 m_exclusion = true;
14782             else
14783                 endMode();
14784             startNewMode( Tag );
14785         }
14786     }
14787     bool TestSpecParser::processOtherChar( char c ) {
14788         if( !isControlChar( c ) )
14789             return false;
14790         m_substring += c;
14791         endMode();
14792         return true;
14793     }
14794     void TestSpecParser::startNewMode( Mode mode ) {
14795         m_mode = mode;
14796     }
14797     void TestSpecParser::endMode() {
14798         switch( m_mode ) {
14799             case Name:
14800             case QuotedName:
14801                 return addNamePattern();
14802             case Tag:
14803                 return addTagPattern();
14804             case EscapedName:
14805                 revertBackToLastMode();
14806                 return;
14807             case None:
14808             default:
14809                 return startNewMode( None );
14810         }
14811     }
14812     void TestSpecParser::escape() {
14813         saveLastMode();

```

```

14814         m_mode = EscapedName;
14815         m_escapeChars.push_back(m_realPatternPos);
14816     }
14817     bool TestSpecParser::isControlChar( char c ) const {
14818         switch( m_mode ) {
14819             default:
14820                 return false;
14821             case None:
14822                 return c == '~';
14823             case Name:
14824                 return c == '[';
14825             case EscapedName:
14826                 return true;
14827             case QuotedName:
14828                 return c == '"';
14829             case Tag:
14830                 return c == '[' || c == ']' ;
14831         }
14832     }
14833
14834     void TestSpecParser::addFilter() {
14835         if( !m_currentFilter.m_patterns.empty() ) {
14836             m_testSpec.m_filters.push_back( m_currentFilter );
14837             m_currentFilter = TestSpec::Filter();
14838         }
14839     }
14840
14841     void TestSpecParser::saveLastMode() {
14842         lastMode = m_mode;
14843     }
14844
14845     void TestSpecParser::revertBackToLastMode() {
14846         m_mode = lastMode;
14847     }
14848
14849     bool TestSpecParser::separate() {
14850         if( (m_mode==QuotedName) || (m_mode==Tag) ){
14851             //invalid argument, signal failure to previous scope.
14852             m_mode = None;
14853             m_pos = m_arg.size();
14854             m_substring.clear();
14855             m_patternName.clear();
14856             m_realPatternPos = 0;
14857             return false;
14858         }
14859         endMode();
14860         addFilter();
14861         return true; //success
14862     }
14863
14864     std::string TestSpecParser::preprocessPattern() {
14865         std::string token = m_patternName;
14866         for (std::size_t i = 0; i < m_escapeChars.size(); ++i)
14867             token = token.substr(0, m_escapeChars[i] - i) + token.substr(m_escapeChars[i] - i + 1);
14868         m_escapeChars.clear();
14869         if (startsWith(token, "exclude:")) {
14870             m_exclusion = true;
14871             token = token.substr(8);
14872         }
14873
14874         m_patternName.clear();
14875         m_realPatternPos = 0;
14876
14877         return token;
14878     }
14879
14880     void TestSpecParser::addNamePattern() {
14881         auto token = preprocessPattern();
14882
14883         if (!token.empty()) {
14884             TestSpec::PatternPtr pattern = std::make_shared<TestSpec::NamePattern>(token,
14885 m_substring);
14886             if (m_exclusion)
14887                 pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
14888             m_currentFilter.m_patterns.push_back(pattern);
14889         }
14890         m_substring.clear();
14891         m_exclusion = false;
14892         m_mode = None;
14893     }
14894
14895     void TestSpecParser::addTagPattern() {
14896         auto token = preprocessPattern();
14897
14898         if (!token.empty()) {
14899             // If the tag pattern is the "hide and tag" shorthand (e.g. [.foo])
14900             // we have to create a separate hide tag and shorten the real one

```

```

14900         if (token.size() > 1 && token[0] == '.') {
14901             token.erase(token.begin());
14902             TestSpec::PatternPtr pattern = std::make_shared<TestSpec::TagPattern>(".",
m_substring);
14903             if (m_exclusion) {
14904                 pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
14905             }
14906             m_currentFilter.m_patterns.push_back(pattern);
14907         }
14908         TestSpec::PatternPtr pattern = std::make_shared<TestSpec::TagPattern>(token, m_substring);
14909
14910         if (m_exclusion) {
14911             pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
14912         }
14913         m_currentFilter.m_patterns.push_back(pattern);
14914     }
14915     m_substring.clear();
14916     m_exclusion = false;
14917     m_mode = None;
14918 }
14919
14920 TestSpec parseTestSpec( std::string const& arg ) {
14921     return TestSpecParser( ITagAliasRegistry::get() ).parse( arg ).testSpec();
14922 }
14923
14924 } // namespace Catch
14925 // end catch_test_spec_parser.cpp
14926 // start catch_timer.cpp
14927
14928 #include <chrono>
14929
14930 static const uint64_t nanosecondsInSecond = 1000000000;
14931
14932 namespace Catch {
14933     auto getCurrentNanosecondsSinceEpoch() -> uint64_t {
14934         return std::chrono::duration_cast<std::chrono::nanoseconds>(
std::chrono::high_resolution_clock::now().time_since_epoch()).count();
14935     }
14936
14937     namespace {
14938         auto estimateClockResolution() -> uint64_t {
14939             uint64_t sum = 0;
14940             static const uint64_t iterations = 1000000;
14941
14942             auto startTime = getCurrentNanosecondsSinceEpoch();
14943
14944             for( std::size_t i = 0; i < iterations; ++i ) {
14945
14946                 uint64_t ticks;
14947                 uint64_t baseTicks = getCurrentNanosecondsSinceEpoch();
14948                 do {
14949                     ticks = getCurrentNanosecondsSinceEpoch();
14950                 } while( ticks == baseTicks );
14951
14952                 auto delta = ticks - baseTicks;
14953                 sum += delta;
14954
14955                 // If we have been calibrating for over 3 seconds -- the clock
14956                 // is terrible and we should move on.
14957                 // TBD: How to signal that the measured resolution is probably wrong?
14958                 if (ticks > startTime + 3 * nanosecondsInSecond) {
14959                     return sum / ( i + 1u );
14960                 }
14961             }
14962         }
14963     }
14964
14965     // We're just taking the mean, here. To do better we could take the std. dev and exclude
outliers
14966     // - and potentially do more iterations if there's a high variance.
14967     return sum/iterations;
14968 }
14969
14970 auto getEstimatedClockResolution() -> uint64_t {
14971     static auto s_resolution = estimateClockResolution();
14972     return s_resolution;
14973 }
14974
14975 void Timer::start() {
14976     m_nanoseconds = getCurrentNanosecondsSinceEpoch();
14977 }
14978
14979 auto Timer::getElapsedNanoseconds() const -> uint64_t {
14980     return getCurrentNanosecondsSinceEpoch() - m_nanoseconds;
14981 }
14982
14983 auto Timer::getElapsedMicroseconds() const -> uint64_t {
14984     return getElapsedNanoseconds()/1000;
14985 }

```

```

14984     auto Timer::getElapsedMilliseconds() const -> unsigned int {
14985         return static_cast<unsigned int>(getElapsedMicroseconds()/1000);
14986     }
14987     auto Timer::getElapsedSeconds() const -> double {
14988         return getElapsedMicroseconds()/1000000.0;
14989     }
14990
14991 } // namespace Catch
14992 // end catch_timer.cpp
14993 // start catch_tostring.cpp
14994
14995 #if defined(__clang__)
14996 #   pragma clang diagnostic push
14997 #   pragma clang diagnostic ignored "-Wexit-time-destructors"
14998 #   pragma clang diagnostic ignored "-Wglobal-constructors"
14999 #endif
15000
15001 // Enable specific decls locally
15002 #if !defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
15003 #define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
15004 #endif
15005
15006 #include <cmath>
15007 #include <iomanip>
15008
15009 namespace Catch {
15010
15011     namespace Detail {
15012
15013         const std::string unprintableString = "{?}";
15014
15015         namespace {
15016             const int hexThreshold = 255;
15017
15018             struct Endianness {
15019                 enum Arch { Big, Little };
15020
15021                 static Arch which() {
15022                     int one = 1;
15023                     // If the lowest byte we read is non-zero, we can assume
15024                     // that little endian format is used.
15025                     auto value = *reinterpret_cast<char*>(&one);
15026                     return value ? Little : Big;
15027                 }
15028             };
15029         }
15030
15031         std::string rawMemoryToString( const void *object, std::size_t size ) {
15032             // Reverse order for little endian architectures
15033             int i = 0, end = static_cast<int>( size ), inc = 1;
15034             if( Endianness::which() == Endianness::Little ) {
15035                 i = end-1;
15036                 end = inc = -1;
15037             }
15038
15039             unsigned char const *bytes = static_cast<unsigned char const *>(object);
15040             ReusableStringStream rss;
15041             rss << "0x" << std::setfill('0') << std::hex;
15042             for( ; i != end; i += inc )
15043                 rss << std::setw(2) << static_cast<unsigned>(bytes[i]);
15044             return rss.str();
15045         }
15046     }
15047
15048     template<typename T>
15049     std::string fpToString( T value, int precision ) {
15050         if (Catch::isnan(value)) {
15051             return "nan";
15052         }
15053
15054         ReusableStringStream rss;
15055         rss << std::setprecision( precision )
15056             << std::fixed
15057             << value;
15058         std::string d = rss.str();
15059         std::size_t i = d.find_last_not_of( '0' );
15060         if( i != std::string::npos && i != d.size()-1 ) {
15061             if( d[i] == '.' )
15062                 i++;
15063             d = d.substr( 0, i+1 );
15064         }
15065         return d;
15066     }
15067
15068     //
15069     // Out-of-line defs for full specialization of StringMaker
15070     //

```



```

15073
15074 std::string StringMaker<std::string>::convert(const std::string& str) {
15075     if (!getCurrentContext().getConfig()->showInvisibles()) {
15076         return '"' + str + '"';
15077     }
15078
15079     std::string s("\\");
15080     for (char c : str) {
15081         switch (c) {
15082             case '\\n':
15083                 s.append("\\\\n");
15084                 break;
15085             case '\\t':
15086                 s.append("\\\\t");
15087                 break;
15088             default:
15089                 s.push_back(c);
15090                 break;
15091         }
15092     }
15093     s.append("\\");
15094     return s;
15095 }
15096
15097 #ifdef CATCH_CONFIG_CPP17_STRING_VIEW
15098 std::string StringMaker<std::string_view>::convert(std::string_view str) {
15099     return ::Catch::Detail::stringify(std::string{ str });
15100 }
15101 #endif
15102
15103 std::string StringMaker<char const*>::convert(char const* str) {
15104     if (str) {
15105         return ::Catch::Detail::stringify(std::string{ str });
15106     } else {
15107         return "{null string}";
15108     }
15109 }
15110
15111 std::string StringMaker<char*>::convert(char* str) {
15112     if (str) {
15113         return ::Catch::Detail::stringify(std::string{ str });
15114     } else {
15115         return "{null string}";
15116     }
15117 }
15118 #ifdef CATCH_CONFIG_WCHAR
15119 std::string StringMaker<std::wstring>::convert(const std::wstring& wstr) {
15120     std::string s;
15121     s.reserve(wstr.size());
15122     for (auto c : wstr) {
15123         s += (c <= 0xff) ? static_cast<char>(c) : '?';
15124     }
15125     return ::Catch::Detail::stringify(s);
15126 }
15127
15128 #ifdef CATCH_CONFIG_CPP17_STRING_VIEW
15129 std::string StringMaker<std::wstring_view>::convert(std::wstring_view str) {
15130     return StringMaker<std::wstring>::convert(std::wstring(str));
15131 }
15132 #endif
15133
15134 std::string StringMaker<wchar_t const*>::convert(wchar_t const* str) {
15135     if (str) {
15136         return ::Catch::Detail::stringify(std::wstring{ str });
15137     } else {
15138         return "{null string}";
15139     }
15140 }
15141
15142 std::string StringMaker<wchar_t*>::convert(wchar_t* str) {
15143     if (str) {
15144         return ::Catch::Detail::stringify(std::wstring{ str });
15145     } else {
15146         return "{null string}";
15147     }
15148 #endif
15149
15150 #if defined(CATCH_CONFIG_CPP17_BYTE)
15151 #include <cstdint>
15152 std::string StringMaker<std::byte>::convert(std::byte value) {
15153     return ::Catch::Detail::stringify(std::to_integer<unsigned long long>(value));
15154 }
15155 #endif // defined(CATCH_CONFIG_CPP17_BYTE)
15156
15157 std::string StringMaker<int>::convert(int value) {
15158     return ::Catch::Detail::stringify(static_cast<long long>(value));
15159 }

```

```

15160 std::string StringMaker<long>::convert(long value) {
15161     return ::Catch::Detail::stringify(static_cast<long long>(value));
15162 }
15163 std::string StringMaker<long long>::convert(long long value) {
15164     ReusableStringStream rss;
15165     rss << value;
15166     if (value > Detail::hexThreshold) {
15167         rss << " (0x" << std::hex << value << ')';
15168     }
15169     return rss.str();
15170 }
15171
15172 std::string StringMaker<unsigned int>::convert(unsigned int value) {
15173     return ::Catch::Detail::stringify(static_cast<unsigned long long>(value));
15174 }
15175 std::string StringMaker<unsigned long>::convert(unsigned long value) {
15176     return ::Catch::Detail::stringify(static_cast<unsigned long long>(value));
15177 }
15178 std::string StringMaker<unsigned long long>::convert(unsigned long long value) {
15179     ReusableStringStream rss;
15180     rss << value;
15181     if (value > Detail::hexThreshold) {
15182         rss << " (0x" << std::hex << value << ')';
15183     }
15184     return rss.str();
15185 }
15186
15187 std::string StringMaker<bool>::convert(bool b) {
15188     return b ? "true" : "false";
15189 }
15190
15191 std::string StringMaker<signed char>::convert(signed char value) {
15192     if (value == '\r') {
15193         return "\\r";
15194     } else if (value == '\f') {
15195         return "\\f";
15196     } else if (value == '\n') {
15197         return "\\n";
15198     } else if (value == '\t') {
15199         return "\\t";
15200     } else if ('\0' <= value && value < ' ') {
15201         return ::Catch::Detail::stringify(static_cast<unsigned int>(value));
15202     } else {
15203         char chstr[] = " ' ";
15204         chstr[1] = value;
15205         return chstr;
15206     }
15207 }
15208 std::string StringMaker<char>::convert(char c) {
15209     return ::Catch::Detail::stringify(static_cast<signed char>(c));
15210 }
15211 std::string StringMaker<unsigned char>::convert(unsigned char c) {
15212     return ::Catch::Detail::stringify(static_cast<char>(c));
15213 }
15214
15215 std::string StringMaker<std::nullptr_t>::convert(std::nullptr_t) {
15216     return "nullptr";
15217 }
15218
15219 int StringMaker<float>::precision = 5;
15220
15221 std::string StringMaker<float>::convert(float value) {
15222     return fpToString(value, precision) + 'f';
15223 }
15224
15225 int StringMaker<double>::precision = 10;
15226
15227 std::string StringMaker<double>::convert(double value) {
15228     return fpToString(value, precision);
15229 }
15230
15231 std::string ratio_string<std::atto>::symbol() { return "a"; }
15232 std::string ratio_string<std::femto>::symbol() { return "f"; }
15233 std::string ratio_string<std::pico>::symbol() { return "p"; }
15234 std::string ratio_string<std::nano>::symbol() { return "n"; }
15235 std::string ratio_string<std::micro>::symbol() { return "u"; }
15236 std::string ratio_string<std::milli>::symbol() { return "m"; }
15237
15238 } // end namespace Catch
15239
15240 #if defined(__clang__)
15241 #    pragma clang diagnostic pop
15242 #endif
15243
15244 // end catch_tostring.cpp
15245 // start catch_totals.cpp
15246

```

```

15247 namespace Catch {
15248
15249     Counts Counts::operator - ( Counts const& other ) const {
15250         Counts diff;
15251         diff.passed = passed - other.passed;
15252         diff.failed = failed - other.failed;
15253         diff.failedButOk = failedButOk - other.failedButOk;
15254         return diff;
15255     }
15256
15257     Counts& Counts::operator += ( Counts const& other ) {
15258         passed += other.passed;
15259         failed += other.failed;
15260         failedButOk += other.failedButOk;
15261         return *this;
15262     }
15263
15264     std::size_t Counts::total() const {
15265         return passed + failed + failedButOk;
15266     }
15267     bool Counts::allPassed() const {
15268         return failed == 0 && failedButOk == 0;
15269     }
15270     bool Counts::allOk() const {
15271         return failed == 0;
15272     }
15273
15274     Totals Totals::operator - ( Totals const& other ) const {
15275         Totals diff;
15276         diff.assertions = assertions - other.assertions;
15277         diff.testCases = testCases - other.testCases;
15278         return diff;
15279     }
15280
15281     Totals& Totals::operator += ( Totals const& other ) {
15282         assertions += other.assertions;
15283         testCases += other.testCases;
15284         return *this;
15285     }
15286
15287     Totals Totals::delta( Totals const& prevTotals ) const {
15288         Totals diff = *this - prevTotals;
15289         if( diff.assertions.failed > 0 )
15290             ++diff.testCases.failed;
15291         else if( diff.assertions.failedButOk > 0 )
15292             ++diff.testCases.failedButOk;
15293         else
15294             ++diff.testCases.passed;
15295         return diff;
15296     }
15297 }
15298
15299 // end catch_totals.cpp
15300 // start catch_uncaught_exceptions.cpp
15301
15302 // start catch_config_uncaught_exceptions.hpp
15303
15304 // Copyright Catch2 Authors
15305 // Distributed under the Boost Software License, Version 1.0.
15306 // (See accompanying file LICENSE_1_0.txt or copy at
15307 // https://www.boost.org/LICENSE\_1\_0.txt)
15308
15309 // SPDX-License-Identifier: BSL-1.0
15310
15311 #ifndef CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
15312 #define CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
15313
15314 #if defined(_MSC_VER)
15315 # if _MSC_VER >= 1900 // Visual Studio 2015 or newer
15316 #   define CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
15317 # endif
15318 #endif
15319
15320 #include <exception>
15321
15322 #if defined(__cpp_lib_uncaught_exceptions) \
15323     && !defined(CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)
15324
15325 #   define CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
15326 #endif // __cpp_lib_uncaught_exceptions
15327
15328 #if defined(CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS) \
15329     && !defined(CATCH_CONFIG_NO_CPP17_UNCAUGHT_EXCEPTIONS) \
15330     && !defined(CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)
15331
15332 #   define CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
15333 #endif

```

```

15334
15335 #endif // CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
15336 // end catch_config_uncaught_exceptions.hpp
15337 #include <exception>
15338
15339 namespace Catch {
15340     bool uncaught_exceptions() {
15341         #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
15342             return false;
15343         #elif defined(CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)
15344             return std::uncaught_exceptions() > 0;
15345         #else
15346             return std::uncaught_exception();
15347         #endif
15348     }
15349 } // end namespace Catch
15350 // end catch_uncaught_exceptions.cpp
15351 // start catch_version.cpp
15352
15353 #include <ostream>
15354
15355 namespace Catch {
15356     Version::Version
15357     (   unsigned int _majorVersion,
15358         unsigned int _minorVersion,
15359         unsigned int _patchNumber,
15360         char const * _branchName,
15361         unsigned int _buildNumber )
15362     :   majorVersion( _majorVersion ),
15363         minorVersion( _minorVersion ),
15364         patchNumber( _patchNumber ),
15365         branchName( _branchName ),
15366         buildNumber( _buildNumber )
15367     {}
15368
15369     std::ostream& operator << ( std::ostream& os, Version const& version ) {
15370         os << version.majorVersion << "."
15371            << version.minorVersion << "."
15372            << version.patchNumber;
15373         // branchName is never null -> 0th char is \0 if it is empty
15374         if (version.branchName[0]) {
15375             os << "-" << version.branchName
15376                << "." << version.buildNumber;
15377         }
15378         return os;
15379     }
15380
15381     Version const& libraryVersion() {
15382         static Version version( 2, 13, 7, "", 0 );
15383         return version;
15384     }
15385 }
15386
15387 // end catch_version.cpp
15388 // start catch_wildcard_pattern.cpp
15389
15390 namespace Catch {
15391     WildcardPattern::WildcardPattern( std::string const& pattern,
15392                                     CaseSensitive::Choice caseSensitivity )
15393     :   m_caseSensitivity( caseSensitivity ),
15394         m_pattern( normaliseString( pattern ) )
15395     {
15396         if ( startsWith( m_pattern, '*' ) ) {
15397             m_pattern = m_pattern.substr( 1 );
15398             m_wildcard = WildcardAtStart;
15399         }
15400         if ( endsWith( m_pattern, '*' ) ) {
15401             m_pattern = m_pattern.substr( 0, m_pattern.size()-1 );
15402             m_wildcard = static_cast<WildcardPosition>( m_wildcard | WildcardAtEnd );
15403         }
15404     }
15405
15406     bool WildcardPattern::matches( std::string const& str ) const {
15407         switch( m_wildcard ) {
15408             case NoWildcard:
15409                 return m_pattern == normaliseString( str );
15410             case WildcardAtStart:
15411                 return endsWith( normaliseString( str ), m_pattern );
15412             case WildcardAtEnd:
15413                 return startsWith( normaliseString( str ), m_pattern );
15414             case WildcardAtBothEnds:
15415                 return contains( normaliseString( str ), m_pattern );
15416             default:
15417                 CATCH_INTERNAL_ERROR( "Unknown enum" );
15418         }
15419     }
15420 }

```

```

15421     }
15422
15423     std::string WildcardPattern::normaliseString( std::string const& str ) const {
15424         return trim( m_caseSensitivity == CaseSensitive::No ? toLower( str ) : str );
15425     }
15426 }
15427 // end catch_wildcard_pattern.cpp
15428 // start catch_xmlwriter.cpp
15429
15430 #include <iomanip>
15431 #include <type_traits>
15432
15433 namespace Catch {
15434
15435     namespace {
15436
15437         size_t trailingBytes(unsigned char c) {
15438             if ((c & 0xE0) == 0xC0) {
15439                 return 2;
15440             }
15441             if ((c & 0xF0) == 0xE0) {
15442                 return 3;
15443             }
15444             if ((c & 0xF8) == 0xF0) {
15445                 return 4;
15446             }
15447             CATCH_INTERNAL_ERROR("Invalid multibyte utf-8 start byte encountered");
15448         }
15449
15450         uint32_t headerValue(unsigned char c) {
15451             if ((c & 0xE0) == 0xC0) {
15452                 return c & 0x1F;
15453             }
15454             if ((c & 0xF0) == 0xE0) {
15455                 return c & 0x0F;
15456             }
15457             if ((c & 0xF8) == 0xF0) {
15458                 return c & 0x07;
15459             }
15460             CATCH_INTERNAL_ERROR("Invalid multibyte utf-8 start byte encountered");
15461         }
15462
15463         void hexEscapeChar(std::ostream& os, unsigned char c) {
15464             std::ios_base::fmtflags f(os.flags());
15465             os << "\\x"
15466                 << std::uppercase << std::hex << std::setfill('0') << std::setw(2)
15467                 << static_cast<int>(c);
15468             os.flags(f);
15469         }
15470
15471         bool shouldNewline(XmlFormatting fmt) {
15472             return !(static_cast<std::underlying_type<XmlFormatting>::type>(fmt &
15473                 XmlFormatting::Newline));
15474         }
15475
15476         bool shouldIndent(XmlFormatting fmt) {
15477             return !(static_cast<std::underlying_type<XmlFormatting>::type>(fmt &
15478                 XmlFormatting::Indent));
15479         }
15480     } // anonymous namespace
15481
15482     XmlFormatting operator | (XmlFormatting lhs, XmlFormatting rhs) {
15483         return static_cast<XmlFormatting>(
15484             static_cast<std::underlying_type<XmlFormatting>::type>(lhs) |
15485             static_cast<std::underlying_type<XmlFormatting>::type>(rhs)
15486         );
15487     }
15488
15489     XmlFormatting operator & (XmlFormatting lhs, XmlFormatting rhs) {
15490         return static_cast<XmlFormatting>(
15491             static_cast<std::underlying_type<XmlFormatting>::type>(lhs) &
15492             static_cast<std::underlying_type<XmlFormatting>::type>(rhs)
15493         );
15494     }
15495
15496     XmlEncode::XmlEncode( std::string const& str, ForWhat forWhat )
15497     :   m_str( str ),
15498         m_forWhat( forWhat )
15499     {}
15500
15501     void XmlEncode::encodeTo( std::ostream& os ) const {
15502         // Apostrophe escaping not necessary if we always use " to write attributes
15503         // (see: http://www.w3.org/TR/xml/#syntax)
15504
15505         for( std::size_t idx = 0; idx < m_str.size(); ++ idx ) {
15506             unsigned char c = m_str[idx];

```

```

15506         switch (c) {
15507         case '<': os « "<"; break;
15508         case '&': os « "&"; break;
15509
15510         case '>':
15511             // See: http://www.w3.org/TR/xml/#syntax
15512             if (idx > 2 && m_str[idx - 1] == ']' && m_str[idx - 2] == ']')
15513                 os « ">";
15514             else
15515                 os « c;
15516             break;
15517
15518         case '\"':
15519             if (m_forWhat == ForAttributes)
15520                 os « """;
15521             else
15522                 os « c;
15523             break;
15524
15525         default:
15526             // Check for control characters and invalid utf-8
15527
15528             // Escape control characters in standard ascii
15529             // see
15530             http://stackoverflow.com/questions/404107/why-are-control-characters-illegal-in-xml-1-0
15531             if (c < 0x09 || (c > 0x0D && c < 0x20) || c == 0x7F) {
15532                 hexEscapeChar(os, c);
15533                 break;
15534             }
15535
15536             // Plain ASCII: Write it to stream
15537             if (c < 0x7F) {
15538                 os « c;
15539                 break;
15540             }
15541
15542             // UTF-8 territory
15543             // Check if the encoding is valid and if it is not, hex escape bytes.
15544             // Important: We do not check the exact decoded values for validity, only the encoding
15545             format
15546             // First check that this bytes is a valid lead byte:
15547             // This means that it is not encoded as 1111 1XXX
15548             // Or as 10XX XXXX
15549             if (c < 0xC0 ||
15550                 c >= 0xF8) {
15551                 hexEscapeChar(os, c);
15552                 break;
15553             }
15554
15555             auto encBytes = trailingBytes(c);
15556             // Are there enough bytes left to avoid accessing out-of-bounds memory?
15557             if (idx + encBytes - 1 >= m_str.size()) {
15558                 hexEscapeChar(os, c);
15559                 break;
15560             }
15561
15562             // The header is valid, check data
15563             // The next encBytes bytes must together be a valid utf-8
15564             // This means: bitpattern 10XX XXXX and the extracted value is sane (ish)
15565             bool valid = true;
15566             uint32_t value = headerValue(c);
15567             for (std::size_t n = 1; n < encBytes; ++n) {
15568                 unsigned char nc = m_str[idx + n];
15569                 valid &= ((nc & 0xC0) == 0x80);
15570                 value = (value << 6) | (nc & 0x3F);
15571             }
15572
15573             if (
15574                 // Wrong bit pattern of following bytes
15575                 (!valid) ||
15576                 // Overlong encodings
15577                 (value < 0x80) ||
15578                 (0x80 <= value && value < 0x800 && encBytes > 2) ||
15579                 (0x800 < value && value < 0x10000 && encBytes > 3) ||
15580                 // Encoded value out of range
15581                 (value >= 0x110000)
15582             ) {
15583                 hexEscapeChar(os, c);
15584                 break;
15585             }
15586
15587             // If we got here, this is in fact a valid(ish) utf-8 sequence
15588             for (std::size_t n = 0; n < encBytes; ++n) {
15589                 os « m_str[idx + n];
15590             }
15591             idx += encBytes - 1;
15592             break;
15593     }

```

```

15591     }
15592 }
15593
15594 std::ostream& operator << ( std::ostream& os, XmlEncode const& xmlEncode ) {
15595     xmlEncode.encodeTo( os );
15596     return os;
15597 }
15598
15599 XmlWriter::ScopedElement::ScopedElement( XmlWriter* writer, XmlFormatting fmt )
15600 :   m_writer( writer ),
15601     m_fmt( fmt )
15602 {}
15603
15604 XmlWriter::ScopedElement::ScopedElement( ScopedElement&& other ) noexcept
15605 :   m_writer( other.m_writer ),
15606     m_fmt( other.m_fmt )
15607 {
15608     other.m_writer = nullptr;
15609     other.m_fmt = XmlFormatting::None;
15610 }
15611 XmlWriter::ScopedElement& XmlWriter::ScopedElement::operator=( ScopedElement&& other ) noexcept {
15612     if ( m_writer ) {
15613         m_writer->endElement();
15614     }
15615     m_writer = other.m_writer;
15616     other.m_writer = nullptr;
15617     m_fmt = other.m_fmt;
15618     other.m_fmt = XmlFormatting::None;
15619     return *this;
15620 }
15621
15622 XmlWriter::ScopedElement::~ScopedElement() {
15623     if (m_writer) {
15624         m_writer->endElement(m_fmt);
15625     }
15626 }
15627
15628 XmlWriter::ScopedElement& XmlWriter::ScopedElement::writeText( std::string const& text,
15629 XmlFormatting fmt ) {
15630     m_writer->writeText( text, fmt );
15631     return *this;
15632 }
15633
15634 XmlWriter::XmlWriter( std::ostream& os ) : m_os( os )
15635 {
15636     writeDeclaration();
15637 }
15638
15639 XmlWriter::~XmlWriter() {
15640     while (!m_tags.empty()) {
15641         endElement();
15642     }
15643     newlineIfNecessary();
15644 }
15645
15646 XmlWriter& XmlWriter::startElement( std::string const& name, XmlFormatting fmt ) {
15647     ensureTagClosed();
15648     newlineIfNecessary();
15649     if (shouldIndent(fmt)) {
15650         m_os << m_indent;
15651         m_indent += " ";
15652     }
15653     m_os << "<" << name;
15654     m_tags.push_back( name );
15655     m_tagIsOpen = true;
15656     applyFormatting(fmt);
15657     return *this;
15658 }
15659
15660 XmlWriter::ScopedElement XmlWriter::scopedElement( std::string const& name, XmlFormatting fmt ) {
15661     ScopedElement scoped( this, fmt );
15662     startElement( name, fmt );
15663     return scoped;
15664 }
15665
15666 XmlWriter& XmlWriter::endElement(XmlFormatting fmt) {
15667     m_indent = m_indent.substr(0, m_indent.size() - 2);
15668
15669     if( m_tagIsOpen ) {
15670         m_os << ">";
15671         m_tagIsOpen = false;
15672     } else {
15673         newlineIfNecessary();
15674         if (shouldIndent(fmt)) {
15675             m_os << m_indent;
15676         }
15677         m_os << "</" << m_tags.back() << ">";

```

```

15677     }
15678     m_os << std::flush;
15679     applyFormatting(fmt);
15680     m_tags.pop_back();
15681     return *this;
15682 }
15683
15684 XmlWriter& XmlWriter::writeAttribute( std::string const& name, std::string const& attribute ) {
15685     if( !name.empty() && !attribute.empty() )
15686         m_os << ' ' << name << "=" << XmlEncode( attribute, XmlEncode::ForAttributes ) << ' ';
15687     return *this;
15688 }
15689
15690 XmlWriter& XmlWriter::writeAttribute( std::string const& name, bool attribute ) {
15691     m_os << ' ' << name << "=" << ( attribute ? "true" : "false" ) << ' ';
15692     return *this;
15693 }
15694
15695 XmlWriter& XmlWriter::writeText( std::string const& text, XmlFormatting fmt ) {
15696     if( !text.empty() ){
15697         bool tagWasOpen = m_tagIsOpen;
15698         ensureTagClosed();
15699         if (tagWasOpen && shouldIndent(fmt)) {
15700             m_os << m_indent;
15701         }
15702         m_os << XmlEncode( text );
15703         applyFormatting(fmt);
15704     }
15705     return *this;
15706 }
15707
15708 XmlWriter& XmlWriter::writeComment( std::string const& text, XmlFormatting fmt ) {
15709     ensureTagClosed();
15710     if (shouldIndent(fmt)) {
15711         m_os << m_indent;
15712     }
15713     m_os << "<!--" << text << "-->";
15714     applyFormatting(fmt);
15715     return *this;
15716 }
15717
15718 void XmlWriter::writeStylesheetRef( std::string const& url ) {
15719     m_os << "<?xml-stylesheet type=\"text/xsl\" href=\"" << url << "\"?>\n";
15720 }
15721
15722 XmlWriter& XmlWriter::writeBlankLine() {
15723     ensureTagClosed();
15724     m_os << '\n';
15725     return *this;
15726 }
15727
15728 void XmlWriter::ensureTagClosed() {
15729     if( m_tagIsOpen ) {
15730         m_os << '>' << std::flush;
15731         newlineIfNecessary();
15732         m_tagIsOpen = false;
15733     }
15734 }
15735
15736 void XmlWriter::applyFormatting(XmlFormatting fmt) {
15737     m_needsNewline = shouldNewline(fmt);
15738 }
15739
15740 void XmlWriter::writeDeclaration() {
15741     m_os << "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
15742 }
15743
15744 void XmlWriter::newlineIfNecessary() {
15745     if( m_needsNewline ) {
15746         m_os << std::endl;
15747         m_needsNewline = false;
15748     }
15749 }
15750 }
15751 // end catch_xmlwriter.cpp
15752 // start catch_reporter_bases.cpp
15753
15754 #include <cstring>
15755 #include <cfloat>
15756 #include <cstdio>
15757 #include <cassert>
15758 #include <memory>
15759
15760 namespace Catch {
15761     void prepareExpandedExpression( AssertionResult& result ) {
15762         result.getExpandedExpression();
15763     }

```



```

15764
15765 // Because formatting using c++ streams is stateful, drop down to C is required
15766 // Alternatively we could use stringstream, but its performance is... not good.
15767 std::string getFormattedDuration( double duration ) {
15768     // Max exponent + 1 is required to represent the whole part
15769     // + 1 for decimal point
15770     // + 3 for the 3 decimal places
15771     // + 1 for null terminator
15772     const std::size_t maxDoubleSize = DBL_MAX_10_EXP + 1 + 1 + 3 + 1;
15773     char buffer[maxDoubleSize];
15774
15775     // Save previous errno, to prevent sprintf from overwriting it
15776     ErrnoGuard guard;
15777 #ifdef _MSC_VER
15778     sprintf_s(buffer, "%.3f", duration);
15779 #else
15780     std::sprintf(buffer, "%.3f", duration);
15781 #endif
15782     return std::string(buffer);
15783 }
15784
15785 bool shouldShowDuration( IConfig const& config, double duration ) {
15786     if ( config.showDurations() == ShowDurations::Always ) {
15787         return true;
15788     }
15789     if ( config.showDurations() == ShowDurations::Never ) {
15790         return false;
15791     }
15792     const double min = config.minDuration();
15793     return min >= 0 && duration >= min;
15794 }
15795
15796 std::string serializeFilters( std::vector<std::string> const& container ) {
15797     ReusableStringStream oss;
15798     bool first = true;
15799     for (auto&& filter : container)
15800     {
15801         if (!first)
15802             oss << ' ';
15803         else
15804             first = false;
15805
15806         oss << filter;
15807     }
15808     return oss.str();
15809 }
15810
15811 TestEventListenerBase::TestEventListenerBase(ReporterConfig const & _config)
15812     :StreamingReporterBase(_config) {}
15813
15814 std::set<Verbosity> TestEventListenerBase::getSupportedVerbsities() {
15815     return { Verbosity::Quiet, Verbosity::Normal, Verbosity::High };
15816 }
15817
15818 void TestEventListenerBase::assertionStarting(AssertionInfo const &) {}
15819
15820 bool TestEventListenerBase::assertionEnded(AssertionStats const &) {
15821     return false;
15822 }
15823
15824 } // end namespace Catch
15825 // end catch_reporter_bases.cpp
15826 // start catch_reporter_compact.cpp
15827
15828 namespace {
15829
15830 #ifdef CATCH_PLATFORM_MAC
15831     const char* failedString() { return "FAILED"; }
15832     const char* passedString() { return "PASSED"; }
15833 #else
15834     const char* failedString() { return "failed"; }
15835     const char* passedString() { return "passed"; }
15836 #endif
15837
15838     // Colour::LightGrey
15839     Catch::Colour::Code dimColour() { return Catch::Colour::FileName; }
15840
15841     std::string bothOrAll( std::size_t count ) {
15842         return count == 1 ? std::string() :
15843             count == 2 ? "both " : "all " ;
15844     }
15845 }
15846 // anon namespace
15847 namespace Catch {
15848     namespace {
15849         // Colour, message variants:
15850

```

```

15851 // - white: No tests ran.
15852 // - red: Failed [both/all] N test cases, failed [both/all] M assertions.
15853 // - white: Passed [both/all] N test cases (no assertions).
15854 // - red: Failed N tests cases, failed M assertions.
15855 // - green: Passed [both/all] N tests cases with M assertions.
15856 void printTotals(std::ostream& out, const Totals& totals) {
15857     if (totals.testCases.total() == 0) {
15858         out << "No tests ran.";
15859     } else if (totals.testCases.failed == totals.testCases.total()) {
15860         Colour colour(Colour::ResultError);
15861         const std::string qualify_assertions_failed =
15862             totals.assertions.failed == totals.assertions.total() ?
15863             bothOrAll(totals.assertions.failed) : std::string();
15864         out <<
15865             "Failed " << bothOrAll(totals.testCases.failed)
15866             << pluralise(totals.testCases.failed, "test case") << ", "
15867             << "failed " << qualify_assertions_failed <<
15868             pluralise(totals.assertions.failed, "assertion") << ' ';
15869     } else if (totals.assertions.total() == 0) {
15870         out <<
15871             "Passed " << bothOrAll(totals.testCases.total())
15872             << pluralise(totals.testCases.total(), "test case")
15873             << " (no assertions).";
15874     } else if (totals.assertions.failed) {
15875         Colour colour(Colour::ResultError);
15876         out <<
15877             "Failed " << pluralise(totals.testCases.failed, "test case") << ", "
15878             << "failed " << pluralise(totals.assertions.failed, "assertion") << ' ';
15879     } else {
15880         Colour colour(Colour::ResultSuccess);
15881         out <<
15882             "Passed " << bothOrAll(totals.testCases.passed)
15883             << pluralise(totals.testCases.passed, "test case") <<
15884             " with " << pluralise(totals.assertions.passed, "assertion") << ' ';
15885     }
15886 }
15887
15888 // Implementation of CompactReporter formatting
15889 class AssertionPrinter {
15890 public:
15891     AssertionPrinter& operator= (AssertionPrinter const&) = delete;
15892     AssertionPrinter(AssertionPrinter const&) = delete;
15893     AssertionPrinter(std::ostream& _stream, AssertionStats const& _stats, bool _printInfoMessages)
15894         : stream(_stream)
15895         , result(_stats.assertionResult)
15896         , messages(_stats.infoMessages)
15897         , itMessage(_stats.infoMessages.begin())
15898         , printInfoMessages(_printInfoMessages) {}
15899
15900     void print() {
15901         printSourceInfo();
15902
15903         itMessage = messages.begin();
15904
15905         switch (result.getResultType()) {
15906             case ResultWas::Ok:
15907                 printResultType(Colour::ResultSuccess, passedString());
15908                 printOriginalExpression();
15909                 printReconstructedExpression();
15910                 if (!result.hasExpression())
15911                     printRemainingMessages(Colour::None);
15912                 else
15913                     printRemainingMessages();
15914                 break;
15915             case ResultWas::ExpressionFailed:
15916                 if (result.isOk())
15917                     printResultType(Colour::ResultSuccess, failedString() + std::string(" - but was ok"));
15918                 else
15919                     printResultType(Colour::Error, failedString());
15920                 printOriginalExpression();
15921                 printReconstructedExpression();
15922                 printRemainingMessages();
15923                 break;
15924             case ResultWas::ThrewException:
15925                 printResultType(Colour::Error, failedString());
15926                 printIssue("unexpected exception with message:");
15927                 printMessage();
15928                 printExpressionWas();
15929                 printRemainingMessages();
15930                 break;
15931             case ResultWas::FatalErrorCondition:
15932                 printResultType(Colour::Error, failedString());
15933                 printIssue("fatal error condition with message:");
15934                 printMessage();
15935                 printExpressionWas();
15936                 printRemainingMessages();
15937                 break;

```

```

15938         case ResultWas::DidntThrowException:
15939             printResultType(Colour::Error, failedString());
15940             printIssue("expected exception, got none");
15941             printExpressionWas();
15942             printRemainingMessages();
15943             break;
15944         case ResultWas::Info:
15945             printResultType(Colour::None, "info");
15946             printMessage();
15947             printRemainingMessages();
15948             break;
15949         case ResultWas::Warning:
15950             printResultType(Colour::None, "warning");
15951             printMessage();
15952             printRemainingMessages();
15953             break;
15954         case ResultWas::ExplicitFailure:
15955             printResultType(Colour::Error, failedString());
15956             printIssue("explicitly");
15957             printRemainingMessages(Colour::None);
15958             break;
15959             // These cases are here to prevent compiler warnings
15960         case ResultWas::Unknown:
15961         case ResultWas::FailureBit:
15962         case ResultWas::Exception:
15963             printResultType(Colour::Error, "** internal error **");
15964             break;
15965     }
15966 }
15967
15968 private:
15969     void printSourceInfo() const {
15970         Colour colourGuard(Colour::FileName);
15971         stream << result.getSourceInfo() << ":'";
15972     }
15973
15974     void printResultType(Colour::Code colour, std::string const& passOrFail) const {
15975         if (!passOrFail.empty()) {
15976             {
15977                 Colour colourGuard(colour);
15978                 stream << " " << passOrFail;
15979             }
15980             stream << ":'";
15981         }
15982     }
15983
15984     void printIssue(std::string const& issue) const {
15985         stream << " " << issue;
15986     }
15987
15988     void printExpressionWas() {
15989         if (result.hasExpression()) {
15990             stream << ":'";
15991             {
15992                 Colour colour(dimColour());
15993                 stream << " expression was:";
15994             }
15995             printOriginalExpression();
15996         }
15997     }
15998
15999     void printOriginalExpression() const {
16000         if (result.hasExpression()) {
16001             stream << " " << result.getExpression();
16002         }
16003     }
16004
16005     void printReconstructedExpression() const {
16006         if (result.hasExpandedExpression()) {
16007             {
16008                 Colour colour(dimColour());
16009                 stream << " for: ";
16010             }
16011             stream << result.getExpandedExpression();
16012         }
16013     }
16014
16015     void printMessage() {
16016         if (itMessage != messages.end()) {
16017             stream << " " << itMessage->message << ":'";
16018             ++itMessage;
16019         }
16020     }
16021
16022     void printRemainingMessages(Colour::Code colour = dimColour()) {
16023         if (itMessage == messages.end())
16024             return;

```

```

16025
16026     const auto itEnd = messages.cend();
16027     const auto N = static_cast<std::size_t>(std::distance(itMessage, itEnd));
16028
16029     {
16030         Colour colourGuard(colour);
16031         stream << " with " << pluralise(N, "message") << ':';
16032     }
16033
16034     while (itMessage != itEnd) {
16035         // If this assertion is a warning ignore any INFO messages
16036         if (printInfoMessages || itMessage->type != ResultWas::Info) {
16037             printMessage();
16038             if (itMessage != itEnd) {
16039                 Colour colourGuard(dimColour());
16040                 stream << " and";
16041             }
16042             continue;
16043         }
16044         ++itMessage;
16045     }
16046 }
16047
16048 private:
16049     std::ostream& stream;
16050     AssertionResult const& result;
16051     std::vector<MessageInfo> messages;
16052     std::vector<MessageInfo>::const_iterator itMessage;
16053     bool printInfoMessages;
16054 };
16055
16056 } // anon namespace
16057
16058     std::string CompactReporter::getDescription() {
16059         return "Reports test results on a single line, suitable for IDEs";
16060     }
16061
16062     void CompactReporter::noMatchingTestCases( std::string const& spec ) {
16063         stream << "No test cases matched '" << spec << "'\n" << std::endl;
16064     }
16065
16066     void CompactReporter::assertionStarting( AssertionInfo const& ) {}
16067
16068     bool CompactReporter::assertionEnded( AssertionStats const& _assertionStats ) {
16069         AssertionResult const& result = _assertionStats.assertionResult;
16070
16071         bool printInfoMessages = true;
16072
16073         // Drop out if result was successful and we're not printing those
16074         if ( !m_config->includeSuccessfulResults() && result.isOk() ) {
16075             if ( result.getResultType() != ResultWas::Warning )
16076                 return false;
16077             printInfoMessages = false;
16078         }
16079
16080         AssertionPrinter printer( stream, _assertionStats, printInfoMessages );
16081         printer.print();
16082
16083         stream << std::endl;
16084         return true;
16085     }
16086
16087     void CompactReporter::sectionEnded(SectionStats const& _sectionStats) {
16088         double dur = _sectionStats.durationInSeconds;
16089         if ( shouldShowDuration( *m_config, dur ) ) {
16090             stream << getFormattedDuration( dur ) << " s: " << _sectionStats.sectionInfo.name <<
16091             std::endl;
16092         }
16093     }
16094
16095     void CompactReporter::testRunEnded( TestRunStats const& _testRunStats ) {
16096         printTotals( stream, _testRunStats.totals );
16097         stream << '\n' << std::endl;
16098         StreamingReporterBase::testRunEnded( _testRunStats );
16099     }
16100
16101     CompactReporter::~CompactReporter() {}
16102
16103     CATCH_REGISTER_REPORTER( "compact", CompactReporter )
16104 } // end namespace Catch
16105 // end catch_reporter_compact.cpp
16106 // start catch_reporter_console.cpp
16107
16108 #include <cfloat>
16109 #include <cstdio>
16110

```

```

16111 #if defined(_MSC_VER)
16112 #pragma warning(push)
16113 #pragma warning(disable:4061) // Not all labels are EXPLICITLY handled in switch
16114 // Note that 4062 (not all labels are handled and default is missing) is enabled
16115 #endif
16116
16117 #if defined(__clang__)
16118 # pragma clang diagnostic push
16119 // For simplicity, benchmarking-only helpers are always enabled
16120 # pragma clang diagnostic ignored "-Wunused-function"
16121 #endif
16122
16123 namespace Catch {
16124
16125     namespace {
16126
16127         // Formatter impl for ConsoleReporter
16128         class ConsoleAssertionPrinter {
16129         public:
16130             ConsoleAssertionPrinter& operator= (ConsoleAssertionPrinter const&) = delete;
16131             ConsoleAssertionPrinter(ConsoleAssertionPrinter const&) = delete;
16132             ConsoleAssertionPrinter(std::ostream& _stream, AssertionStats const& _stats, bool
                _printInfoMessages)
16133                 : stream(_stream),
16134                   stats(_stats),
16135                   result(_stats.assertionResult),
16136                   colour(Colour::None),
16137                   message(result.getMessage()),
16138                   messages(_stats.infoMessages),
16139                   printInfoMessages(_printInfoMessages) {
16140                 switch (result.getResultType()) {
16141                     case ResultWas::Ok:
16142                         colour = Colour::Success;
16143                         passOrFail = "PASSED";
16144                         //if( result.hasMessage() )
16145                         if (_stats.infoMessages.size() == 1)
16146                             messageLabel = "with message";
16147                         if (_stats.infoMessages.size() > 1)
16148                             messageLabel = "with messages";
16149                         break;
16150                     case ResultWas::ExpressionFailed:
16151                         if (result.isOk()) {
16152                             colour = Colour::Success;
16153                             passOrFail = "FAILED - but was ok";
16154                         } else {
16155                             colour = Colour::Error;
16156                             passOrFail = "FAILED";
16157                         }
16158                         if (_stats.infoMessages.size() == 1)
16159                             messageLabel = "with message";
16160                         if (_stats.infoMessages.size() > 1)
16161                             messageLabel = "with messages";
16162                         break;
16163                     case ResultWas::ThrewException:
16164                         colour = Colour::Error;
16165                         passOrFail = "FAILED";
16166                         messageLabel = "due to unexpected exception with ";
16167                         if (_stats.infoMessages.size() == 1)
16168                             messageLabel += "message";
16169                         if (_stats.infoMessages.size() > 1)
16170                             messageLabel += "messages";
16171                         break;
16172                     case ResultWas::FatalErrorCondition:
16173                         colour = Colour::Error;
16174                         passOrFail = "FAILED";
16175                         messageLabel = "due to a fatal error condition";
16176                         break;
16177                     case ResultWas::DidntThrowException:
16178                         colour = Colour::Error;
16179                         passOrFail = "FAILED";
16180                         messageLabel = "because no exception was thrown where one was expected";
16181                         break;
16182                     case ResultWas::Info:
16183                         messageLabel = "info";
16184                         break;
16185                     case ResultWas::Warning:
16186                         messageLabel = "warning";
16187                         break;
16188                     case ResultWas::ExplicitFailure:
16189                         passOrFail = "FAILED";
16190                         colour = Colour::Error;
16191                         if (_stats.infoMessages.size() == 1)
16192                             messageLabel = "explicitly with message";
16193                         if (_stats.infoMessages.size() > 1)
16194                             messageLabel = "explicitly with messages";
16195                         break;
16196                     // These cases are here to prevent compiler warnings

```

```

16197         case ResultWas::Unknown:
16198         case ResultWas::FailureBit:
16199         case ResultWas::Exception:
16200             passOrFail = "*** internal error ***";
16201             colour = Colour::Error;
16202             break;
16203     }
16204 }
16205
16206 void print() const {
16207     printSourceInfo();
16208     if (stats.totals.assertions.total() > 0) {
16209         printResultType();
16210         printOriginalExpression();
16211         printReconstructedExpression();
16212     } else {
16213         stream << '\n';
16214     }
16215     printMessage();
16216 }
16217
16218 private:
16219     void printResultType() const {
16220         if (!passOrFail.empty()) {
16221             Colour colourGuard(colour);
16222             stream << passOrFail << ":\n";
16223         }
16224     }
16225     void printOriginalExpression() const {
16226         if (result.hasExpression()) {
16227             Colour colourGuard(Colour::OriginalExpression);
16228             stream << " ";
16229             stream << result.getExpressionInMacro();
16230             stream << '\n';
16231         }
16232     }
16233     void printReconstructedExpression() const {
16234         if (result.hasExpandedExpression()) {
16235             stream << "with expansion:\n";
16236             Colour colourGuard(Colour::ReconstructedExpression);
16237             stream << Column(result.getExpandedExpression()).indent(2) << '\n';
16238         }
16239     }
16240     void printMessage() const {
16241         if (!messageLabel.empty())
16242             stream << messageLabel << ':' << '\n';
16243         for (auto const& msg : messages) {
16244             // If this assertion is a warning ignore any INFO messages
16245             if (printInfoMessages || msg.type != ResultWas::Info)
16246                 stream << Column(msg.message).indent(2) << '\n';
16247         }
16248     }
16249     void printSourceInfo() const {
16250         Colour colourGuard(Colour::FileName);
16251         stream << result.getSourceInfo() << ":\n";
16252     }
16253
16254     std::ostream& stream;
16255     AssertionStats const& stats;
16256     AssertionResult const& result;
16257     Colour::Code colour;
16258     std::string passOrFail;
16259     std::string messageLabel;
16260     std::string message;
16261     std::vector<MessageInfo> messages;
16262     bool printInfoMessages;
16263 };
16264
16265 std::size_t makeRatio(std::size_t number, std::size_t total) {
16266     std::size_t ratio = total > 0 ? CATCH_CONFIG_CONSOLE_WIDTH * number / total : 0;
16267     return (ratio == 0 && number > 0) ? 1 : ratio;
16268 }
16269
16270 std::size_t& findMax(std::size_t& i, std::size_t& j, std::size_t& k) {
16271     if (i > j && i > k)
16272         return i;
16273     else if (j > k)
16274         return j;
16275     else
16276         return k;
16277 }
16278
16279 struct ColumnInfo {
16280     enum Justification { Left, Right };
16281     std::string name;
16282     int width;
16283     Justification justification;

```

```

16284 };
16285 struct ColumnBreak {};
16286 struct RowBreak {};
16287
16288 class Duration {
16289     enum class Unit {
16290         Auto,
16291         Nanoseconds,
16292         Microseconds,
16293         Milliseconds,
16294         Seconds,
16295         Minutes
16296     };
16297     static const uint64_t s_nanosecondsInAMicrosecond = 1000;
16298     static const uint64_t s_nanosecondsInAMillisecond = 1000 * s_nanosecondsInAMicrosecond;
16299     static const uint64_t s_nanosecondsInASecond = 1000 * s_nanosecondsInAMillisecond;
16300     static const uint64_t s_nanosecondsInAMinute = 60 * s_nanosecondsInASecond;
16301
16302     double m_inNanoseconds;
16303     Unit m_units;
16304
16305 public:
16306     explicit Duration(double inNanoseconds, Unit units = Unit::Auto)
16307         : m_inNanoseconds(inNanoseconds),
16308           m_units(units) {
16309         if (m_units == Unit::Auto) {
16310             if (m_inNanoseconds < s_nanosecondsInAMicrosecond)
16311                 m_units = Unit::Nanoseconds;
16312             else if (m_inNanoseconds < s_nanosecondsInAMillisecond)
16313                 m_units = Unit::Microseconds;
16314             else if (m_inNanoseconds < s_nanosecondsInASecond)
16315                 m_units = Unit::Milliseconds;
16316             else if (m_inNanoseconds < s_nanosecondsInAMinute)
16317                 m_units = Unit::Seconds;
16318             else
16319                 m_units = Unit::Minutes;
16320         }
16321     }
16322
16323     auto value() const -> double {
16324         switch (m_units) {
16325             case Unit::Microseconds:
16326                 return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMicrosecond);
16327             case Unit::Milliseconds:
16328                 return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMillisecond);
16329             case Unit::Seconds:
16330                 return m_inNanoseconds / static_cast<double>(s_nanosecondsInASecond);
16331             case Unit::Minutes:
16332                 return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMinute);
16333             default:
16334                 return m_inNanoseconds;
16335         }
16336     }
16337
16338     auto unitsAsString() const -> std::string {
16339         switch (m_units) {
16340             case Unit::Nanoseconds:
16341                 return "ns";
16342             case Unit::Microseconds:
16343                 return "us";
16344             case Unit::Milliseconds:
16345                 return "ms";
16346             case Unit::Seconds:
16347                 return "s";
16348             case Unit::Minutes:
16349                 return "m";
16350             default:
16351                 return "** internal error **";
16352         }
16353     }
16354
16355     friend auto operator << (std::ostream& os, Duration const& duration) -> std::ostream& {
16356         return os << duration.value() << ' ' << duration.unitsAsString();
16357     }
16358 };
16359 } // end anon namespace
16360
16361 class TablePrinter {
16362     std::ostream& m_os;
16363     std::vector<ColumnInfo> m_columnInfos;
16364     std::ostringstream m_oss;
16365     int m_currentColumn = -1;
16366     bool m_isOpen = false;
16367
16368 public:
16369     TablePrinter(std::ostream& os, std::vector<ColumnInfo> columnInfos)
16370         : m_os(os),

```

```

16371         m_columnInfos( std::move( columnInfos ) ) {}
16372
16373     auto columnInfos() const -> std::vector<ColumnInfo> const& {
16374         return m_columnInfos;
16375     }
16376
16377     void open() {
16378         if (!m_isOpen) {
16379             m_isOpen = true;
16380             *this << RowBreak();
16381
16382             Columns headerCols;
16383             Spacer spacer(2);
16384             for (auto const& info : m_columnInfos) {
16385                 headerCols += Column(info.name).width(static_cast<std::size_t>(info.width - 2));
16386                 headerCols += spacer;
16387             }
16388             m_os << headerCols << '\n';
16389
16390             m_os << Catch::getLineOfChars<'-'>() << '\n';
16391         }
16392     }
16393
16394     void close() {
16395         if (m_isOpen) {
16396             *this << RowBreak();
16397             m_os << std::endl;
16398             m_isOpen = false;
16399         }
16400     }
16401
16402     template<typename T>
16403     friend TablePrinter& operator << (TablePrinter& tp, T const& value) {
16404         tp.m_oss << value;
16405         return tp;
16406     }
16407
16408     friend TablePrinter& operator << (TablePrinter& tp, ColumnBreak) {
16409         auto colStr = tp.m_oss.str();
16410         const auto strSize = colStr.size();
16411         tp.m_oss.str("");
16412         tp.open();
16413         if (tp.m_currentColumn == static_cast<int>(tp.m_columnInfos.size() - 1)) {
16414             tp.m_currentColumn = -1;
16415             tp.m_os << '\n';
16416         }
16417         tp.m_currentColumn++;
16418
16419         auto colInfo = tp.m_columnInfos[tp.m_currentColumn];
16420         auto padding = (strSize + 1 < static_cast<std::size_t>(colInfo.width))
16421             ? std::string(colInfo.width - (strSize + 1), ' ')
16422             : std::string();
16423         if (colInfo.justification == ColumnInfo::Left)
16424             tp.m_os << colStr << padding << ' ';
16425         else
16426             tp.m_os << padding << colStr << ' ';
16427         return tp;
16428     }
16429
16430     friend TablePrinter& operator << (TablePrinter& tp, RowBreak) {
16431         if (tp.m_currentColumn > 0) {
16432             tp.m_os << '\n';
16433             tp.m_currentColumn = -1;
16434         }
16435         return tp;
16436     }
16437 };
16438
16439 ConsoleReporter::ConsoleReporter(ReporterConfig const& config)
16440 : StreamingReporterBase(config),
16441   m_tablePrinter(new TablePrinter(config.stream(),
16442                                   [&config]() -> std::vector<ColumnInfo> {
16443                                       if (config.fullConfig()->benchmarkNoAnalysis())
16444                                           {
16445                                               return{
16446                                                   { "benchmark name", CATCH_CONFIG_CONSOLE_WIDTH - 43, ColumnInfo::Left },
16447                                                   { "      samples", 14, ColumnInfo::Right },
16448                                                   { "   iterations", 14, ColumnInfo::Right },
16449                                                   { "           mean", 14, ColumnInfo::Right }
16450                                               };
16451                                           }
16452                                       else
16453                                           {
16454                                               return{
16455                                                   { "benchmark name", CATCH_CONFIG_CONSOLE_WIDTH - 43, ColumnInfo::Left },
16456                                                   { "samples      mean      std dev", 14, ColumnInfo::Right },
16457                                                   { "iterations  low mean  low std dev", 14, ColumnInfo::Right },
16458                                                   { "estimated   high mean high std dev", 14, ColumnInfo::Right }
16459                                           };
16460                                       }
16461                                   })

```



```

16458         };
16459     }
16460     }()) {}
16461     ConsoleReporter::~ConsoleReporter() = default;
16462
16463     std::string ConsoleReporter::getDescription() {
16464         return "Reports test results as plain lines of text";
16465     }
16466
16467     void ConsoleReporter::noMatchingTestCases(std::string const& spec) {
16468         stream << "No test cases matched '" << spec << "'\n" << std::endl;
16469     }
16470
16471     void ConsoleReporter::reportInvalidArguments(std::string const& arg) {
16472         stream << "Invalid Filter: " << arg << std::endl;
16473     }
16474
16475     void ConsoleReporter::assertionStarting(AssertionInfo const&) {}
16476
16477     bool ConsoleReporter::assertionEnded(AssertionStats const& _assertionStats) {
16478         AssertionResult const& result = _assertionStats.assertionResult;
16479
16480         bool includeResults = m_config->includeSuccessfulResults() || !result.isOk();
16481
16482         // Drop out if result was successful but we're not printing them.
16483         if (!includeResults && result.getResultType() != ResultWas::Warning)
16484             return false;
16485
16486         lazyPrint();
16487
16488         ConsoleAssertionPrinter printer(stream, _assertionStats, includeResults);
16489         printer.print();
16490         stream << std::endl;
16491         return true;
16492     }
16493
16494     void ConsoleReporter::sectionStarting(SectionInfo const& _sectionInfo) {
16495         m_tablePrinter->close();
16496         m_headerPrinted = false;
16497         StreamingReporterBase::sectionStarting(_sectionInfo);
16498     }
16499     void ConsoleReporter::sectionEnded(SectionStats const& _sectionStats) {
16500         m_tablePrinter->close();
16501         if (_sectionStats.missingAssertions) {
16502             lazyPrint();
16503             Colour colour(Colour::ResultError);
16504             if (m_sectionStack.size() > 1)
16505                 stream << "\nNo assertions in section";
16506             else
16507                 stream << "\nNo assertions in test case";
16508             stream << " '" << _sectionStats.sectionInfo.name << "'\n" << std::endl;
16509         }
16510         double dur = _sectionStats.durationInSeconds;
16511         if (shouldShowDuration(*m_config, dur)) {
16512             stream << getFormattedDuration(dur) << " s: " << _sectionStats.sectionInfo.name << std::endl;
16513         }
16514         if (m_headerPrinted) {
16515             m_headerPrinted = false;
16516         }
16517         StreamingReporterBase::sectionEnded(_sectionStats);
16518     }
16519
16520     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
16521     void ConsoleReporter::benchmarkPreparing(std::string const& name) {
16522         lazyPrintWithoutClosingBenchmarkTable();
16523
16524         auto nameCol = Column(name).width(static_cast<std::size_t>(m_tablePrinter->columnInfos()[0].width
16525 - 2));
16526
16527         bool firstLine = true;
16528         for (auto line : nameCol) {
16529             if (!firstLine)
16530                 (*m_tablePrinter) << ColumnBreak() << ColumnBreak() << ColumnBreak();
16531             else
16532                 firstLine = false;
16533             (*m_tablePrinter) << line << ColumnBreak();
16534         }
16535     }
16536
16537     void ConsoleReporter::benchmarkStarting(BenchmarkInfo const& info) {
16538         (*m_tablePrinter) << info.samples << ColumnBreak()
16539             << info.iterations << ColumnBreak();
16540         if (!m_config->benchmarkNoAnalysis())
16541             (*m_tablePrinter) << Duration(info.estimatedDuration) << ColumnBreak();
16542     }
16543     void ConsoleReporter::benchmarkEnded(BenchmarkStats const& stats) {

```

```

16544     if (m_config->benchmarkNoAnalysis())
16545     {
16546         (*m_tablePrinter) << Duration(stats.mean.point.count()) << ColumnBreak();
16547     }
16548     else
16549     {
16550         (*m_tablePrinter) << ColumnBreak()
16551             << Duration(stats.mean.point.count()) << ColumnBreak()
16552             << Duration(stats.mean.lower_bound.count()) << ColumnBreak()
16553             << Duration(stats.mean.upper_bound.count()) << ColumnBreak() << ColumnBreak()
16554             << Duration(stats.standardDeviation.point.count()) << ColumnBreak()
16555             << Duration(stats.standardDeviation.lower_bound.count()) << ColumnBreak()
16556             << Duration(stats.standardDeviation.upper_bound.count()) << ColumnBreak() << ColumnBreak() <<
ColumnBreak() << ColumnBreak() << ColumnBreak();
16557     }
16558 }
16559
16560 void ConsoleReporter::benchmarkFailed(std::string const& error) {
16561     Colour colour(Colour::Red);
16562     (*m_tablePrinter)
16563         << "Benchmark failed (" << error << ')'
16564         << ColumnBreak() << RowBreak();
16565 }
16566 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
16567
16568 void ConsoleReporter::testCaseEnded(TestCaseStats const& _testCaseStats) {
16569     m_tablePrinter->close();
16570     StreamingReporterBase::testCaseEnded(_testCaseStats);
16571     m_headerPrinted = false;
16572 }
16573 void ConsoleReporter::testGroupEnded(TestGroupStats const& _testGroupStats) {
16574     if (currentGroupInfo.used) {
16575         printSummaryDivider();
16576         stream << "Summary for group '" << _testGroupStats.groupInfo.name << "':\n";
16577         printTotals(_testGroupStats.totals);
16578         stream << '\n' << std::endl;
16579     }
16580     StreamingReporterBase::testGroupEnded(_testGroupStats);
16581 }
16582 void ConsoleReporter::testRunEnded(TestRunStats const& _testRunStats) {
16583     printTotalsDivider(_testRunStats.totals);
16584     printTotals(_testRunStats.totals);
16585     stream << std::endl;
16586     StreamingReporterBase::testRunEnded(_testRunStats);
16587 }
16588 void ConsoleReporter::testRunStarting(TestRunInfo const& _testInfo) {
16589     StreamingReporterBase::testRunStarting(_testInfo);
16590     printTestFilters();
16591 }
16592
16593 void ConsoleReporter::lazyPrint() {
16594
16595     m_tablePrinter->close();
16596     lazyPrintWithoutClosingBenchmarkTable();
16597 }
16598
16599 void ConsoleReporter::lazyPrintWithoutClosingBenchmarkTable() {
16600
16601     if (!currentTestRunInfo.used)
16602         lazyPrintRunInfo();
16603     if (!currentGroupInfo.used)
16604         lazyPrintGroupInfo();
16605
16606     if (!m_headerPrinted) {
16607         printTestCaseAndSectionHeader();
16608         m_headerPrinted = true;
16609     }
16610 }
16611 void ConsoleReporter::lazyPrintRunInfo() {
16612     stream << '\n' << getLineOfChars<'~'>() << '\n';
16613     Colour colour(Colour::SecondaryText);
16614     stream << currentTestRunInfo->name
16615         << " is a Catch v" << libraryVersion() << " host application.\n"
16616         << "Run with -? for options\n\n";
16617
16618     if (m_config->rngSeed() != 0)
16619         stream << "Randomness seeded to: " << m_config->rngSeed() << "\n\n";
16620
16621     currentTestRunInfo.used = true;
16622 }
16623 void ConsoleReporter::lazyPrintGroupInfo() {
16624     if (!currentGroupInfo->name.empty() && currentGroupInfo->groupsCounts > 1) {
16625         printClosedHeader("Group: " + currentGroupInfo->name);
16626         currentGroupInfo.used = true;
16627     }
16628 }
16629 void ConsoleReporter::printTestCaseAndSectionHeader() {

```

```

16630     assert(!m_sectionStack.empty());
16631     printOpenHeader(currentTestCaseInfo->name);
16632
16633     if (m_sectionStack.size() > 1) {
16634         Colour colourGuard(Colour::Headers);
16635
16636         auto
16637             it = m_sectionStack.begin() + 1, // Skip first section (test case)
16638             itEnd = m_sectionStack.end();
16639         for (; it != itEnd; ++it)
16640             printHeaderString(it->name, 2);
16641     }
16642
16643     SourceLineInfo lineInfo = m_sectionStack.back().lineInfo;
16644
16645     stream << getLineOfChars<'-'>() << '\n';
16646     Colour colourGuard(Colour::FileName);
16647     stream << lineInfo << '\n';
16648     stream << getLineOfChars<'.'>() << '\n' << std::endl;
16649 }
16650
16651 void ConsoleReporter::printClosedHeader(std::string const& _name) {
16652     printOpenHeader(_name);
16653     stream << getLineOfChars<'.'>() << '\n';
16654 }
16655 void ConsoleReporter::printOpenHeader(std::string const& _name) {
16656     stream << getLineOfChars<'-'>() << '\n';
16657     {
16658         Colour colourGuard(Colour::Headers);
16659         printHeaderString(_name);
16660     }
16661 }
16662
16663 // if string has a : in first line will set indent to follow it on
16664 // subsequent lines
16665 void ConsoleReporter::printHeaderString(std::string const& _string, std::size_t indent) {
16666     std::size_t i = _string.find(": ");
16667     if (i != std::string::npos)
16668         i += 2;
16669     else
16670         i = 0;
16671     stream << Column(_string).indent(indent + i).initialIndent(indent) << '\n';
16672 }
16673
16674 struct SummaryColumn {
16675
16676     SummaryColumn( std::string _label, Colour::Code _colour )
16677     :   label( std::move( _label ) ),
16678         colour( _colour ) {}
16679
16680     SummaryColumn addRow( std::size_t count ) {
16681         ReusableStringStream rss;
16682         rss << count;
16683         std::string row = rss.str();
16684         for (auto& oldRow : rows) {
16685             while (oldRow.size() < row.size())
16686                 oldRow = ' ' + oldRow;
16687             while (oldRow.size() > row.size())
16688                 row = ' ' + row;
16689         }
16690         rows.push_back(row);
16691         return *this;
16692     }
16693
16694     std::string label;
16695     Colour::Code colour;
16696     std::vector<std::string> rows;
16697 };
16698
16699 void ConsoleReporter::printTotals( Totals const& totals ) {
16700     if (totals.testCases.total() == 0) {
16701         stream << Colour(Colour::Warning) << "No tests ran\n";
16702     } else if (totals.assertions.total() > 0 && totals.testCases.allPassed()) {
16703         stream << Colour(Colour::ResultSuccess) << "All tests passed";
16704         stream << " ("
16705             << pluralise(totals.assertions.passed, "assertion") << " in "
16706             << pluralise(totals.testCases.passed, "test case") << ') '
16707             << '\n';
16708     } else {
16709
16710         std::vector<SummaryColumn> columns;
16711         columns.push_back(SummaryColumn(" ", Colour::None)
16712             .addRow(totals.testCases.total())
16713             .addRow(totals.assertions.total()));
16714         columns.push_back(SummaryColumn("passed", Colour::Success)
16715             .addRow(totals.testCases.passed)
16716             .addRow(totals.assertions.passed));

```

```

16717         columns.push_back(SummaryColumn("failed", Colour::ResultError)
16718                             .addRow(totals.testCases.failed)
16719                             .addRow(totals.assertions.failed));
16720         columns.push_back(SummaryColumn("failed as expected", Colour::ResultExpectedFailure)
16721                             .addRow(totals.testCases.failedButOk)
16722                             .addRow(totals.assertions.failedButOk));
16723
16724         printSummaryRow("test cases", columns, 0);
16725         printSummaryRow("assertions", columns, 1);
16726     }
16727 }
16728 void ConsoleReporter::printSummaryRow(std::string const& label, std::vector<SummaryColumn> const&
cols, std::size_t row) {
16729     for (auto col : cols) {
16730         std::string value = col.rows[row];
16731         if (col.label.empty()) {
16732             stream << label << ": ";
16733             if (value != "0")
16734                 stream << value;
16735             else
16736                 stream << Colour(Colour::Warning) << "- none -";
16737         } else if (value != "0") {
16738             stream << Colour(Colour::LightGrey) << " | ";
16739             stream << Colour(col.colour)
16740                 << value << ' ' << col.label;
16741         }
16742     }
16743     stream << '\n';
16744 }
16745
16746 void ConsoleReporter::printTotalsDivider(Totals const& totals) {
16747     if (totals.testCases.total() > 0) {
16748         std::size_t failedRatio = makeRatio(totals.testCases.failed, totals.testCases.total());
16749         std::size_t failedButOkRatio = makeRatio(totals.testCases.failedButOk,
totals.testCases.total());
16750         std::size_t passedRatio = makeRatio(totals.testCases.passed, totals.testCases.total());
16751         while (failedRatio + failedButOkRatio + passedRatio < CATCH_CONFIG_CONSOLE_WIDTH - 1)
16752             findMax(failedRatio, failedButOkRatio, passedRatio)++;
16753         while (failedRatio + failedButOkRatio + passedRatio > CATCH_CONFIG_CONSOLE_WIDTH - 1)
16754             findMax(failedRatio, failedButOkRatio, passedRatio)--;
16755
16756         stream << Colour(Colour::Error) << std::string(failedRatio, '=');
16757         stream << Colour(Colour::ResultExpectedFailure) << std::string(failedButOkRatio, '=');
16758         if (totals.testCases.allPassed())
16759             stream << Colour(Colour::ResultSuccess) << std::string(passedRatio, '=');
16760         else
16761             stream << Colour(Colour::Success) << std::string(passedRatio, '=');
16762     } else {
16763         stream << Colour(Colour::Warning) << std::string(CATCH_CONFIG_CONSOLE_WIDTH - 1, '=');
16764     }
16765     stream << '\n';
16766 }
16767 void ConsoleReporter::printSummaryDivider() {
16768     stream << getLineOfChars<'-'>() << '\n';
16769 }
16770
16771 void ConsoleReporter::printTestFilters() {
16772     if (m_config->testSpec().hasFilters()) {
16773         Colour guard(Colour::BrightYellow);
16774         stream << "Filters: " << serializeFilters(m_config->getTestsOrTags()) << '\n';
16775     }
16776 }
16777
16778 CATCH_REGISTER_REPORTER("console", ConsoleReporter)
16779
16780 } // end namespace Catch
16781
16782 #if defined(_MSC_VER)
16783 #pragma warning(pop)
16784 #endif
16785
16786 #if defined(__clang__)
16787 #pragma clang diagnostic pop
16788 #endif
16789 // end catch_reporter_console.cpp
16790 // start catch_reporter_junit.cpp
16791
16792 #include <cassert>
16793 #include <sstream>
16794 #include <ctime>
16795 #include <algorithm>
16796 #include <iomanip>
16797
16798 namespace Catch {
16799     namespace {
16800         std::string getTimestamp() {

```

```

16802         // Beware, this is not reentrant because of backward compatibility issues
16803         // Also, UTC only, again because of backward compatibility (%z is C++11)
16804         time_t rawtime;
16805         std::time(&rawtime);
16806         auto const timeStampSize = sizeof("2017-01-16T17:06:45Z");
16807
16808 #ifdef _MSC_VER
16809         std::tm timeInfo = {};
16810         gmtime_s(&timeInfo, &rawtime);
16811 #else
16812         std::tm* timeInfo;
16813         timeInfo = std::gmtime(&rawtime);
16814 #endif
16815
16816         char timeStamp[timeStampSize];
16817         const char * const fmt = "%Y-%m-%dT%H:%M:%SZ";
16818
16819 #ifdef _MSC_VER
16820         std::strftime(timeStamp, timeStampSize, fmt, &timeInfo);
16821 #else
16822         std::strftime(timeStamp, timeStampSize, fmt, timeInfo);
16823 #endif
16824         return std::string(timeStamp, timeStampSize-1);
16825     }
16826
16827     std::string fileNameTag(const std::vector<std::string> &tags) {
16828         auto it = std::find_if(begin(tags),
16829                                end(tags),
16830                                [](const std::string & tag) {return tag.front() == '#'; });
16831         if (it != tags.end())
16832             return it->substr(1);
16833         return std::string();
16834     }
16835
16836     // Formats the duration in seconds to 3 decimal places.
16837     // This is done because some genius defined Maven Surefire schema
16838     // in a way that only accepts 3 decimal places, and tools like
16839     // Jenkins use that schema for validation JUnit reporter output.
16840     std::string formatDuration( double seconds ) {
16841         ReusableStringStream rss;
16842         rss << std::fixed << std::setprecision( 3 ) << seconds;
16843         return rss.str();
16844     }
16845
16846 } // anonymous namespace
16847
16848 JUnitReporter::JUnitReporter( ReporterConfig const& _config )
16849 :   CumulativeReporterBase( _config ),
16850   xml( _config.stream() )
16851 {
16852     m_reporterPrefs.shouldRedirectStdOut = true;
16853     m_reporterPrefs.shouldReportAllAssertions = true;
16854 }
16855
16856 JUnitReporter::~JUnitReporter() {}
16857
16858 std::string JUnitReporter::getDescription() {
16859     return "Reports test results in an XML format that looks like Ant's junitreport target";
16860 }
16861
16862 void JUnitReporter::noMatchingTestCases( std::string const& /*spec*/ ) {}
16863
16864 void JUnitReporter::testRunStarting( TestRunInfo const& runInfo ) {
16865     CumulativeReporterBase::testRunStarting( runInfo );
16866     xml.startElement( "testsuites" );
16867 }
16868
16869 void JUnitReporter::testGroupStarting( GroupInfo const& groupInfo ) {
16870     suiteTimer.start();
16871     stdOutForSuite.clear();
16872     stdErrForSuite.clear();
16873     unexpectedExceptions = 0;
16874     CumulativeReporterBase::testGroupStarting( groupInfo );
16875 }
16876
16877 void JUnitReporter::testCaseStarting( TestCaseInfo const& testCaseInfo ) {
16878     m_okToFail = testCaseInfo.okToFail();
16879 }
16880
16881 bool JUnitReporter::assertionEnded( AssertionStats const& assertionStats ) {
16882     if( assertionStats.assertionResult.getResultType() == ResultWas::ThrewException && !m_okToFail )
16883         unexpectedExceptions++;
16884     return CumulativeReporterBase::assertionEnded( assertionStats );
16885 }
16886
16887 void JUnitReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {

```

```

16888         stdOutForSuite += testCaseStats.stdOut;
16889         stdErrForSuite += testCaseStats.stdErr;
16890         CumulativeReporterBase::testCaseEnded( testCaseStats );
16891     }
16892
16893     void JunitReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
16894         double suiteTime = suiteTimer.getElapsedSeconds();
16895         CumulativeReporterBase::testGroupEnded( testGroupStats );
16896         writeGroup( *m_testGroups.back(), suiteTime );
16897     }
16898
16899     void JunitReporter::testRunEndedCumulative() {
16900         xml.endElement();
16901     }
16902
16903     void JunitReporter::writeGroup( TestGroupNode const& groupNode, double suiteTime ) {
16904         XmlWriter::ScopedElement e = xml.scopedElement( "testsuite" );
16905
16906         TestGroupStats const& stats = groupNode.value;
16907         xml.writeAttribute( "name", stats.groupInfo.name );
16908         xml.writeAttribute( "errors", unexpectedExceptions );
16909         xml.writeAttribute( "failures", stats.totals.assertions.failed-unexpectedExceptions );
16910         xml.writeAttribute( "tests", stats.totals.assertions.total() );
16911         xml.writeAttribute( "hostname", "tbd" ); // !TBD
16912         if( m_config->showDurations() == ShowDurations::Never )
16913             xml.writeAttribute( "time", "" );
16914         else
16915             xml.writeAttribute( "time", formatDuration( suiteTime ) );
16916         xml.writeAttribute( "timestamp", getTimestamp() );
16917
16918         // Write properties if there are any
16919         if (m_config->hasTestFilters() || m_config->rngSeed() != 0) {
16920             auto properties = xml.scopedElement("properties");
16921             if (m_config->hasTestFilters()) {
16922                 xml.scopedElement("property")
16923                     .writeAttribute("name", "filters")
16924                     .writeAttribute("value", serializeFilters(m_config->getTestsOrTags()));
16925             }
16926             if (m_config->rngSeed() != 0) {
16927                 xml.scopedElement("property")
16928                     .writeAttribute("name", "random-seed")
16929                     .writeAttribute("value", m_config->rngSeed());
16930             }
16931         }
16932
16933         // Write test cases
16934         for( auto const& child : groupNode.children )
16935             writeTestCase( *child );
16936
16937         xml.scopedElement( "system-out" ).writeText( trim( stdOutForSuite ), XmlFormatting::Newline );
16938         xml.scopedElement( "system-err" ).writeText( trim( stdErrForSuite ), XmlFormatting::Newline );
16939     }
16940
16941     void JunitReporter::writeTestCase( TestCaseNode const& testCaseNode ) {
16942         TestCaseStats const& stats = testCaseNode.value;
16943
16944         // All test cases have exactly one section - which represents the
16945         // test case itself. That section may have 0-n nested sections
16946         assert( testCaseNode.children.size() == 1 );
16947         SectionNode const& rootSection = *testCaseNode.children.front();
16948
16949         std::string className = stats.testInfo.className;
16950
16951         if( className.empty() ) {
16952             className = fileNameTag(stats.testInfo.tags);
16953             if ( className.empty() )
16954                 className = "global";
16955         }
16956
16957         if ( !m_config->name().empty() )
16958             className = m_config->name() + "." + className;
16959
16960         writeSection( className, "", rootSection, stats.testInfo.okToFail() );
16961     }
16962
16963     void JunitReporter::writeSection( std::string const& className,
16964                                     std::string const& rootName,
16965                                     SectionNode const& sectionNode,
16966                                     bool testOkToFail ) {
16967         std::string name = trim( sectionNode.stats.sectionInfo.name );
16968         if( !rootName.empty() )
16969             name = rootName + '/' + name;
16970
16971         if( !sectionNode.assertions.empty() ||
16972             !sectionNode.stdOut.empty() ||
16973             !sectionNode.stdErr.empty() ) {
16974             XmlWriter::ScopedElement e = xml.scopedElement( "testcase" );

```

```

16975         if( className.empty() ) {
16976             xml.writeAttribute( "classname", name );
16977             xml.writeAttribute( "name", "root" );
16978         }
16979         else {
16980             xml.writeAttribute( "classname", className );
16981             xml.writeAttribute( "name", name );
16982         }
16983         xml.writeAttribute( "time", formatDuration( sectionNode.stats.durationInSeconds ) );
16984         // This is not ideal, but it should be enough to mimic gtest's
16985         // junit output.
16986         // Ideally the JUnit reporter would also handle `skipTest`
16987         // events and write those out appropriately.
16988         xml.writeAttribute( "status", "run" );
16989
16990         if (sectionNode.stats.assertions.failedButOk) {
16991             xml.scopedElement("skipped")
16992                 .writeAttribute("message", "TEST_CASE tagged with !mayfail");
16993         }
16994
16995         writeAssertions( sectionNode );
16996
16997         if( !sectionNode.stdOut.empty() )
16998             xml.scopedElement( "system-out" ).writeText( trim( sectionNode.stdOut ),
16999 XmlFormatting::Newline );
17000         if( !sectionNode.stdErr.empty() )
17001             xml.scopedElement( "system-err" ).writeText( trim( sectionNode.stdErr ),
17002 XmlFormatting::Newline );
17003     }
17004     for( auto const& childNode : sectionNode.childSections )
17005         if( className.empty() )
17006             writeSection( name, "", *childNode, testOkToFail );
17007         else
17008             writeSection( className, name, *childNode, testOkToFail );
17009 }
17010
17011 void JunitReporter::writeAssertions( SectionNode const& sectionNode ) {
17012     for( auto const& assertion : sectionNode.assertions )
17013         writeAssertion( assertion );
17014 }
17015
17016 void JunitReporter::writeAssertion( AssertionStats const& stats ) {
17017     AssertionResult const& result = stats.assertionResult;
17018     if( !result.isOk() ) {
17019         std::string elementName;
17020         switch( result.getResultType() ) {
17021             case ResultWas::ThrowException:
17022             case ResultWas::FatalErrorCondition:
17023                 elementName = "error";
17024                 break;
17025             case ResultWas::ExplicitFailure:
17026             case ResultWas::ExpressionFailed:
17027             case ResultWas::DidntThrowException:
17028                 elementName = "failure";
17029                 break;
17030
17031             // We should never see these here:
17032             case ResultWas::Info:
17033             case ResultWas::Warning:
17034             case ResultWas::Ok:
17035             case ResultWas::Unknown:
17036             case ResultWas::FailureBit:
17037             case ResultWas::Exception:
17038                 elementName = "internalError";
17039                 break;
17040         }
17041
17042         XmlWriter::ScopedElement e = xml.scopedElement( elementName );
17043
17044         xml.writeAttribute( "message", result.getExpression() );
17045         xml.writeAttribute( "type", result.getTestMacroName() );
17046
17047         ReusableStringStream rss;
17048         if (stats.totals.assertions.total() > 0) {
17049             rss << "FAILED" << ":\n";
17050             if (result.hasExpression()) {
17051                 rss << " ";
17052                 rss << result.getExpressionInMacro();
17053                 rss << '\n';
17054             }
17055             if (result.hasExpandedExpression()) {
17056                 rss << "with expansion:\n";
17057                 rss << Column(result.getExpandedExpression()).indent(2) << '\n';
17058             }
17059         } else {
17060             rss << '\n';
17061         }
17062     }

```

```

17060
17061         if( !result.getMessage().empty() )
17062             rss << result.getMessage() << '\n';
17063         for( auto const& msg : stats.infoMessages )
17064             if( msg.type == ResultWas::Info )
17065                 rss << msg.message << '\n';
17066
17067         rss << "at " << result.getSourceInfo();
17068         xml.writeText( rss.str(), XmlFormatting::Newline );
17069     }
17070 }
17071
17072 CATCH_REGISTER_REPORTER( "junit", JunitReporter )
17073
17074 } // end namespace Catch
17075 // end catch_reporter_junit.cpp
17076 // start catch_reporter_listening.cpp
17077
17078 #include <cassert>
17079
17080 namespace Catch {
17081
17082     ListeningReporter::ListeningReporter() {
17083         // We will assume that listeners will always want all assertions
17084         m_preferences.shouldReportAllAssertions = true;
17085     }
17086
17087     void ListeningReporter::addListener( IStreamingReporterPtr&& listener ) {
17088         m_listeners.push_back( std::move( listener ) );
17089     }
17090
17091     void ListeningReporter::addReporter( IStreamingReporterPtr&& reporter ) {
17092         assert( !m_reporter && "Listening reporter can wrap only 1 real reporter" );
17093         m_reporter = std::move( reporter );
17094         m_preferences.shouldRedirectStdOut = m_reporter->getPreferences().shouldRedirectStdOut;
17095     }
17096
17097     ReporterPreferences ListeningReporter::getPreferences() const {
17098         return m_preferences;
17099     }
17100
17101     std::set<Verbosity> ListeningReporter::getSupportedVerbsities() {
17102         return std::set<Verbosity>{ };
17103     }
17104
17105     void ListeningReporter::noMatchingTestCases( std::string const& spec ) {
17106         for ( auto const& listener : m_listeners ) {
17107             listener->noMatchingTestCases( spec );
17108         }
17109         m_reporter->noMatchingTestCases( spec );
17110     }
17111
17112     void ListeningReporter::reportInvalidArguments( std::string const& arg ) {
17113         for ( auto const& listener : m_listeners ) {
17114             listener->reportInvalidArguments( arg );
17115         }
17116         m_reporter->reportInvalidArguments( arg );
17117     }
17118
17119     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
17120     void ListeningReporter::benchmarkPreparing( std::string const& name ) {
17121         for ( auto const& listener : m_listeners ) {
17122             listener->benchmarkPreparing( name );
17123         }
17124         m_reporter->benchmarkPreparing( name );
17125     }
17126     void ListeningReporter::benchmarkStarting( BenchmarkInfo const& benchmarkInfo ) {
17127         for ( auto const& listener : m_listeners ) {
17128             listener->benchmarkStarting( benchmarkInfo );
17129         }
17130         m_reporter->benchmarkStarting( benchmarkInfo );
17131     }
17132     void ListeningReporter::benchmarkEnded( BenchmarkStats<> const& benchmarkStats ) {
17133         for ( auto const& listener : m_listeners ) {
17134             listener->benchmarkEnded( benchmarkStats );
17135         }
17136         m_reporter->benchmarkEnded( benchmarkStats );
17137     }
17138
17139     void ListeningReporter::benchmarkFailed( std::string const& error ) {
17140         for ( auto const& listener : m_listeners ) {
17141             listener->benchmarkFailed( error );
17142         }
17143         m_reporter->benchmarkFailed( error );
17144     }
17145 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
17146

```



```

17147     void ListeningReporter::testRunStarting( TestRunInfo const& testRunInfo ) {
17148         for ( auto const& listener : m_listeners ) {
17149             listener->testRunStarting( testRunInfo );
17150         }
17151         m_reporter->testRunStarting( testRunInfo );
17152     }
17153
17154     void ListeningReporter::testGroupStarting( GroupInfo const& groupInfo ) {
17155         for ( auto const& listener : m_listeners ) {
17156             listener->testGroupStarting( groupInfo );
17157         }
17158         m_reporter->testGroupStarting( groupInfo );
17159     }
17160
17161     void ListeningReporter::testCaseStarting( TestCaseInfo const& testInfo ) {
17162         for ( auto const& listener : m_listeners ) {
17163             listener->testCaseStarting( testInfo );
17164         }
17165         m_reporter->testCaseStarting( testInfo );
17166     }
17167
17168     void ListeningReporter::sectionStarting( SectionInfo const& sectionInfo ) {
17169         for ( auto const& listener : m_listeners ) {
17170             listener->sectionStarting( sectionInfo );
17171         }
17172         m_reporter->sectionStarting( sectionInfo );
17173     }
17174
17175     void ListeningReporter::assertionStarting( AssertionInfo const& assertionInfo ) {
17176         for ( auto const& listener : m_listeners ) {
17177             listener->assertionStarting( assertionInfo );
17178         }
17179         m_reporter->assertionStarting( assertionInfo );
17180     }
17181
17182     // The return value indicates if the messages buffer should be cleared:
17183     bool ListeningReporter::assertionEnded( AssertionStats const& assertionStats ) {
17184         for ( auto const& listener : m_listeners ) {
17185             static_cast<void>( listener->assertionEnded( assertionStats ) );
17186         }
17187         return m_reporter->assertionEnded( assertionStats );
17188     }
17189
17190     void ListeningReporter::sectionEnded( SectionStats const& sectionStats ) {
17191         for ( auto const& listener : m_listeners ) {
17192             listener->sectionEnded( sectionStats );
17193         }
17194         m_reporter->sectionEnded( sectionStats );
17195     }
17196
17197     void ListeningReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
17198         for ( auto const& listener : m_listeners ) {
17199             listener->testCaseEnded( testCaseStats );
17200         }
17201         m_reporter->testCaseEnded( testCaseStats );
17202     }
17203
17204     void ListeningReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
17205         for ( auto const& listener : m_listeners ) {
17206             listener->testGroupEnded( testGroupStats );
17207         }
17208         m_reporter->testGroupEnded( testGroupStats );
17209     }
17210
17211     void ListeningReporter::testRunEnded( TestRunStats const& testRunStats ) {
17212         for ( auto const& listener : m_listeners ) {
17213             listener->testRunEnded( testRunStats );
17214         }
17215         m_reporter->testRunEnded( testRunStats );
17216     }
17217
17218     void ListeningReporter::skipTest( TestCaseInfo const& testInfo ) {
17219         for ( auto const& listener : m_listeners ) {
17220             listener->skipTest( testInfo );
17221         }
17222         m_reporter->skipTest( testInfo );
17223     }
17224
17225     bool ListeningReporter::isMulti() const {
17226         return true;
17227     }
17228
17229 } // end namespace Catch
17230 // end catch_reporter_listening.cpp
17231 // start catch_reporter_xml.cpp
17232
17233 #if defined(_MSC_VER)

```

```

17234 #pragma warning(push)
17235 #pragma warning(disable:4061) // Not all labels are EXPLICITLY handled in switch
17236 // Note that 4062 (not all labels are handled
17237 // and default is missing) is enabled
17238 #endif
17239
17240 namespace Catch {
17241     XmlReporter::XmlReporter( ReporterConfig const& _config )
17242     :   StreamingReporterBase( _config ),
17243         m_xml(_config.stream())
17244     {
17245         m_reporterPrefs.shouldRedirectStdOut = true;
17246         m_reporterPrefs.shouldReportAllAssertions = true;
17247     }
17248
17249     XmlReporter::~XmlReporter() = default;
17250
17251     std::string XmlReporter::getDescription() {
17252         return "Reports test results as an XML document";
17253     }
17254
17255     std::string XmlReporter::getStylesheetRef() const {
17256         return std::string();
17257     }
17258
17259     void XmlReporter::writeSourceInfo( SourceLineInfo const& sourceInfo ) {
17260         m_xml
17261             .writeAttribute( "filename", sourceInfo.file )
17262             .writeAttribute( "line", sourceInfo.line );
17263     }
17264
17265     void XmlReporter::noMatchingTestCases( std::string const& s ) {
17266         StreamingReporterBase::noMatchingTestCases( s );
17267     }
17268
17269     void XmlReporter::testRunStarting( TestRunInfo const& testInfo ) {
17270         StreamingReporterBase::testRunStarting( testInfo );
17271         std::string stylesheetRef = getStylesheetRef();
17272         if ( !stylesheetRef.empty() )
17273             m_xml.writeStylesheetRef( stylesheetRef );
17274         m_xml.startElement( "Catch" );
17275         if ( !m_config->name().empty() )
17276             m_xml.writeAttribute( "name", m_config->name() );
17277         if ( m_config->testSpec().hasFilters() )
17278             m_xml.writeAttribute( "filters", serializeFilters( m_config->getTestsOrTags() ) );
17279         if ( m_config->rngSeed() != 0 )
17280             m_xml.scopedElement( "Randomness" )
17281                 .writeAttribute( "seed", m_config->rngSeed() );
17282     }
17283
17284     void XmlReporter::testGroupStarting( GroupInfo const& groupInfo ) {
17285         StreamingReporterBase::testGroupStarting( groupInfo );
17286         m_xml.startElement( "Group" )
17287             .writeAttribute( "name", groupInfo.name );
17288     }
17289
17290     void XmlReporter::testCaseStarting( TestCaseInfo const& testInfo ) {
17291         StreamingReporterBase::testCaseStarting( testInfo );
17292         m_xml.startElement( "TestCase" )
17293             .writeAttribute( "name", trim( testInfo.name ) )
17294             .writeAttribute( "description", testInfo.description )
17295             .writeAttribute( "tags", testInfo.tagsAsString );
17296
17297         writeSourceInfo( testInfo.lineInfo );
17298
17299         if ( m_config->showDurations() == ShowDurations::Always )
17300             m_testCaseTimer.start();
17301         m_xml.ensureTagClosed();
17302     }
17303
17304     void XmlReporter::sectionStarting( SectionInfo const& sectionInfo ) {
17305         StreamingReporterBase::sectionStarting( sectionInfo );
17306         if ( m_sectionDepth++ > 0 ) {
17307             m_xml.startElement( "Section" )
17308                 .writeAttribute( "name", trim( sectionInfo.name ) );
17309             writeSourceInfo( sectionInfo.lineInfo );
17310             m_xml.ensureTagClosed();
17311         }
17312     }
17313
17314     void XmlReporter::assertionStarting( AssertionInfo const& ) { }
17315
17316     bool XmlReporter::assertionEnded( AssertionStats const& assertionStats ) {
17317         AssertionResult const& result = assertionStats.assertionResult;
17318
17319         bool includeResults = m_config->includeSuccessfulResults() || !result.isOk();
17320
17321

```

```

17321
17322     if( includeResults || result.getResultType() == ResultWas::Warning ) {
17323         // Print any info messages in <Info> tags.
17324         for( auto const& msg : assertionStats.infoMessages ) {
17325             if( msg.type == ResultWas::Info && includeResults ) {
17326                 m_xml.scopedElement( "Info" )
17327                     .writeText( msg.message );
17328             } else if ( msg.type == ResultWas::Warning ) {
17329                 m_xml.scopedElement( "Warning" )
17330                     .writeText( msg.message );
17331             }
17332         }
17333     }
17334
17335     // Drop out if result was successful but we're not printing them.
17336     if( !includeResults && result.getResultType() != ResultWas::Warning )
17337         return true;
17338
17339     // Print the expression if there is one.
17340     if( result.hasExpression() ) {
17341         m_xml.startElement( "Expression" )
17342             .writeAttribute( "success", result.succeeded() )
17343             .writeAttribute( "type", result.getTestMacroName() );
17344
17345         writeSourceInfo( result.getSourceInfo() );
17346
17347         m_xml.scopedElement( "Original" )
17348             .writeText( result.getExpression() );
17349         m_xml.scopedElement( "Expanded" )
17350             .writeText( result.getExpandedExpression() );
17351     }
17352
17353     // And... Print a result applicable to each result type.
17354     switch( result.getResultType() ) {
17355         case ResultWas::ThrowException:
17356             m_xml.startElement( "Exception" );
17357             writeSourceInfo( result.getSourceInfo() );
17358             m_xml.writeText( result.getMessage() );
17359             m_xml.endElement();
17360             break;
17361         case ResultWas::FatalErrorCondition:
17362             m_xml.startElement( "FatalErrorCondition" );
17363             writeSourceInfo( result.getSourceInfo() );
17364             m_xml.writeText( result.getMessage() );
17365             m_xml.endElement();
17366             break;
17367         case ResultWas::Info:
17368             m_xml.scopedElement( "Info" )
17369                 .writeText( result.getMessage() );
17370             break;
17371         case ResultWas::Warning:
17372             // Warning will already have been written
17373             break;
17374         case ResultWas::ExplicitFailure:
17375             m_xml.startElement( "Failure" );
17376             writeSourceInfo( result.getSourceInfo() );
17377             m_xml.writeText( result.getMessage() );
17378             m_xml.endElement();
17379             break;
17380         default:
17381             break;
17382     }
17383
17384     if( result.hasExpression() )
17385         m_xml.endElement();
17386
17387     return true;
17388 }
17389
17390 void XmlReporter::sectionEnded( SectionStats const& sectionStats ) {
17391     StreamingReporterBase::sectionEnded( sectionStats );
17392     if( --m_sectionDepth > 0 ) {
17393         XmlWriter::ScopedElement e = m_xml.scopedElement( "OverallResults" );
17394         e.writeAttribute( "successes", sectionStats.assertions.passed );
17395         e.writeAttribute( "failures", sectionStats.assertions.failed );
17396         e.writeAttribute( "expectedFailures", sectionStats.assertions.failedButOk );
17397
17398         if ( m_config->showDurations() == ShowDurations::Always )
17399             e.writeAttribute( "durationInSeconds", sectionStats.durationInSeconds );
17400
17401         m_xml.endElement();
17402     }
17403 }
17404
17405 void XmlReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
17406     StreamingReporterBase::testCaseEnded( testCaseStats );
17407     XmlWriter::ScopedElement e = m_xml.scopedElement( "OverallResult" );

```

```

17408         e.writeAttribute( "success", testCaseStats.totals.assertions.allOk() );
17409
17410         if ( m_config->showDurations() == ShowDurations::Always )
17411             e.writeAttribute( "durationInSeconds", m_testCaseTimer.getElapsedSeconds() );
17412
17413         if( !testCaseStats.stdOut.empty() )
17414             m_xml.scopedElement( "StdOut" ).writeText( trim( testCaseStats.stdOut ),
XmlFormatting::Newline );
17415         if( !testCaseStats.stdErr.empty() )
17416             m_xml.scopedElement( "StdErr" ).writeText( trim( testCaseStats.stdErr ),
XmlFormatting::Newline );
17417
17418         m_xml.endElement();
17419     }
17420
17421     void XmlReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
17422         StreamingReporterBase::testGroupEnded( testGroupStats );
17423         // TODO: Check testGroupStats.aborting and act accordingly.
17424         m_xml.scopedElement( "OverallResults" )
17425             .writeAttribute( "successes", testGroupStats.totals.assertions.passed )
17426             .writeAttribute( "failures", testGroupStats.totals.assertions.failed )
17427             .writeAttribute( "expectedFailures", testGroupStats.totals.assertions.failedButOk );
17428         m_xml.scopedElement( "OverallResultsCases" )
17429             .writeAttribute( "successes", testGroupStats.totals.testCases.passed )
17430             .writeAttribute( "failures", testGroupStats.totals.testCases.failed )
17431             .writeAttribute( "expectedFailures", testGroupStats.totals.testCases.failedButOk );
17432         m_xml.endElement();
17433     }
17434
17435     void XmlReporter::testRunEnded( TestRunStats const& testRunStats ) {
17436         StreamingReporterBase::testRunEnded( testRunStats );
17437         m_xml.scopedElement( "OverallResults" )
17438             .writeAttribute( "successes", testRunStats.totals.assertions.passed )
17439             .writeAttribute( "failures", testRunStats.totals.assertions.failed )
17440             .writeAttribute( "expectedFailures", testRunStats.totals.assertions.failedButOk );
17441         m_xml.scopedElement( "OverallResultsCases" )
17442             .writeAttribute( "successes", testRunStats.totals.testCases.passed )
17443             .writeAttribute( "failures", testRunStats.totals.testCases.failed )
17444             .writeAttribute( "expectedFailures", testRunStats.totals.testCases.failedButOk );
17445         m_xml.endElement();
17446     }
17447
17448 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
17449     void XmlReporter::benchmarkPreparing(std::string const& name) {
17450         m_xml.startElement("BenchmarkResults")
17451             .writeAttribute("name", name);
17452     }
17453
17454     void XmlReporter::benchmarkStarting(BenchmarkInfo const& info) {
17455         m_xml.writeAttribute("samples", info.samples)
17456             .writeAttribute("resamples", info.resamples)
17457             .writeAttribute("iterations", info.iterations)
17458             .writeAttribute("clockResolution", info.clockResolution)
17459             .writeAttribute("estimatedDuration", info.estimatedDuration)
17460             .writeComment("All values in nano seconds");
17461     }
17462
17463     void XmlReporter::benchmarkEnded(BenchmarkStats<> const& benchmarkStats) {
17464         m_xml.startElement("mean")
17465             .writeAttribute("value", benchmarkStats.mean.point.count())
17466             .writeAttribute("lowerBound", benchmarkStats.mean.lower_bound.count())
17467             .writeAttribute("upperBound", benchmarkStats.mean.upper_bound.count())
17468             .writeAttribute("ci", benchmarkStats.mean.confidence_interval);
17469         m_xml.endElement();
17470         m_xml.startElement("standardDeviation")
17471             .writeAttribute("value", benchmarkStats.standardDeviation.point.count())
17472             .writeAttribute("lowerBound", benchmarkStats.standardDeviation.lower_bound.count())
17473             .writeAttribute("upperBound", benchmarkStats.standardDeviation.upper_bound.count())
17474             .writeAttribute("ci", benchmarkStats.standardDeviation.confidence_interval);
17475         m_xml.endElement();
17476         m_xml.startElement("outliers")
17477             .writeAttribute("variance", benchmarkStats.outlierVariance)
17478             .writeAttribute("lowMild", benchmarkStats.outliers.low_mild)
17479             .writeAttribute("lowSevere", benchmarkStats.outliers.low_severe)
17480             .writeAttribute("highMild", benchmarkStats.outliers.high_mild)
17481             .writeAttribute("highSevere", benchmarkStats.outliers.high_severe);
17482         m_xml.endElement();
17483         m_xml.endElement();
17484     }
17485
17486     void XmlReporter::benchmarkFailed(std::string const& error) {
17487         m_xml.scopedElement("failed")
17488             .writeAttribute("message", error);
17489         m_xml.endElement();
17490     }
17491 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
17492

```

```

17493     CATCH_REGISTER_REPORTER( "xml", XmlReporter )
17494
17495 } // end namespace Catch
17496
17497 #if defined(_MSC_VER)
17498 #pragma warning(pop)
17499 #endif
17500 // end catch_reporter_xml.cpp
17501
17502 namespace Catch {
17503     LeakDetector leakDetector;
17504 }
17505
17506 #ifdef __clang__
17507 #pragma clang diagnostic pop
17508 #endif
17509 // end catch_impl.hpp
17510 #endif
17511
17512 #ifdef CATCH_CONFIG_MAIN
17513 // start catch_default_main.hpp
17514
17515 #ifndef __OBJC__
17516
17517 #if defined(CATCH_CONFIG_WCHAR) && defined(CATCH_PLATFORM_WINDOWS) && defined(_UNICODE) &&
!defined(DO_NOT_USE_WMAIN)
17518 // Standard C/C++ Win32 Unicode wmain entry point
17519 extern "C" int wmain( int argc, wchar_t * argv[], wchar_t * [] ) {
17520     #else
17521 // Standard C/C++ main entry point
17522 int main( int argc, char * argv[] ) {
17523 #endif
17524     #endif
17525
17526     return Catch::Session().run( argc, argv );
17527 }
17528
17529 #else // __OBJC__
17530
17531 // Objective-C entry point
17532 int main( int argc, char * const argv[] ) {
17533     #if !CATCH_ARC_ENABLED
17534         NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
17535     #endif
17536
17537     Catch::registerTestMethods();
17538     int result = Catch::Session().run( argc, (char**)argv );
17539
17540     #if !CATCH_ARC_ENABLED
17541         [pool drain];
17542     #endif
17543
17544     return result;
17545 }
17546 #endif // __OBJC__
17547 // end catch_default_main.hpp
17548 #endif
17549
17550 #if !defined(CATCH_CONFIG_IMPL_ONLY)
17551
17552 #ifdef CLARA_CONFIG_MAIN_NOT_DEFINED
17553     #undef CLARA_CONFIG_MAIN
17554 #endif
17555
17556 #if !defined(CATCH_CONFIG_DISABLE)
17557 // If this config identifier is defined then all CATCH macros are prefixed with CATCH_
17558 #define CATCH_CONFIG_PREFIX_ALL
17559
17560 #define CATCH_REQUIRE( ... ) INTERNAL_CATCH_TEST( "CATCH_REQUIRE", Catch::ResultDisposition::Normal, __VA_ARGS__ )
17561 #define CATCH_REQUIRE_FALSE( ... ) INTERNAL_CATCH_TEST( "CATCH_REQUIRE_FALSE", Catch::ResultDisposition::Normal | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
17562
17563 #define CATCH_REQUIRE_THROWS( ... ) INTERNAL_CATCH_THROWS( "CATCH_REQUIRE_THROWS", Catch::ResultDisposition::Normal, __VA_ARGS__ )
17564 #define CATCH_REQUIRE_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS( "CATCH_REQUIRE_THROWS_AS", exceptionType, Catch::ResultDisposition::Normal, expr )
17565 #define CATCH_REQUIRE_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "CATCH_REQUIRE_THROWS_WITH", Catch::ResultDisposition::Normal, matcher, expr )
17566 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17567 #define CATCH_REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES( "CATCH_REQUIRE_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::Normal, matcher, expr )
17568 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17569 #define CATCH_REQUIRE_NO_THROW( ... ) INTERNAL_CATCH_NO_THROW( "CATCH_REQUIRE_NO_THROW", Catch::ResultDisposition::Normal, __VA_ARGS__ )

```

```
17573
17574 #define CATCH_CHECK( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17575 #define CATCH_CHECK_FALSE( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK_FALSE",
    Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
17576 #define CATCH_CHECKED_IF( ... ) INTERNAL_CATCH_IF( "CATCH_CHECKED_IF",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17577 #define CATCH_CHECKED_ELSE( ... ) INTERNAL_CATCH_ELSE( "CATCH_CHECKED_ELSE",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17578 #define CATCH_CHECK_NOFAIL( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK_NOFAIL",
    Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::SuppressFail, __VA_ARGS__ )
17579
17580 #define CATCH_CHECK_THROWS( ... ) INTERNAL_CATCH_THROWS( "CATCH_CHECK_THROWS",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17581 #define CATCH_CHECK_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS(
    "CATCH_CHECK_THROWS_AS", exceptionType, Catch::ResultDisposition::ContinueOnFailure, expr )
17582 #define CATCH_CHECK_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES(
    "CATCH_CHECK_THROWS_WITH", Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
17583 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17584 #define CATCH_CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
    "CATCH_CHECK_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::ContinueOnFailure, matcher,
    expr )
17585 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17586 #define CATCH_CHECK_NO_THROW( ... ) INTERNAL_CATCH_NO_THROW( "CATCH_CHECK_NO_THROW",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17587
17588 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17589 #define CATCH_CHECK_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CATCH_CHECK_THAT", matcher,
    Catch::ResultDisposition::ContinueOnFailure, arg )
17590
17591 #define CATCH_REQUIRE_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CATCH_REQUIRE_THAT", matcher,
    Catch::ResultDisposition::Normal, arg )
17592 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17593
17594 #define CATCH_INFO( msg ) INTERNAL_CATCH_INFO( "CATCH_INFO", msg )
17595 #define CATCH_UNSCOPED_INFO( msg ) INTERNAL_CATCH_UNSCOPED_INFO( "CATCH_UNSCOPED_INFO", msg )
17596 #define CATCH_WARN( msg ) INTERNAL_CATCH_MSG( "CATCH_WARN", Catch::ResultWas::Warning,
    Catch::ResultDisposition::ContinueOnFailure, msg )
17597 #define CATCH_CAPTURE( ... ) INTERNAL_CATCH_CAPTURE( INTERNAL_CATCH_UNIQUE_NAME(capturer),
    "CATCH_CAPTURE", __VA_ARGS__ )
17598
17599 #define CATCH_TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE( __VA_ARGS__ )
17600 #define CATCH_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className,
    __VA_ARGS__ )
17601 #define CATCH_METHOD_AS_TEST_CASE( method, ... ) INTERNAL_CATCH_METHOD_AS_TEST_CASE( method,
    __VA_ARGS__ )
17602 #define CATCH_REGISTER_TEST_CASE( Function, ... ) INTERNAL_CATCH_REGISTER_TESTCASE( Function,
    __VA_ARGS__ )
17603 #define CATCH_SECTION( ... ) INTERNAL_CATCH_SECTION( __VA_ARGS__ )
17604 #define CATCH_DYNAMIC_SECTION( ... ) INTERNAL_CATCH_DYNAMIC_SECTION( __VA_ARGS__ )
17605 #define CATCH_FAIL( ... ) INTERNAL_CATCH_MSG( "CATCH_FAIL", Catch::ResultWas::ExplicitFailure,
    Catch::ResultDisposition::Normal, __VA_ARGS__ )
17606 #define CATCH_FAIL_CHECK( ... ) INTERNAL_CATCH_MSG( "CATCH_FAIL_CHECK",
    Catch::ResultWas::ExplicitFailure, Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17607 #define CATCH_SUCCEED( ... ) INTERNAL_CATCH_MSG( "CATCH_SUCCEED", Catch::ResultWas::Ok,
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17608
17609 #define CATCH_ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE()
17610
17611 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
17612 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17613 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ )
17614 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD(
    className, __VA_ARGS__ )
17615 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
    INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17616 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__
    )
17617 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(
    __VA_ARGS__ )
17618 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... )
    INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ )
17619 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... )
    INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17620 #else
17621 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
    INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ ) )
17622 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
    INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ ) )
17623 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
    INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
17624 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
    INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
17625 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
    INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ ) )
17626 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
    INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( __VA_ARGS__ ) )
```

```

17627 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS (
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
17628 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS (
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
17629 #endif
17630
17631 #if !defined(CATCH_CONFIG_RUNTIME_STATIC_REQUIRE)
17632 #define CATCH_STATIC_REQUIRE( ... )          static_assert(    __VA_ARGS__ ,      #__VA_ARGS__ );
CATCH_SUCCEED( #__VA_ARGS__ )
17633 #define CATCH_STATIC_REQUIRE_FALSE( ... )    static_assert( !( __VA_ARGS__ ), "!( " #__VA_ARGS__ " )" );
CATCH_SUCCEED( #__VA_ARGS__ )
17634 #else
17635 #define CATCH_STATIC_REQUIRE( ... )          CATCH_REQUIRE(    __VA_ARGS__ )
17636 #define CATCH_STATIC_REQUIRE_FALSE( ... )    CATCH_REQUIRE_FALSE( __VA_ARGS__ )
17637 #endif
17638
17639 // "BDD-style" convenience wrappers
17640 #define CATCH_SCENARIO( ... ) CATCH_TEST_CASE( "Scenario: " __VA_ARGS__ )
17641 #define CATCH_SCENARIO_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, "Scenario:
" __VA_ARGS__ )
17642 #define CATCH_GIVEN( desc )      INTERNAL_CATCH_DYNAMIC_SECTION( "    Given: " « desc )
17643 #define CATCH_AND_GIVEN( desc )  INTERNAL_CATCH_DYNAMIC_SECTION( "And given: " « desc )
17644 #define CATCH_WHEN( desc )       INTERNAL_CATCH_DYNAMIC_SECTION( "    When: " « desc )
17645 #define CATCH_AND_WHEN( desc )   INTERNAL_CATCH_DYNAMIC_SECTION( " And when: " « desc )
17646 #define CATCH_THEN( desc )       INTERNAL_CATCH_DYNAMIC_SECTION( "    Then: " « desc )
17647 #define CATCH_AND_THEN( desc )   INTERNAL_CATCH_DYNAMIC_SECTION( " And: " « desc )
17648
17649 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
17650 #define CATCH_BENCHMARK(...) \
17651     INTERNAL_CATCH_BENCHMARK(INTERNAL_CATCH_UNIQUE_NAME(____C_A_T_C_H____B_E_N_C_H____),
INTERNAL_CATCH_GET_1_ARG(__VA_ARGS__), INTERNAL_CATCH_GET_2_ARG(__VA_ARGS__))
17652 #define CATCH_BENCHMARK_ADVANCED(name) \
17653     INTERNAL_CATCH_BENCHMARK_ADVANCED(INTERNAL_CATCH_UNIQUE_NAME(____C_A_T_C_H____B_E_N_C_H____),
name)
17654 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
17655
17656 // If CATCH_CONFIG_PREFIX_ALL is not defined then the CATCH_ prefix is not required
17657 #else
17658
17659 #define REQUIRE( ... ) INTERNAL_CATCH_TEST( "REQUIRE", Catch::ResultDisposition::Normal, __VA_ARGS__
)
17660 #define REQUIRE_FALSE( ... ) INTERNAL_CATCH_TEST( "REQUIRE_FALSE", Catch::ResultDisposition::Normal |
Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
17661
17662 #define REQUIRE_THROWS( ... ) INTERNAL_CATCH_THROWS( "REQUIRE_THROWS",
Catch::ResultDisposition::Normal, __VA_ARGS__ )
17663 #define REQUIRE_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS( "REQUIRE_THROWS_AS",
exceptionType, Catch::ResultDisposition::Normal, expr )
17664 #define REQUIRE_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "REQUIRE_THROWS_WITH",
Catch::ResultDisposition::Normal, matcher, expr )
17665 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17666 #define REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
"REQUIRE_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::Normal, matcher, expr )
17667 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17668 #define REQUIRE_NO_THROW( ... ) INTERNAL_CATCH_NO_THROW( "REQUIRE_NO_THROW",
Catch::ResultDisposition::Normal, __VA_ARGS__ )
17669
17670 #define CHECK( ... ) INTERNAL_CATCH_TEST( "CHECK", Catch::ResultDisposition::ContinueOnFailure,
__VA_ARGS__ )
17671 #define CHECK_FALSE( ... ) INTERNAL_CATCH_TEST( "CHECK_FALSE",
Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
17672 #define CHECKED_IF( ... ) INTERNAL_CATCH_IF( "CHECKED_IF",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17673 #define CHECKED_ELSE( ... ) INTERNAL_CATCH_ELSE( "CHECKED_ELSE",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17674 #define CHECK_NO_FAIL( ... ) INTERNAL_CATCH_TEST( "CHECK_NO_FAIL",
Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::SuppressFail, __VA_ARGS__ )
17675
17676 #define CHECK_THROWS( ... ) INTERNAL_CATCH_THROWS( "CHECK_THROWS",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17677 #define CHECK_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS( "CHECK_THROWS_AS",
exceptionType, Catch::ResultDisposition::ContinueOnFailure, expr )
17678 #define CHECK_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "CHECK_THROWS_WITH",
Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
17679 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17680 #define CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
"CHECK_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
17681 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17682 #define CHECK_NO_THROW( ... ) INTERNAL_CATCH_NO_THROW( "CHECK_NO_THROW",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17683
17684 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17685 #define CHECK_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CHECK_THAT", matcher,
Catch::ResultDisposition::ContinueOnFailure, arg )
17686
17687 #define REQUIRE_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "REQUIRE_THAT", matcher,
Catch::ResultDisposition::Normal, arg )

```



```

17688 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17689
17690 #define INFO( msg ) INTERNAL_CATCH_INFO( "INFO", msg )
17691 #define UNSCOPED_INFO( msg ) INTERNAL_CATCH_UNSCOPED_INFO( "UNSCOPED_INFO", msg )
17692 #define WARN( msg ) INTERNAL_CATCH_MSG( "WARN", Catch::ResultWas::Warning,
17693   Catch::ResultDisposition::ContinueOnFailure, msg )
17694 #define CAPTURE( ... ) INTERNAL_CATCH_CAPTURE( INTERNAL_CATCH_UNIQUE_NAME(capturer),
17695   "CAPTURE",__VA_ARGS__ )
17696
17697 #define TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE( __VA_ARGS__ )
17698 #define TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, __VA_ARGS__ )
17699 #define METHOD_AS_TEST_CASE( method, ... ) INTERNAL_CATCH_TEST_CASE_METHOD_AS_TEST_CASE( method, __VA_ARGS__ )
17700 #define REGISTER_TEST_CASE( Function, ... ) INTERNAL_CATCH_REGISTER_TESTCASE( Function, __VA_ARGS__ )
17701 #define SECTION( ... ) INTERNAL_CATCH_SECTION( __VA_ARGS__ )
17702 #define DYNAMIC_SECTION( ... ) INTERNAL_CATCH_DYNAMIC_SECTION( __VA_ARGS__ )
17703 #define FAIL( ... ) INTERNAL_CATCH_MSG( "FAIL", Catch::ResultWas::ExplicitFailure,
17704   Catch::ResultDisposition::Normal, __VA_ARGS__ )
17705 #define FAIL_CHECK( ... ) INTERNAL_CATCH_MSG( "FAIL_CHECK", Catch::ResultWas::ExplicitFailure,
17706   Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17707 #define SUCCEED( ... ) INTERNAL_CATCH_MSG( "SUCCEED", Catch::ResultWas::Ok,
17708   Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17709 #define ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE()
17710
17711 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
17712 #define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17713 #define TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ )
17714 #define TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD(
17715   className, __VA_ARGS__ )
17716 #define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG(
17717   className, __VA_ARGS__ )
17718 #define TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ )
17719 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(
17720   __VA_ARGS__ )
17721 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... )
17722   INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ )
17723 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... )
17724   INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17725 #define TEMPLATE_LIST_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE( __VA_ARGS__ )
17726 #define TEMPLATE_LIST_TEST_CASE_METHOD( className, ... )
17727   INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD( className, __VA_ARGS__ )
17728 #define TEMPLATE_LIST_TEST_CASE_METHOD_SIG( className, ... )
17729   INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17730 #define TEMPLATE_LIST_TEST_CASE_METHOD_SIG( className, ... )
17731   INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17732 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(
17733   __VA_ARGS__ )
17734 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD(
17735   className, __VA_ARGS__ )
17736 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG(
17737   className, __VA_ARGS__ )
17738 #define TEMPLATE_LIST_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE( __VA_ARGS__ )
17739 #define TEMPLATE_LIST_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD(
17740   className, __VA_ARGS__ )
17741 #define TEMPLATE_LIST_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_SIG(
17742   className, __VA_ARGS__ )
17743 #define TEMPLATE_LIST_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_SIG(
17744   className, __VA_ARGS__ )
17745 #endif
17746
17747 #if !defined(CATCH_CONFIG_RUNTIME_STATIC_REQUIRE)
17748 #define STATIC_REQUIRE( ... ) static_assert( __VA_ARGS__, #__VA_ARGS__ ); SUCCEED(
17749   #__VA_ARGS__ )
17750 #define STATIC_REQUIRE_FALSE( ... ) static_assert( !( __VA_ARGS__ ), "!( " #__VA_ARGS__ " )" ); SUCCEED(
17751   "!( " #__VA_ARGS__ " )" )
17752 #else
17753 #define STATIC_REQUIRE( ... ) REQUIRE( __VA_ARGS__ )
17754 #define STATIC_REQUIRE_FALSE( ... ) REQUIRE_FALSE( __VA_ARGS__ )
17755 #endif
17756
17757 #endif
17758
17759 #define CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION( signature )
17760
17761 // "BDD-style" convenience wrappers
17762 #define SCENARIO( ... ) TEST_CASE( "Scenario: " __VA_ARGS__ )
17763 #define SCENARIO_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, "Scenario: "
17764   __VA_ARGS__ )
17765
17766 #define GIVEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( "    Given: " « desc )
17767 #define AND_GIVEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( "And given: " « desc )
17768 #define WHEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( "    When: " « desc )
17769 #define AND_WHEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( "And when: " « desc )
17770 #define THEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( "    Then: " « desc )

```



```

17751 #define AND_THEN( desc )    INTERNAL_CATCH_DYNAMIC_SECTION( "      And: " « desc )
17752
17753 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
17754 #define BENCHMARK(...) \
17755     INTERNAL_CATCH_BENCHMARK(INTERNAL_CATCH_UNIQUE_NAME(____C_A_T_C_H____B_E_N_C_H____),
17756     INTERNAL_CATCH_GET_1_ARG(__VA_ARGS__,), INTERNAL_CATCH_GET_2_ARG(__VA_ARGS__,))
17757 #define BENCHMARK_ADVANCED(name) \
17758     INTERNAL_CATCH_BENCHMARK_ADVANCED(INTERNAL_CATCH_UNIQUE_NAME(____C_A_T_C_H____B_E_N_C_H____),
17759     name)
17760 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
17761
17762 using Catch::Detail::Approx;
17763
17764 #else // CATCH_CONFIG_DISABLE
17765
17766 // If this config identifier is defined then all CATCH macros are prefixed with CATCH_
17767 #ifndef CATCH_CONFIG_PREFIX_ALL
17768 #define CATCH_REQUIRE( ... )          (void)(0)
17769 #define CATCH_REQUIRE_FALSE( ... )    (void)(0)
17770
17771 #define CATCH_REQUIRE_THROWS( ... )   (void)(0)
17772 #define CATCH_REQUIRE_THROWS_AS( expr, exceptionType ) (void)(0)
17773 #define CATCH_REQUIRE_THROWS_WITH( expr, matcher )      (void)(0)
17774 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17775 #define CATCH_REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
17776 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17777 #define CATCH_REQUIRE_NO_THROW( ... ) (void)(0)
17778
17779 #define CATCH_CHECK( ... )             (void)(0)
17780 #define CATCH_CHECK_FALSE( ... )       (void)(0)
17781 #define CATCH_CHECKED_IF( ... )        if (__VA_ARGS__)
17782 #define CATCH_CHECKED_ELSE( ... )      if (!(__VA_ARGS__))
17783 #define CATCH_CHECK_NOFAIL( ... )      (void)(0)
17784
17785 #define CATCH_CHECK_THROWS( ... )       (void)(0)
17786 #define CATCH_CHECK_THROWS_AS( expr, exceptionType ) (void)(0)
17787 #define CATCH_CHECK_THROWS_WITH( expr, matcher )      (void)(0)
17788 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17789 #define CATCH_CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
17790 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17791 #define CATCH_CHECK_NO_THROW( ... )      (void)(0)
17792
17793 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17794 #define CATCH_CHECK_THAT( arg, matcher )    (void)(0)
17795
17796 #define CATCH_REQUIRE_THAT( arg, matcher )   (void)(0)
17797 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17798
17799 #define CATCH_INFO( msg )                  (void)(0)
17800 #define CATCH_UNSCOPED_INFO( msg )         (void)(0)
17801 #define CATCH_WARN( msg )                  (void)(0)
17802 #define CATCH_CAPTURE( msg )               (void)(0)
17803
17804 #define CATCH_TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
17805     ____C_A_T_C_H____T_E_S_T____ ))
17806 #define CATCH_TEST_CASE_METHOD( className, ... )
17807     INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( ____C_A_T_C_H____T_E_S_T____ ))
17808 #define CATCH_METHOD_AS_TEST_CASE( method, ... )
17809     CATCH_REGISTER_TEST_CASE( Function, ... ) (void)(0)
17810 #define CATCH_SECTION( ... )
17811 #define CATCH_DYNAMIC_SECTION( ... )
17812 #define CATCH_FAIL( ... ) (void)(0)
17813 #define CATCH_FAIL_CHECK( ... ) (void)(0)
17814 #define CATCH_SUCCEED( ... ) (void)(0)
17815
17816 #define CATCH_ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
17817     ____C_A_T_C_H____T_E_S_T____ ))
17818
17819 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
17820 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__)
17821 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... )
17822     INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__)
17823 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... )
17824     INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__)
17825 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17826 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17827 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
17828     className, __VA_ARGS__ )
17829 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
17830     className, __VA_ARGS__ )
17831 #else
17832 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
17833     INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__) )
17834 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(

```

```

INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__ )
17828 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS (
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__ ) )
17829 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS (
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ ) )
17830 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17831 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17832 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
className, __VA_ARGS__ )
17833 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
className, __VA_ARGS__ )
17834 #endif
17835
17836 // "BDD-style" convenience wrappers
17837 #define CATCH_SCENARIO( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
__C_A_T_C_H____T_E_S_T____ ))
17838 #define CATCH_SCENARIO_METHOD( className, ... )
INTERNAL_CATCH_TESTCASE_METHOD_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
__C_A_T_C_H____T_E_S_T____ ), className )
17839 #define CATCH_GIVEN( desc )
17840 #define CATCH_AND_GIVEN( desc )
17841 #define CATCH_WHEN( desc )
17842 #define CATCH_AND_WHEN( desc )
17843 #define CATCH_THEN( desc )
17844 #define CATCH_AND_THEN( desc )
17845
17846 #define CATCH_STATIC_REQUIRE( ... ) (void)(0)
17847 #define CATCH_STATIC_REQUIRE_FALSE( ... ) (void)(0)
17848
17849 // If CATCH_CONFIG_PREFIX_ALL is not defined then the CATCH_ prefix is not required
17850 #else
17851
17852 #define REQUIRE( ... ) (void)(0)
17853 #define REQUIRE_FALSE( ... ) (void)(0)
17854
17855 #define REQUIRE_THROWS( ... ) (void)(0)
17856 #define REQUIRE_THROWS_AS( expr, exceptionType ) (void)(0)
17857 #define REQUIRE_THROWS_WITH( expr, matcher ) (void)(0)
17858 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17859 #define REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
17860 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17861 #define REQUIRE_NOTHROW( ... ) (void)(0)
17862
17863 #define CHECK( ... ) (void)(0)
17864 #define CHECK_FALSE( ... ) (void)(0)
17865 #define CHECKED_IF( ... ) if ( __VA_ARGS__ )
17866 #define CHECKED_ELSE( ... ) if ( !(__VA_ARGS__) )
17867 #define CHECK_NOFAIL( ... ) (void)(0)
17868
17869 #define CHECK_THROWS( ... ) (void)(0)
17870 #define CHECK_THROWS_AS( expr, exceptionType ) (void)(0)
17871 #define CHECK_THROWS_WITH( expr, matcher ) (void)(0)
17872 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17873 #define CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
17874 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17875 #define CHECK_NOTHROW( ... ) (void)(0)
17876
17877 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17878 #define CHECK_THAT( arg, matcher ) (void)(0)
17879
17880 #define REQUIRE_THAT( arg, matcher ) (void)(0)
17881 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17882
17883 #define INFO( msg ) (void)(0)
17884 #define UNSCOPED_INFO( msg ) (void)(0)
17885 #define WARN( msg ) (void)(0)
17886 #define CAPTURE( msg ) (void)(0)
17887
17888 #define TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
__C_A_T_C_H____T_E_S_T____ ))
17889 #define TEST_CASE_METHOD( className, ... )
INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( __C_A_T_C_H____T_E_S_T____ ))
17890 #define METHOD_AS_TEST_CASE( method, ... )
17891 #define REGISTER_TEST_CASE( Function, ... ) (void)(0)
17892 #define SECTION( ... )
17893 #define DYNAMIC_SECTION( ... )
17894 #define FAIL( ... ) (void)(0)
17895 #define FAIL_CHECK( ... ) (void)(0)
17896 #define SUCCEED( ... ) (void)(0)
17897 #define ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
__C_A_T_C_H____T_E_S_T____ ))
17898
17899 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
17900 #define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION( __VA_ARGS__ )
17901 #define TEMPLATE_TEST_CASE_SIG( ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION( __VA_ARGS__ )
17902 #define TEMPLATE_TEST_CASE_METHOD( className, ... )

```

```

INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__)
17903 #define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ )
17904 #define TEMPLATE_PRODUCT_TEST_CASE( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
17905 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
17906 #define TEMPLATE_TEMPLATE_TEST_CASE_METHOD( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
__VA_ARGS__ )
17907 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
__VA_ARGS__ )
17908 #else
17909 #define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__ ) )
17910 #define TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__ ) )
17911 #define TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__ ) )
17912 #define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ ) )
17913 #define TEMPLATE_PRODUCT_TEST_CASE( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
17914 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
17915 #define TEMPLATE_TEMPLATE_TEST_CASE_METHOD( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
__VA_ARGS__ )
17916 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
__VA_ARGS__ )
17917 #endif
17918
17919 #define STATIC_REQUIRE( ... ) (void)(0)
17920 #define STATIC_REQUIRE_FALSE( ... ) (void)(0)
17921
17922 #endif
17923
17924 #define CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION_NO_REG(
INTERNAL_CATCH_UNIQUE_NAME( catch_internal_ExceptionTranslator ), signature )
17925
17926 // "BDD-style" convenience wrappers
17927 #define SCENARIO( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
__C_A_T_C_H____T_E_S_T____ ) )
17928 #define SCENARIO_METHOD( className, ... )
INTERNAL_CATCH_TESTCASE_METHOD_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
__C_A_T_C_H____T_E_S_T____ ), className )
17929
17930 #define GIVEN( desc )
17931 #define AND_GIVEN( desc )
17932 #define WHEN( desc )
17933 #define AND_WHEN( desc )
17934 #define THEN( desc )
17935 #define AND_THEN( desc )
17936
17937 using Catch::Detail::Approx;
17938
17939 #endif
17940
17941 #endif // ! CATCH_CONFIG_IMPL_ONLY
17942
17943 // start catch_reenable_warnings.h
17944
17945
17946 #ifdef __clang__
17947 #   ifdef __ICC // icpc defines the __clang__ macro
17948 #       pragma warning(pop)
17949 #   else
17950 #       pragma clang diagnostic pop
17951 #   endif
17952 #elif defined __GNUC__
17953 #   pragma GCC diagnostic pop
17954 #endif
17955
17956 // end catch_reenable_warnings.h
17957 // end catch.hpp
17958 #endif // TWOBLUECUBES_SINGLE_INCLUDE_CATCH_HPP_INCLUDED
17959

```

6.2 /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/cmdline.h

```

00001 #ifndef CMDLINE_H
00002 #define CMDLINE_H
00003
00004 void use_arguments(int argc, char* argv[]);
00005
00006 #endif //UNTITLED_CMDLINE_H

```

6.3 /Users/tailangcao/myGithubRepo/CS6015/Assignment/Assignment 4/Expr.h

```

00001 #ifndef Expr_Expr_H
00002 #define Expr_Expr_H
00003
00004 #include <string>
00005 #include <sstream>
00006 #include "catch.hpp"
00007 #include "iostream"
00008
00009
00010 typedef enum {
00011     prec_none,    // = 0
00012     prec_add,     // = 1
00013     prec_mult     // = 2
00014 } precedence_t;
00015
00016
00017 class Expr {
00018 public:
00019     virtual bool equals(Expr *e) = 0;
00020     virtual int interp() = 0;
00021     virtual bool has_variable() = 0;
00022     virtual Expr *subst(std::string, Expr *s) = 0;
00023     virtual void print(std::ostream &ot) = 0;
00024     virtual void pretty_print(std::ostream &ot, precedence_t prec) = 0;
00025     std::string to_string() {
00026         std::stringstream st("");
00027         this->print(st);
00028         return st.str();
00029     }
00030     virtual void pretty_print_dr(std::ostream &ot) = 0;
00031     std::string to_pretty_string() {
00032         std::stringstream st("");
00033         this->pretty_print(st, prec_none);
00034         return st.str();
00035     }
00036 };
00037
00038
00039 class Var:public Expr{
00040 public:
00041     std::string val;
00042     Var(std::string val);
00043     bool equals(Expr *e) ;
00044     int interp () ;
00045     bool has_variable();
00046     Expr *subst(std::string, Expr *s) ;
00047     void print(std::ostream &ot) ;
00048     void pretty_print_dr(std::ostream &ot);
00049     void pretty_print(std::ostream &ot, precedence_t prec);
00050 };
00051
00052 class Num:public Expr{
00053 public:
00054     int val;
00055     Num(int val);
00056     bool equals(Expr *e);
00057     int interp ();
00058     bool has_variable();
00059     Expr *subst(std::string, Expr *s);
00060     void print(std::ostream &ot) ;
00061     void pretty_print_dr(std::ostream &ot);
00062     void pretty_print(std::ostream &ot, precedence_t prec);
00063 };
00064
00065 class Add:public Expr{
00066 public:
00067     Expr *lhs;
00068     Expr *rhs;
00069     Add(Expr *lhs, Expr *rhs);
00070     bool equals(Expr *e);
00071     int interp () ;
00072     bool has_variable() ;
00073     Expr *subst(std::string, Expr *s);
00074     void print(std::ostream &ot);
00075     void pretty_print_dr(std::ostream &ot);
00076     void pretty_print(std::ostream &ot, precedence_t prec);
00077 };
00078
00079 class Mult:public Expr{
00080 public:
00081     Expr *lhs;
00082     Expr *rhs;

```

```
00083     Mult(Expr *lhs, Expr *rhs);
00084     bool equals(Expr *e) ;
00085     int interp () ;
00086     bool has_variable() ;
00087     Expr *subst(std::string,Expr *s) ;
00088     void print(std::ostream &ot) ;
00089     void pretty_print_dr(std::ostream &ot);
00090     void pretty_print(std::ostream &ot, precedence_t prec);
00091 };
00092
00093
00094 #endif //EXPR_EXPR_H
```


Index

- Add, [13](#)
 - Add, [13](#)
 - equals, [14](#)
 - has_variable, [14](#)
 - interp, [14](#)
 - pretty_print, [14](#)
 - pretty_print_dr, [14](#)
 - print, [14](#)
 - subst, [14](#)
- Catch::always_false< T >, [15](#)
- Catch::AssertionHandler, [18](#)
- Catch::AssertionInfo, [18](#)
- Catch::AssertionReaction, [18](#)
- Catch::AutoReg, [19](#)
- Catch::BinaryExpr< LhsT, RhsT >, [19](#)
- Catch::Captor, [20](#)
- Catch::CaseSensitive, [21](#)
- Catch::Counts, [26](#)
- Catch::Composer, [26](#)
- Catch::Detail::Approx, [15](#)
- Catch::Detail::EnumInfo, [28](#)
- Catch::detail::is_range_impl< T, typename >, [43](#)
- Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >()))>::type >, [44](#)
- Catch::Detail::IsStreamInsertable< T >, [44](#)
- Catch::detail::void_type<... >, [94](#)
- Catch::ExceptionTranslatorRegistrar, [31](#)
- Catch::ExprLhs< LhsT >, [32](#)
- Catch::GeneratorException, [35](#)
- Catch::Generators::as< T >, [17](#)
- Catch::Generators::ChunkGenerator< T >, [21](#)
 - get, [22](#)
 - next, [22](#)
- Catch::Generators::FilterGenerator< T, Predicate >, [33](#)
 - get, [33](#)
 - next, [33](#)
- Catch::Generators::FixedValuesGenerator< T >, [34](#)
 - get, [34](#)
 - next, [34](#)
- Catch::Generators::Generators< T >, [35](#)
 - get, [36](#)
 - next, [36](#)
- Catch::Generators::GeneratorUntypedBase, [36](#)
- Catch::Generators::GeneratorWrapper< T >, [37](#)
- Catch::Generators::IGenerator< T >, [39](#)
- Catch::Generators::IteratorGenerator< T >, [44](#)
 - get, [45](#)
 - next, [45](#)
- Catch::Generators::MapGenerator< T, U, Func >, [47](#)
 - get, [48](#)
 - next, [48](#)
- Catch::Generators::RandomFloatingGenerator< Float >, [64](#)
 - get, [65](#)
 - next, [65](#)
- Catch::Generators::RandomIntegerGenerator< Integer >, [65](#)
 - get, [66](#)
 - next, [66](#)
- Catch::Generators::RangeGenerator< T >, [66](#)
 - get, [67](#)
 - next, [67](#)
- Catch::Generators::RepeatGenerator< T >, [68](#)
 - get, [69](#)
 - next, [69](#)
- Catch::Generators::SingleValueGenerator< T >, [73](#)
 - get, [73](#)
 - next, [73](#)
- Catch::Generators::TakeGenerator< T >, [85](#)
 - get, [85](#)
 - next, [85](#)
- Catch::IConfig, [37](#)
- Catch::IContext, [38](#)
- Catch::IExceptionTranslator, [38](#)
- Catch::IExceptionTranslatorRegistry, [38](#)
- Catch::IGeneratorTracker, [39](#)
- Catch::IMutableContext, [40](#)
- Catch::IMutableEnumValuesRegistry, [40](#)
- Catch::IMutableRegistryHub, [41](#)
- Catch::IRegistryHub, [41](#)
- Catch::IResultCapture, [41](#)
- Catch::IRunner, [42](#)
- Catch::is_callable< Fun(Args...)>, [42](#)
- Catch::is_callable< T >, [42](#)
- Catch::is_callable_tester, [43](#)
- Catch::is_range< T >, [43](#)
- Catch::IStream, [44](#)
- Catch::ITestCaseRegistry, [45](#)
- Catch::ITestInvoker, [46](#)
- Catch::ITransientExpression, [46](#)
- Catch::LazyExpression, [47](#)
- Catch::Matchers::Exception::ExceptionMessageMatcher, [30](#)
 - describe, [31](#)
- Catch::Matchers::Floating::WithinAbsMatcher, [95](#)
 - describe, [95](#)
- Catch::Matchers::Floating::WithinRelMatcher, [96](#)

- describe, 96
- Catch::Matchers::Floating::WithinUlpMatcher, 97
 - describe, 97
- Catch::Matchers::Generic::PredicateMatcher< T >, 63
 - describe, 64
 - match, 64
- Catch::Matchers::Impl::MatchAllOf< ArgT >, 48
 - describe, 49
- Catch::Matchers::Impl::MatchAnyOf< ArgT >, 49
 - describe, 50
- Catch::Matchers::Impl::MatcherBase< T >, 51
- Catch::Matchers::Impl::MatcherMethod< ObjectT >, 52
- Catch::Matchers::Impl::MatcherUntypedBase, 52
- Catch::Matchers::Impl::MatchNotOf< ArgT >, 54
 - describe, 55
- Catch::Matchers::StdString::CasedString, 20
- Catch::Matchers::StdString::ContainsMatcher, 23
- Catch::Matchers::StdString::EndsWithMatcher, 27
- Catch::Matchers::StdString::EqualsMatcher, 28
- Catch::Matchers::StdString::RegexMatcher, 67
 - describe, 68
- Catch::Matchers::StdString::StartsWithMatcher, 74
- Catch::Matchers::StdString::StringMatcherBase, 83
 - describe, 84
- Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >, 16
 - describe, 17
- Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >, 22
 - describe, 23
- Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >, 25
 - describe, 26
- Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >, 29
 - describe, 30
- Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >, 90
 - describe, 91
- Catch::MatchExpr< ArgT, MatcherT >, 53
 - streamReconstructedExpression, 53
- Catch::MessageBuilder, 55
- Catch::MessageInfo, 56
- Catch::MessageStream, 56
- Catch::NameAndTags, 59
- Catch::NonCopyable, 59
- Catch::Option< T >, 62
- Catch::pluralise, 62
- Catch::RegistrarForTagAliases, 68
- Catch::ResultDisposition, 69
- Catch::ResultWas, 70
- Catch::ReusableStringStream, 70
- Catch::RunTests, 70
- Catch::ScopedMessage, 71
- Catch::Section, 71
- Catch::SectionEndInfo, 71
- Catch::SectionInfo, 72
- Catch::ShowDurations, 72
- Catch::SimplePcg32, 72
- Catch::SourceLineInfo, 74
- Catch::StreamEndStop, 76
- Catch::StringMaker< bool >, 76
- Catch::StringMaker< Catch::Detail::Approx >, 76
- Catch::StringMaker< char >, 77
- Catch::StringMaker< char * >, 77
- Catch::StringMaker< char const * >, 77
- Catch::StringMaker< char[SZ]>, 77
- Catch::StringMaker< double >, 78
- Catch::StringMaker< float >, 78
- Catch::StringMaker< int >, 78
- Catch::StringMaker< long >, 78
- Catch::StringMaker< long long >, 79
- Catch::StringMaker< R C::* >, 79
- Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStreamInsertable< R >::value >::type >, 79
- Catch::StringMaker< signed char >, 79
- Catch::StringMaker< signed char[SZ]>, 80
- Catch::StringMaker< std::nullptr_t >, 80
- Catch::StringMaker< std::string >, 80
- Catch::StringMaker< std::wstring >, 80
- Catch::StringMaker< T * >, 81
- Catch::StringMaker< T, typename >, 76
- Catch::StringMaker< T[SZ]>, 81
- Catch::StringMaker< unsigned char >, 81
- Catch::StringMaker< unsigned char[SZ]>, 81
- Catch::StringMaker< unsigned int >, 82
- Catch::StringMaker< unsigned long >, 82
- Catch::StringMaker< unsigned long long >, 82
- Catch::StringMaker< wchar_t * >, 82
- Catch::StringMaker< wchar_t const * >, 83
- Catch::StringRef, 84
- Catch::TestCase, 86
- Catch::TestCaseInfo, 87
- Catch::TestFailureException, 88
- Catch::TestInvokerAsMethod< C >, 88
 - invoke, 88
- Catch::Timer, 89
- Catch::Totals, 89
- Catch::true_given< typename >, 89
- Catch::UnaryExpr< LhsT >, 90
- Catch::UseColour, 91
- Catch::WaitForKeypress, 94
- Catch::WarnAbout, 94
- Catch_global_namespace_dummy, 21
- describe
 - Catch::Matchers::Exception::ExceptionMessageMatcher, 31
 - Catch::Matchers::Floating::WithinAbsMatcher, 95
 - Catch::Matchers::Floating::WithinRelMatcher, 96
 - Catch::Matchers::Floating::WithinUlpMatcher, 97
 - Catch::Matchers::Generic::PredicateMatcher< T >, 64
 - Catch::Matchers::Impl::MatchAllOf< ArgT >, 49
 - Catch::Matchers::Impl::MatchAnyOf< ArgT >, 50
 - Catch::Matchers::Impl::MatchNotOf< ArgT >, 55

- Catch::Matchers::StdString::RegexMatcher, 68
- Catch::Matchers::StdString::StringMatcherBase, 84
- Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >, 17
- Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >, 23
- Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >, 26
- Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >, 30
- Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >, 91
- expr
 - Add, 14
 - Mult, 58
 - Num, 61
 - Var, 93
- Expr, 32
- get
 - Catch::Generators::ChunkGenerator< T >, 22
 - Catch::Generators::FilterGenerator< T, Predicate >, 33
 - Catch::Generators::FixedValuesGenerator< T >, 34
 - Catch::Generators::Generators< T >, 36
 - Catch::Generators::IteratorGenerator< T >, 45
 - Catch::Generators::MapGenerator< T, U, Func >, 48
 - Catch::Generators::RandomFloatingGenerator< Float >, 65
 - Catch::Generators::RandomIntegerGenerator< Integer >, 66
 - Catch::Generators::RangeGenerator< T >, 67
 - Catch::Generators::RepeatGenerator< T >, 69
 - Catch::Generators::SingleValueGenerator< T >, 73
 - Catch::Generators::TakeGenerator< T >, 85
- has_variable
 - Add, 14
 - Mult, 58
 - Num, 61
 - Var, 93
- interp
 - Add, 14
 - Mult, 58
 - Num, 61
 - Var, 93
- invoke
 - Catch::TestInvokerAsMethod< C >, 88
- match
 - Catch::Matchers::Generic::PredicateMatcher< T >, 64
- MSDScript, 1
- Mult, 57
 - equals, 58
 - has_variable, 58
 - interp, 58
 - Mult, 57
 - pretty_print, 58
 - pretty_print_dr, 58
 - print, 58
 - subst, 58
- next
 - Catch::Generators::ChunkGenerator< T >, 22
 - Catch::Generators::FilterGenerator< T, Predicate >, 33
 - Catch::Generators::FixedValuesGenerator< T >, 34
 - Catch::Generators::Generators< T >, 36
 - Catch::Generators::IteratorGenerator< T >, 45
 - Catch::Generators::MapGenerator< T, U, Func >, 48
 - Catch::Generators::RandomFloatingGenerator< Float >, 65
 - Catch::Generators::RandomIntegerGenerator< Integer >, 66
 - Catch::Generators::RangeGenerator< T >, 67
 - Catch::Generators::RepeatGenerator< T >, 69
 - Catch::Generators::SingleValueGenerator< T >, 73
 - Catch::Generators::TakeGenerator< T >, 85
- Num, 60
 - equals, 61
 - has_variable, 61
 - interp, 61
 - Num, 60
 - pretty_print, 61
 - pretty_print_dr, 61
 - print, 61
 - subst, 61
- pretty_print
 - Add, 14
 - Mult, 58
 - Num, 61
 - Var, 93
- pretty_print_dr
 - Add, 14
 - Mult, 58
 - Num, 61
 - Var, 93
- print
 - Add, 14
 - Mult, 58
 - Num, 61
 - Var, 93
- streamReconstructedExpression
 - Catch::MatchExpr< ArgT, MatcherT >, 53
- subst
 - Add, 14

Mult, [58](#)

Num, [61](#)

Var, [93](#)

Var, [92](#)

 equals, [93](#)

 has_variable, [93](#)

 interp, [93](#)

 pretty_print, [93](#)

 pretty_print_dr, [93](#)

 print, [93](#)

 subst, [93](#)

 Var, [92](#)