

My Project

Generated by Doxygen 1.10.0

1 Namespace Index	1
1.1 Namespace List	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 Namespace Documentation	13
5.1 Catch Namespace Reference	13
5.1.1 Typedef Documentation	17
5.1.1.1 exceptionTranslateFunction	17
5.1.1.2 ExceptionTranslators	17
5.1.1.3 FunctionReturnType	17
5.1.1.4 IConfigPtr	17
5.1.1.5 IReporterFactoryPtr	18
5.1.1.6 StringMatcher	18
5.1.2 Enumeration Type Documentation	18
5.1.2.1 Verbosity	18
5.1.3 Function Documentation	18
5.1.3.1 cerr()	18
5.1.3.2 cleanUp()	18
5.1.3.3 cleanUpContext()	18
5.1.3.4 clog()	18
5.1.3.5 compareEqual() [1/5]	19
5.1.3.6 compareEqual() [2/5]	19
5.1.3.7 compareEqual() [3/5]	19
5.1.3.8 compareEqual() [4/5]	19
5.1.3.9 compareEqual() [5/5]	19
5.1.3.10 compareNotEqual() [1/5]	19
5.1.3.11 compareNotEqual() [2/5]	20
5.1.3.12 compareNotEqual() [3/5]	20
5.1.3.13 compareNotEqual() [4/5]	20
5.1.3.14 compareNotEqual() [5/5]	20
5.1.3.15 contains()	20
5.1.3.16 cout()	20
5.1.3.17 endsWith() [1/2]	21
5.1.3.18 endsWith() [2/2]	21
5.1.3.19 filterTests()	21
5.1.3.20 formatReconstructedExpression()	21

5.1.3.21 getAllTestCasesSorted()	21
5.1.3.22 getCurrentContext()	21
5.1.3.23 getCurrentMutableContext()	21
5.1.3.24 getCurrentNanosecondsSinceEpoch()	22
5.1.3.25 getEstimatedClockResolution()	22
5.1.3.26 getMutableRegistryHub()	22
5.1.3.27 getRegistryHub()	22
5.1.3.28 getResultCapture()	22
5.1.3.29 handleExceptionMatchExpr() [1/2]	22
5.1.3.30 handleExceptionMatchExpr() [2/2]	22
5.1.3.31 handleExpression() [1/2]	22
5.1.3.32 handleExpression() [2/2]	23
5.1.3.33 isFalseTest()	23
5.1.3.34 isJustInfo()	23
5.1.3.35 isOk()	23
5.1.3.36 isThrowSafe()	23
5.1.3.37 makeMatchExpr()	23
5.1.3.38 makeStream()	23
5.1.3.39 makeTestCase()	24
5.1.3.40 makeTestInvoker() [1/2]	24
5.1.3.41 makeTestInvoker() [2/2]	24
5.1.3.42 matchTest()	24
5.1.3.43 operator""_sr()	24
5.1.3.44 operator+()	24
5.1.3.45 operator+=()	25
5.1.3.46 operator<<() [1/2]	25
5.1.3.47 operator<<() [2/2]	25
5.1.3.48 operator" ()	25
5.1.3.49 rangeToString() [1/2]	25
5.1.3.50 rangeToString() [2/2]	25
5.1.3.51 replaceInPlace()	25
5.1.3.52 rng()	26
5.1.3.53 rngSeed()	26
5.1.3.54 shouldContinueOnFailure()	26
5.1.3.55 shouldSuppressFailure()	26
5.1.3.56 splitStringRef()	26
5.1.3.57 startsWith() [1/2]	26
5.1.3.58 startsWith() [2/2]	26
5.1.3.59 throw_domain_error()	26
5.1.3.60 throw_exception()	26
5.1.3.61 throw_logic_error()	27
5.1.3.62 throw_runtime_error()	27

5.1.3.63 toLower()	27
5.1.3.64 toLowerInPlace()	27
5.1.3.65 translateActiveException()	27
5.1.3.66 trim() [1/2]	27
5.1.3.67 trim() [2/2]	27
5.2 Catch::Detail Namespace Reference	27
5.2.1 Function Documentation	28
5.2.1.1 convertUnknownEnumToString()	28
5.2.1.2 convertUnstreamable() [1/3]	28
5.2.1.3 convertUnstreamable() [2/3]	28
5.2.1.4 convertUnstreamable() [3/3]	29
5.2.1.5 rangeToString()	29
5.2.1.6 rawMemoryToString() [1/2]	29
5.2.1.7 rawMemoryToString() [2/2]	29
5.2.1.8 stringify()	29
5.2.2 Variable Documentation	29
5.2.2.1 unprintableString	29
5.3 Catch::detail Namespace Reference	30
5.4 Catch::Generators Namespace Reference	30
5.4.1 Typedef Documentation	31
5.4.1.1 GeneratorBasePtr	31
5.4.2 Function Documentation	32
5.4.2.1 acquireGeneratorTracker()	32
5.4.2.2 chunk()	32
5.4.2.3 filter()	32
5.4.2.4 from_range() [1/2]	32
5.4.2.5 from_range() [2/2]	32
5.4.2.6 generate()	33
5.4.2.7 makeGenerators() [1/4]	33
5.4.2.8 makeGenerators() [2/4]	33
5.4.2.9 makeGenerators() [3/4]	33
5.4.2.10 makeGenerators() [4/4]	33
5.4.2.11 map()	34
5.4.2.12 random() [1/2]	34
5.4.2.13 random() [2/2]	34
5.4.2.14 range() [1/2]	34
5.4.2.15 range() [2/2]	34
5.4.2.16 repeat()	35
5.4.2.17 table()	35
5.4.2.18 take()	35
5.4.2.19 value()	35
5.4.2.20 values()	35

5.5 Catch::Generators::pf Namespace Reference	35
5.5.1 Function Documentation	36
5.5.1.1 make_unique()	36
5.6 Catch::literals Namespace Reference	36
5.6.1 Function Documentation	36
5.6.1.1 operator""_a() [1/2]	36
5.6.1.2 operator""_a() [2/2]	36
5.7 Catch::Matchers Namespace Reference	36
5.7.1 Function Documentation	37
5.7.1.1 Approx()	37
5.7.1.2 Contains() [1/2]	37
5.7.1.3 Contains() [2/2]	38
5.7.1.4 EndsWith()	38
5.7.1.5 Equals() [1/2]	38
5.7.1.6 Equals() [2/2]	38
5.7.1.7 Matches()	38
5.7.1.8 Message()	38
5.7.1.9 Predicate()	38
5.7.1.10 StartsWith()	39
5.7.1.11 UnorderedEquals()	39
5.7.1.12 VectorContains()	39
5.7.1.13 WithinAbs()	39
5.7.1.14 WithinRel() [1/4]	39
5.7.1.15 WithinRel() [2/4]	39
5.7.1.16 WithinRel() [3/4]	39
5.7.1.17 WithinRel() [4/4]	40
5.7.1.18 WithinULP() [1/2]	40
5.7.1.19 WithinULP() [2/2]	40
5.8 Catch::Matchers::Exception Namespace Reference	40
5.9 Catch::Matchers::Floating Namespace Reference	40
5.10 Catch::Matchers::Generic Namespace Reference	40
5.11 Catch::Matchers::Generic::Detail Namespace Reference	41
5.11.1 Function Documentation	41
5.11.1.1 finalizeDescription()	41
5.12 Catch::Matchers::Impl Namespace Reference	41
5.13 Catch::Matchers::StdString Namespace Reference	41
5.14 Catch::Matchers::Vector Namespace Reference	41
5.15 mpl_ Namespace Reference	41
6 Class Documentation	43
6.1 Add Class Reference	43
6.1.1 Detailed Description	44

6.1.2 Constructor & Destructor Documentation	44
6.1.2.1 Add()	44
6.1.3 Member Function Documentation	44
6.1.3.1 equals()	44
6.1.3.2 hasVariable()	45
6.1.3.3 interp()	45
6.1.3.4 pretty_print()	45
6.1.3.5 print()	45
6.1.3.6 subst()	47
6.1.4 Member Data Documentation	47
6.1.4.1 lhs	47
6.1.4.2 rhs	47
6.2 Catch::always_false< T > Struct Template Reference	48
6.2.1 Detailed Description	48
6.3 Catch::Detail::Approx Class Reference	48
6.3.1 Detailed Description	49
6.3.2 Constructor & Destructor Documentation	49
6.3.2.1 Approx() [1/2]	49
6.3.2.2 Approx() [2/2]	49
6.3.3 Member Function Documentation	49
6.3.3.1 custom()	49
6.3.3.2 epsilon()	50
6.3.3.3 margin()	50
6.3.3.4 operator()()	50
6.3.3.5 operator-()	50
6.3.3.6 scale()	50
6.3.3.7 toString()	50
6.3.4 Friends And Related Symbol Documentation	51
6.3.4.1 operator"!=" [1/2]	51
6.3.4.2 operator"!=" [2/2]	51
6.3.4.3 operator<=" [1/2]	51
6.3.4.4 operator<=" [2/2]	51
6.3.4.5 operator==" [1/2]	51
6.3.4.6 operator==" [2/2]	52
6.3.4.7 operator>=" [1/2]	52
6.3.4.8 operator>=" [2/2]	52
6.4 Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch > Struct Template Reference	52
6.4.1 Detailed Description	54
6.4.2 Constructor & Destructor Documentation	54
6.4.2.1 ApproxMatcher()	54
6.4.3 Member Function Documentation	54
6.4.3.1 describe()	54

6.4.3.2 epsilon()	54
6.4.3.3 margin()	54
6.4.3.4 match()	55
6.4.3.5 scale()	55
6.4.4 Member Data Documentation	55
6.4.4.1 approx	55
6.4.4.2 m_comparator	55
6.5 Catch::Generators::as< T > Struct Template Reference	55
6.5.1 Detailed Description	56
6.6 Catch::AssertionHandler Class Reference	56
6.6.1 Detailed Description	56
6.6.2 Constructor & Destructor Documentation	56
6.6.2.1 AssertionHandler()	56
6.6.2.2 ~AssertionHandler()	57
6.6.3 Member Function Documentation	57
6.6.3.1 allowThrows()	57
6.6.3.2 complete()	57
6.6.3.3 handleExceptionNotThrownAsExpected()	57
6.6.3.4 handleExceptionThrownAsExpected()	57
6.6.3.5 handleExpr() [1/2]	57
6.6.3.6 handleExpr() [2/2]	57
6.6.3.7 handleMessage()	57
6.6.3.8 handleThrowingCallSkipped()	58
6.6.3.9 handleUnexpectedExceptionNotThrown()	58
6.6.3.10 handleUnexpectedInflightException()	58
6.6.3.11 setCompleted()	58
6.7 Catch::AssertionInfo Struct Reference	58
6.7.1 Detailed Description	58
6.7.2 Member Data Documentation	58
6.7.2.1 capturedExpression	58
6.7.2.2 lineInfo	59
6.7.2.3 macroName	59
6.7.2.4 resultDisposition	59
6.8 Catch::AssertionReaction Struct Reference	59
6.8.1 Detailed Description	59
6.8.2 Member Data Documentation	59
6.8.2.1 shouldDebugBreak	59
6.8.2.2 shouldThrow	60
6.9 Catch::AutoReg Struct Reference	60
6.9.1 Detailed Description	60
6.9.2 Constructor & Destructor Documentation	60
6.9.2.1 AutoReg()	60

6.9.2.2 ~AutoReg()	61
6.10 Catch::BinaryExpr< LhsT, RhstT > Class Template Reference	61
6.10.1 Detailed Description	62
6.10.2 Constructor & Destructor Documentation	62
6.10.2.1 BinaryExpr()	62
6.10.3 Member Function Documentation	62
6.10.3.1 operator"!=()	62
6.10.3.2 operator&&()	62
6.10.3.3 operator<()	63
6.10.3.4 operator<=()	63
6.10.3.5 operator==()	63
6.10.3.6 operator>()	63
6.10.3.7 operator>=()	63
6.10.3.8 operator" " ()	64
6.11 Catch::Capturer Class Reference	64
6.11.1 Detailed Description	64
6.11.2 Constructor & Destructor Documentation	64
6.11.2.1 Capturer()	64
6.11.2.2 ~Capturer()	64
6.11.3 Member Function Documentation	65
6.11.3.1 captureValue()	65
6.11.3.2 captureValues() [1/2]	65
6.11.3.3 captureValues() [2/2]	65
6.12 Catch::Matchers::StdString::CasedString Struct Reference	65
6.12.1 Detailed Description	66
6.12.2 Constructor & Destructor Documentation	66
6.12.2.1 CasedString()	66
6.12.3 Member Function Documentation	66
6.12.3.1 adjustString()	66
6.12.3.2 caseSensitivitySuffix()	66
6.12.4 Member Data Documentation	66
6.12.4.1 m_caseSensitivity	66
6.12.4.2 m_str	66
6.13 Catch::CaseSensitive Struct Reference	66
6.13.1 Detailed Description	67
6.13.2 Member Enumeration Documentation	67
6.13.2.1 Choice	67
6.14 Catch_global_namespace_dummy Struct Reference	67
6.14.1 Detailed Description	67
6.15 Catch::Generators::ChunkGenerator< T > Class Template Reference	68
6.15.1 Detailed Description	68
6.15.2 Constructor & Destructor Documentation	69

6.15.2.1 ChunkGenerator()	69
6.15.3 Member Function Documentation	69
6.15.3.1 get()	69
6.15.3.2 next()	69
6.16 Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc > Struct Template Reference	69
6.16.1 Detailed Description	70
6.16.2 Constructor & Destructor Documentation	71
6.16.2.1 ContainsElementMatcher()	71
6.16.3 Member Function Documentation	71
6.16.3.1 describe()	71
6.16.3.2 match()	71
6.16.4 Member Data Documentation	71
6.16.4.1 m_comparator	71
6.17 Catch::Matchers::StdString::ContainsMatcher Struct Reference	72
6.17.1 Detailed Description	73
6.17.2 Constructor & Destructor Documentation	73
6.17.2.1 ContainsMatcher()	73
6.17.3 Member Function Documentation	73
6.17.3.1 match()	73
6.18 Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch > Struct Template Reference	73
6.18.1 Detailed Description	74
6.18.2 Constructor & Destructor Documentation	75
6.18.2.1 ContainsMatcher()	75
6.18.3 Member Function Documentation	75
6.18.3.1 describe()	75
6.18.3.2 match()	75
6.18.4 Member Data Documentation	75
6.18.4.1 m_comparator	75
6.19 Catch::Counts Struct Reference	75
6.19.1 Detailed Description	76
6.19.2 Member Function Documentation	76
6.19.2.1 allOk()	76
6.19.2.2 allPassed()	76
6.19.2.3 operator+=()	76
6.19.2.4 operator-()	76
6.19.2.5 total()	76
6.19.3 Member Data Documentation	77
6.19.3.1 failed	77
6.19.3.2 failedButOk	77
6.19.3.3 passed	77
6.20 Catch::Decomposer Struct Reference	77
6.20.1 Detailed Description	77

6.20.2 Member Function Documentation	77
6.20.2.1 operator<=() [1/2]	77
6.20.2.2 operator<=() [2/2]	78
6.21 Catch::Matchers::StdString::EndsWithMatcher Struct Reference	78
6.21.1 Detailed Description	79
6.21.2 Constructor & Destructor Documentation	79
6.21.2.1 EndsWithMatcher()	79
6.21.3 Member Function Documentation	79
6.21.3.1 match()	79
6.22 Catch::Detail::EnumInfo Struct Reference	79
6.22.1 Detailed Description	80
6.22.2 Constructor & Destructor Documentation	80
6.22.2.1 ~EnumInfo()	80
6.22.3 Member Function Documentation	80
6.22.3.1 lookup()	80
6.22.4 Member Data Documentation	80
6.22.4.1 m_name	80
6.22.4.2 m_values	80
6.23 Catch::Matchers::StdString::EqualsMatcher Struct Reference	81
6.23.1 Detailed Description	82
6.23.2 Constructor & Destructor Documentation	82
6.23.2.1 EqualsMatcher()	82
6.23.3 Member Function Documentation	82
6.23.3.1 match()	82
6.24 Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch > Struct Template Reference	82
6.24.1 Detailed Description	83
6.24.2 Constructor & Destructor Documentation	84
6.24.2.1 EqualsMatcher()	84
6.24.3 Member Function Documentation	84
6.24.3.1 describe()	84
6.24.3.2 match()	84
6.24.4 Member Data Documentation	84
6.24.4.1 m_comparator	84
6.25 Catch::Matchers::Exception::ExceptionMessageMatcher Class Reference	85
6.25.1 Detailed Description	86
6.25.2 Constructor & Destructor Documentation	86
6.25.2.1 ExceptionMessageMatcher()	86
6.25.3 Member Function Documentation	86
6.25.3.1 describe()	86
6.25.3.2 match()	86
6.26 Catch::ExceptionTranslatorRegistrar Class Reference	86
6.26.1 Detailed Description	86

6.26.2 Constructor & Destructor Documentation	87
6.26.2.1 ExceptionTranslatorRegistrar()	87
6.27 Expr Class Reference	87
6.27.1 Detailed Description	87
6.27.2 Constructor & Destructor Documentation	87
6.27.2.1 ~Expr()	87
6.27.3 Member Function Documentation	88
6.27.3.1 equals()	88
6.27.3.2 hasVariable()	88
6.27.3.3 interp()	88
6.27.3.4 pretty_print()	88
6.27.3.5 pretty_print_at()	88
6.27.3.6 print()	88
6.27.3.7 subst()	89
6.27.3.8 to_pp_string()	89
6.27.3.9 to_string()	89
6.28 Catch::ExprLhs< LhsT > Class Template Reference	89
6.28.1 Detailed Description	90
6.28.2 Constructor & Destructor Documentation	90
6.28.2.1 ExprLhs()	90
6.28.3 Member Function Documentation	90
6.28.3.1 makeUnaryExpr()	90
6.28.3.2 operator!=() [1/2]	90
6.28.3.3 operator!=() [2/2]	90
6.28.3.4 operator&()	91
6.28.3.5 operator&&()	91
6.28.3.6 operator<()	91
6.28.3.7 operator<=()	91
6.28.3.8 operator==() [1/2]	91
6.28.3.9 operator==() [2/2]	91
6.28.3.10 operator>()	92
6.28.3.11 operator>=()	92
6.28.3.12 operator^()	92
6.28.3.13 operator" ()	92
6.28.3.14 operator" " ()	92
6.29 Catch::Generators::FilterGenerator< T, Predicate > Class Template Reference	93
6.29.1 Detailed Description	93
6.29.2 Constructor & Destructor Documentation	94
6.29.2.1 FilterGenerator()	94
6.29.3 Member Function Documentation	94
6.29.3.1 get()	94
6.29.3.2 next()	94

6.30 Catch::Generators::FixedValuesGenerator< T > Class Template Reference	94
6.30.1 Detailed Description	95
6.30.2 Constructor & Destructor Documentation	95
6.30.2.1 FixedValuesGenerator()	95
6.30.3 Member Function Documentation	95
6.30.3.1 get()	95
6.30.3.2 next()	96
6.31 Catch::GeneratorException Class Reference	96
6.31.1 Detailed Description	96
6.31.2 Constructor & Destructor Documentation	96
6.31.2.1 GeneratorException()	96
6.31.3 Member Function Documentation	97
6.31.3.1 what()	97
6.32 Catch::Generators::Generators< T > Class Template Reference	97
6.32.1 Detailed Description	98
6.32.2 Constructor & Destructor Documentation	98
6.32.2.1 Generators()	98
6.32.3 Member Function Documentation	98
6.32.3.1 get()	98
6.32.3.2 next()	98
6.33 Catch::Generators::GeneratorUntypedBase Class Reference	99
6.33.1 Detailed Description	99
6.33.2 Constructor & Destructor Documentation	99
6.33.2.1 GeneratorUntypedBase()	99
6.33.2.2 ~GeneratorUntypedBase()	99
6.33.3 Member Function Documentation	99
6.33.3.1 next()	99
6.34 Catch::Generators::GeneratorWrapper< T > Class Template Reference	100
6.34.1 Detailed Description	100
6.34.2 Constructor & Destructor Documentation	100
6.34.2.1 GeneratorWrapper()	100
6.34.3 Member Function Documentation	100
6.34.3.1 get()	100
6.34.3.2 next()	100
6.35 Catch::IConfig Struct Reference	101
6.35.1 Detailed Description	101
6.35.2 Constructor & Destructor Documentation	102
6.35.2.1 ~IConfig()	102
6.35.3 Member Function Documentation	102
6.35.3.1 abortAfter()	102
6.35.3.2 allowThrows()	102
6.35.3.3 benchmarkConfidenceInterval()	102

6.35.3.4 benchmarkNoAnalysis()	102
6.35.3.5 benchmarkResamples()	102
6.35.3.6 benchmarkSamples()	102
6.35.3.7 benchmarkWarmupTime()	102
6.35.3.8 getSectionsToRun()	102
6.35.3.9 getTestsOrTags()	103
6.35.3.10 hasTestFilters()	103
6.35.3.11 includeSuccessfulResults()	103
6.35.3.12 minDuration()	103
6.35.3.13 name()	103
6.35.3.14 rngSeed()	103
6.35.3.15 runOrder()	103
6.35.3.16 shouldDebugBreak()	103
6.35.3.17 showDurations()	103
6.35.3.18 showInvisibles()	103
6.35.3.19 stream()	104
6.35.3.20 testSpec()	104
6.35.3.21 useColour()	104
6.35.3.22 verbosity()	104
6.35.3.23 warnAboutMissingAssertions()	104
6.35.3.24 warnAboutNoTests()	104
6.36 Catch::IContext Struct Reference	104
6.36.1 Detailed Description	105
6.36.2 Constructor & Destructor Documentation	105
6.36.2.1 ~IContext()	105
6.36.3 Member Function Documentation	105
6.36.3.1 getConfig()	105
6.36.3.2 getResultCapture()	105
6.36.3.3 getRunner()	105
6.37 Catch::IExceptionTranslator Struct Reference	105
6.37.1 Detailed Description	105
6.37.2 Constructor & Destructor Documentation	106
6.37.2.1 ~IExceptionTranslator()	106
6.37.3 Member Function Documentation	106
6.37.3.1 translate()	106
6.38 Catch::IExceptionTranslatorRegistry Struct Reference	106
6.38.1 Detailed Description	106
6.38.2 Constructor & Destructor Documentation	106
6.38.2.1 ~IExceptionTranslatorRegistry()	106
6.38.3 Member Function Documentation	106
6.38.3.1 translateActiveException()	106
6.39 Catch::Generators::IGenerator< T > Struct Template Reference	107

6.39.1 Detailed Description	107
6.39.2 Member Typedef Documentation	108
6.39.2.1 type	108
6.39.3 Constructor & Destructor Documentation	108
6.39.3.1 ~IGenerator()	108
6.39.4 Member Function Documentation	108
6.39.4.1 get()	108
6.40 Catch::IGeneratorTracker Struct Reference	108
6.40.1 Detailed Description	108
6.40.2 Constructor & Destructor Documentation	109
6.40.2.1 ~IGeneratorTracker()	109
6.40.3 Member Function Documentation	109
6.40.3.1 getGenerator()	109
6.40.3.2 hasGenerator()	109
6.40.3.3 setGenerator()	109
6.41 Catch::IMutableContext Struct Reference	109
6.41.1 Detailed Description	110
6.41.2 Constructor & Destructor Documentation	110
6.41.2.1 ~IMutableContext()	110
6.41.3 Member Function Documentation	110
6.41.3.1 setConfig()	110
6.41.3.2 setResultCapture()	110
6.41.3.3 setRunner()	110
6.41.4 Friends And Related Symbol Documentation	110
6.41.4.1 cleanUpContext	110
6.41.4.2 getCurrentMutableContext	111
6.42 Catch::IMutableEnumValuesRegistry Struct Reference	111
6.42.1 Detailed Description	111
6.42.2 Constructor & Destructor Documentation	111
6.42.2.1 ~IMutableEnumValuesRegistry()	111
6.42.3 Member Function Documentation	111
6.42.3.1 registerEnum() [1/2]	111
6.42.3.2 registerEnum() [2/2]	112
6.43 Catch::IMutableRegistryHub Struct Reference	112
6.43.1 Detailed Description	112
6.43.2 Constructor & Destructor Documentation	112
6.43.2.1 ~IMutableRegistryHub()	112
6.43.3 Member Function Documentation	112
6.43.3.1 getMutableEnumValuesRegistry()	112
6.43.3.2 registerListener()	113
6.43.3.3 registerReporter()	113
6.43.3.4 registerStartupException()	113

6.43.3.5 registerTagAlias()	113
6.43.3.6 registerTest()	113
6.43.3.7 registerTranslator()	113
6.44 Catch::IRegistryHub Struct Reference	113
6.44.1 Detailed Description	114
6.44.2 Constructor & Destructor Documentation	114
6.44.2.1 ~IRegistryHub()	114
6.44.3 Member Function Documentation	114
6.44.3.1 getExceptionTranslatorRegistry()	114
6.44.3.2 getReporterRegistry()	114
6.44.3.3 getStartupExceptionRegistry()	114
6.44.3.4 getTagAliasRegistry()	114
6.44.3.5 getTestCaseRegistry()	115
6.45 Catch::IResultCapture Struct Reference	115
6.45.1 Detailed Description	115
6.45.2 Constructor & Destructor Documentation	116
6.45.2.1 ~IResultCapture()	116
6.45.3 Member Function Documentation	116
6.45.3.1 acquireGeneratorTracker()	116
6.45.3.2 assertionPassed()	116
6.45.3.3 emplaceUnscopedMessage()	116
6.45.3.4 exceptionEarlyReported()	116
6.45.3.5 getCurrentTestName()	116
6.45.3.6 getLastResult()	116
6.45.3.7 handleExpr()	116
6.45.3.8 handleFatalErrorCondition()	117
6.45.3.9 handleIncomplete()	117
6.45.3.10 handleMessage()	117
6.45.3.11 handleNonExpr()	117
6.45.3.12 handleUnexpectedExceptionNotThrown()	117
6.45.3.13 handleUnexpectedInflightException()	117
6.45.3.14 lastAssertionPassed()	117
6.45.3.15 popScopedMessage()	117
6.45.3.16 pushScopedMessage()	118
6.45.3.17 sectionEnded()	118
6.45.3.18 sectionEndedEarly()	118
6.45.3.19 sectionStarted()	118
6.46 Catch::IRunner Struct Reference	118
6.46.1 Detailed Description	118
6.46.2 Constructor & Destructor Documentation	118
6.46.2.1 ~IRunner()	118
6.46.3 Member Function Documentation	119

6.46.3.1 aborting()	119
6.47 Catch::is_callable< T > Struct Template Reference	119
6.47.1 Detailed Description	119
6.48 Catch::is_callable< Fun(Args...) > Struct Template Reference	119
6.48.1 Detailed Description	119
6.49 Catch::is_callable_tester Struct Reference	119
6.49.1 Detailed Description	120
6.49.2 Member Function Documentation	120
6.49.2.1 test() [1/2]	120
6.49.2.2 test() [2/2]	120
6.50 Catch::is_range< T > Struct Template Reference	120
6.50.1 Detailed Description	121
6.51 Catch::detail::is_range_impl< T, typename > Struct Template Reference	121
6.51.1 Detailed Description	121
6.52 Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >())) >::type > Struct Template Reference	121
6.52.1 Detailed Description	122
6.53 Catch::Detail::IsStreamInsertable< T > Class Template Reference	122
6.53.1 Detailed Description	122
6.53.2 Member Data Documentation	122
6.53.2.1 value	122
6.54 Catch::IStream Struct Reference	122
6.54.1 Detailed Description	123
6.54.2 Constructor & Destructor Documentation	123
6.54.2.1 ~IStream()	123
6.54.3 Member Function Documentation	123
6.54.3.1 stream()	123
6.55 Catch::Generators::IteratorGenerator< T > Class Template Reference	123
6.55.1 Detailed Description	124
6.55.2 Constructor & Destructor Documentation	124
6.55.2.1 IteratorGenerator()	124
6.55.3 Member Function Documentation	124
6.55.3.1 get()	124
6.55.3.2 next()	125
6.56 Catch::ITestCaseRegistry Struct Reference	125
6.56.1 Detailed Description	125
6.56.2 Constructor & Destructor Documentation	125
6.56.2.1 ~ITestCaseRegistry()	125
6.56.3 Member Function Documentation	125
6.56.3.1 getAllTests()	125
6.56.3.2 getAllTestsSorted()	125
6.57 Catch::ITestInvoker Struct Reference	126

6.57.1 Detailed Description	126
6.57.2 Constructor & Destructor Documentation	126
6.57.2.1 ~ITestInvoker()	126
6.57.3 Member Function Documentation	126
6.57.3.1 invoke()	126
6.58 Catch::ITransientExpression Struct Reference	126
6.58.1 Detailed Description	127
6.58.2 Constructor & Destructor Documentation	127
6.58.2.1 ITransientExpression()	127
6.58.2.2 ~ITransientExpression()	127
6.58.3 Member Function Documentation	127
6.58.3.1 getResult()	127
6.58.3.2 isBinaryExpression()	127
6.58.3.3 streamReconstructedExpression()	128
6.58.4 Member Data Documentation	128
6.58.4.1 m_isBinaryExpression	128
6.58.4.2 m_result	128
6.59 Catch::LazyExpression Class Reference	128
6.59.1 Detailed Description	128
6.59.2 Constructor & Destructor Documentation	129
6.59.2.1 LazyExpression() [1/2]	129
6.59.2.2 LazyExpression() [2/2]	129
6.59.3 Member Function Documentation	129
6.59.3.1 operator bool()	129
6.59.3.2 operator=()	129
6.59.4 Friends And Related Symbol Documentation	129
6.59.4.1 AssertionHandler	129
6.59.4.2 AssertionStats	129
6.59.4.3 operator<<	129
6.59.4.4 RunContext	130
6.60 Catch::Generators::MapGenerator< T, U, Func > Class Template Reference	130
6.60.1 Detailed Description	131
6.60.2 Constructor & Destructor Documentation	131
6.60.2.1 MapGenerator()	131
6.60.3 Member Function Documentation	131
6.60.3.1 get()	131
6.60.3.2 next()	131
6.61 Catch::Matchers::Impl::MatchAllOf< ArgT > Struct Template Reference	132
6.61.1 Detailed Description	133
6.61.2 Member Function Documentation	133
6.61.2.1 describe()	133
6.61.2.2 match()	133

6.61.2.3 operator&&()	133
6.61.3 Member Data Documentation	133
6.61.3.1 m_matchers	133
6.62 Catch::Matchers::Impl::MatchAnyOf< ArgT > Struct Template Reference	134
6.62.1 Detailed Description	135
6.62.2 Member Function Documentation	135
6.62.2.1 describe()	135
6.62.2.2 match()	135
6.62.2.3 operator" " ()	135
6.62.3 Member Data Documentation	135
6.62.3.1 m_matchers	135
6.63 Catch::Matchers::Impl::MatcherBase< T > Struct Template Reference	136
6.63.1 Detailed Description	137
6.63.2 Member Function Documentation	137
6.63.2.1 operator"!()	137
6.63.2.2 operator&&()	137
6.63.2.3 operator" " ()	137
6.64 Catch::Matchers::Impl::MatcherMethod< ObjectT > Struct Template Reference	137
6.64.1 Detailed Description	138
6.64.2 Member Function Documentation	138
6.64.2.1 match()	138
6.65 Catch::Matchers::Impl::MatcherUntypedBase Class Reference	138
6.65.1 Detailed Description	139
6.65.2 Constructor & Destructor Documentation	139
6.65.2.1 MatcherUntypedBase() [1/2]	139
6.65.2.2 MatcherUntypedBase() [2/2]	139
6.65.2.3 ~MatcherUntypedBase()	139
6.65.3 Member Function Documentation	139
6.65.3.1 describe()	139
6.65.3.2 operator=()	139
6.65.3.3 toString()	139
6.65.4 Member Data Documentation	140
6.65.4.1 m_cachedToString	140
6.66 Catch::MatchExpr< ArgT, MatcherT > Class Template Reference	140
6.66.1 Detailed Description	140
6.66.2 Constructor & Destructor Documentation	141
6.66.2.1 MatchExpr()	141
6.66.3 Member Function Documentation	141
6.66.3.1 streamReconstructedExpression()	141
6.67 Catch::Matchers::Impl::MatchNotOf< ArgT > Struct Template Reference	141
6.67.1 Detailed Description	142
6.67.2 Constructor & Destructor Documentation	142

6.67.2.1 MatchNotOf()	142
6.67.3 Member Function Documentation	143
6.67.3.1 describe()	143
6.67.3.2 match()	143
6.67.4 Member Data Documentation	143
6.67.4.1 m_underlyingMatcher	143
6.68 Catch::MessageBuilder Struct Reference	143
6.68.1 Detailed Description	144
6.68.2 Constructor & Destructor Documentation	144
6.68.2.1 MessageBuilder()	144
6.68.3 Member Function Documentation	144
6.68.3.1 operator<<()	144
6.68.4 Member Data Documentation	144
6.68.4.1 m_info	144
6.69 Catch::MessageInfo Struct Reference	145
6.69.1 Detailed Description	145
6.69.2 Constructor & Destructor Documentation	145
6.69.2.1 MessageInfo()	145
6.69.3 Member Function Documentation	145
6.69.3.1 operator<()	145
6.69.3.2 operator==(())	145
6.69.4 Member Data Documentation	146
6.69.4.1 lineInfo	146
6.69.4.2 macroName	146
6.69.4.3 message	146
6.69.4.4 sequence	146
6.69.4.5 type	146
6.70 Catch::MessageStream Struct Reference	146
6.70.1 Detailed Description	147
6.70.2 Member Function Documentation	147
6.70.2.1 operator<<()	147
6.70.3 Member Data Documentation	147
6.70.3.1 m_stream	147
6.71 Mult Class Reference	147
6.71.1 Detailed Description	148
6.71.2 Constructor & Destructor Documentation	148
6.71.2.1 Mult()	148
6.71.3 Member Function Documentation	149
6.71.3.1 equals()	149
6.71.3.2 hasVariable()	149
6.71.3.3 interp()	149
6.71.3.4 pretty_print()	149

6.71.3.5 print()	150
6.71.3.6 subst()	150
6.71.4 Member Data Documentation	150
6.71.4.1 lhs	150
6.71.4.2 rhs	151
6.72 Catch::NameAndTags Struct Reference	151
6.72.1 Detailed Description	151
6.72.2 Constructor & Destructor Documentation	151
6.72.2.1 NameAndTags()	151
6.72.3 Member Data Documentation	151
6.72.3.1 name	151
6.72.3.2 tags	152
6.73 Catch::NonCopyable Class Reference	152
6.73.1 Detailed Description	152
6.73.2 Constructor & Destructor Documentation	152
6.73.2.1 NonCopyable()	152
6.73.2.2 ~NonCopyable()	152
6.74 Num Class Reference	153
6.74.1 Detailed Description	153
6.74.2 Constructor & Destructor Documentation	153
6.74.2.1 Num()	153
6.74.3 Member Function Documentation	154
6.74.3.1 equals()	154
6.74.3.2 hasVariable()	154
6.74.3.3 interp()	154
6.74.3.4 pretty_print()	155
6.74.3.5 print()	155
6.74.3.6 subst()	155
6.74.4 Member Data Documentation	156
6.74.4.1 val	156
6.75 Catch::Option< T > Class Template Reference	156
6.75.1 Detailed Description	156
6.75.2 Constructor & Destructor Documentation	157
6.75.2.1 Option() [1/3]	157
6.75.2.2 Option() [2/3]	157
6.75.2.3 Option() [3/3]	157
6.75.2.4 ~Option()	157
6.75.3 Member Function Documentation	157
6.75.3.1 none()	157
6.75.3.2 operator bool()	157
6.75.3.3 operator"!()	158
6.75.3.4 operator*() [1/2]	158

6.75.3.5 operator*() [2/2]	158
6.75.3.6 operator->() [1/2]	158
6.75.3.7 operator->() [2/2]	158
6.75.3.8 operator=() [1/2]	158
6.75.3.9 operator=() [2/2]	158
6.75.3.10 reset()	159
6.75.3.11 some()	159
6.75.3.12 valueOr()	159
6.76 Catch::pluralise Struct Reference	159
6.76.1 Detailed Description	159
6.76.2 Constructor & Destructor Documentation	160
6.76.2.1 pluralise()	160
6.76.3 Friends And Related Symbol Documentation	160
6.76.3.1 operator<<	160
6.76.4 Member Data Documentation	160
6.76.4.1 m_count	160
6.76.4.2 m_label	160
6.77 Catch::Matchers::Generic::PredicateMatcher< T > Class Template Reference	160
6.77.1 Detailed Description	161
6.77.2 Constructor & Destructor Documentation	161
6.77.2.1 PredicateMatcher()	161
6.77.3 Member Function Documentation	162
6.77.3.1 describe()	162
6.77.3.2 match()	162
6.78 Catch::Generators::RandomFloatingGenerator< Float > Class Template Reference	162
6.78.1 Detailed Description	163
6.78.2 Constructor & Destructor Documentation	163
6.78.2.1 RandomFloatingGenerator()	163
6.78.3 Member Function Documentation	163
6.78.3.1 get()	163
6.78.3.2 next()	164
6.79 Catch::Generators::RandomIntegerGenerator< Integer > Class Template Reference	164
6.79.1 Detailed Description	165
6.79.2 Constructor & Destructor Documentation	165
6.79.2.1 RandomIntegerGenerator()	165
6.79.3 Member Function Documentation	165
6.79.3.1 get()	165
6.79.3.2 next()	165
6.80 Catch::Generators::RangeGenerator< T > Class Template Reference	166
6.80.1 Detailed Description	166
6.80.2 Constructor & Destructor Documentation	167
6.80.2.1 RangeGenerator() [1/2]	167

6.80.2.2 RangeGenerator() [2/2]	167
6.80.3 Member Function Documentation	167
6.80.3.1 get()	167
6.80.3.2 next()	167
6.81 Catch::Matchers::StdString::RegexMatcher Struct Reference	168
6.81.1 Detailed Description	169
6.81.2 Constructor & Destructor Documentation	169
6.81.2.1 RegexMatcher()	169
6.81.3 Member Function Documentation	169
6.81.3.1 describe()	169
6.81.3.2 match()	169
6.82 Catch::RegistrarForTagAliases Struct Reference	169
6.82.1 Detailed Description	169
6.82.2 Constructor & Destructor Documentation	170
6.82.2.1 RegistrarForTagAliases()	170
6.83 Catch::Generators::RepeatGenerator< T > Class Template Reference	170
6.83.1 Detailed Description	171
6.83.2 Constructor & Destructor Documentation	171
6.83.2.1 RepeatGenerator()	171
6.83.3 Member Function Documentation	171
6.83.3.1 get()	171
6.83.3.2 next()	171
6.84 Catch::ResultDisposition Struct Reference	171
6.84.1 Detailed Description	172
6.84.2 Member Enumeration Documentation	172
6.84.2.1 Flags	172
6.85 Catch::ResultWas Struct Reference	172
6.85.1 Detailed Description	172
6.85.2 Member Enumeration Documentation	172
6.85.2.1 OfType	172
6.86 Catch::ReusableStringStream Class Reference	173
6.86.1 Detailed Description	173
6.86.2 Constructor & Destructor Documentation	174
6.86.2.1 ReusableStringStream()	174
6.86.2.2 ~ReusableStringStream()	174
6.86.3 Member Function Documentation	174
6.86.3.1 get()	174
6.86.3.2 operator<<()	174
6.86.3.3 str()	174
6.87 Catch::RunTests Struct Reference	174
6.87.1 Detailed Description	174
6.87.2 Member Enumeration Documentation	174

6.87.2.1 InWhatOrder	174
6.88 Catch::ScopedMessage Class Reference	175
6.88.1 Detailed Description	175
6.88.2 Constructor & Destructor Documentation	175
6.88.2.1 ScopedMessage() [1/3]	175
6.88.2.2 ScopedMessage() [2/3]	175
6.88.2.3 ScopedMessage() [3/3]	176
6.88.2.4 ~ScopedMessage()	176
6.88.3 Member Data Documentation	176
6.88.3.1 m_info	176
6.88.3.2 m_moved	176
6.89 Catch::Section Class Reference	176
6.89.1 Detailed Description	177
6.89.2 Constructor & Destructor Documentation	177
6.89.2.1 Section()	177
6.89.2.2 ~Section()	177
6.89.3 Member Function Documentation	177
6.89.3.1 operator bool()	177
6.90 Catch::SectionEndInfo Struct Reference	177
6.90.1 Detailed Description	177
6.90.2 Member Data Documentation	178
6.90.2.1 durationInSeconds	178
6.90.2.2 prevAssertions	178
6.90.2.3 sectionInfo	178
6.91 Catch::SectionInfo Struct Reference	178
6.91.1 Detailed Description	178
6.91.2 Constructor & Destructor Documentation	179
6.91.2.1 SectionInfo() [1/2]	179
6.91.2.2 SectionInfo() [2/2]	179
6.91.3 Member Data Documentation	179
6.91.3.1 description	179
6.91.3.2 lineInfo	179
6.91.3.3 name	179
6.92 Catch::ShowDurations Struct Reference	179
6.92.1 Detailed Description	180
6.92.2 Member Enumeration Documentation	180
6.92.2.1 OrNot	180
6.93 Catch::SimplePcg32 Class Reference	180
6.93.1 Detailed Description	181
6.93.2 Member Typedef Documentation	181
6.93.2.1 result_type	181
6.93.3 Constructor & Destructor Documentation	181

6.93.3.1 SimplePcg32() [1/2]	181
6.93.3.2 SimplePcg32() [2/2]	181
6.93.4 Member Function Documentation	181
6.93.4.1 discard()	181
6.93.4.2 max()	181
6.93.4.3 min()	182
6.93.4.4 operator()()	182
6.93.4.5 seed()	182
6.93.5 Friends And Related Symbol Documentation	182
6.93.5.1 operator"!="	182
6.93.5.2 operator==	182
6.94 Catch::Generators::SingleValueGenerator< T > Class Template Reference	182
6.94.1 Detailed Description	183
6.94.2 Constructor & Destructor Documentation	183
6.94.2.1 SingleValueGenerator()	183
6.94.3 Member Function Documentation	183
6.94.3.1 get()	183
6.94.3.2 next()	184
6.95 Catch::SourceLineInfo Struct Reference	184
6.95.1 Detailed Description	184
6.95.2 Constructor & Destructor Documentation	184
6.95.2.1 SourceLineInfo() [1/4]	184
6.95.2.2 SourceLineInfo() [2/4]	185
6.95.2.3 SourceLineInfo() [3/4]	185
6.95.2.4 SourceLineInfo() [4/4]	185
6.95.3 Member Function Documentation	185
6.95.3.1 empty()	185
6.95.3.2 operator<()	185
6.95.3.3 operator=() [1/2]	185
6.95.3.4 operator=() [2/2]	185
6.95.3.5 operator==()	185
6.95.4 Member Data Documentation	186
6.95.4.1 file	186
6.95.4.2 line	186
6.96 Catch::Matchers::StdString::StartsWithMatcher Struct Reference	186
6.96.1 Detailed Description	187
6.96.2 Constructor & Destructor Documentation	187
6.96.2.1 StartsWithMatcher()	187
6.96.3 Member Function Documentation	187
6.96.3.1 match()	187
6.97 Catch::StreamEndStop Struct Reference	188
6.97.1 Detailed Description	188

6.97.2 Member Function Documentation	188
6.97.2.1 operator+()	188
6.98 Catch::StringMaker< T, typename > Struct Template Reference	188
6.98.1 Detailed Description	188
6.98.2 Member Function Documentation	189
6.98.2.1 convert() [1/2]	189
6.98.2.2 convert() [2/2]	189
6.99 Catch::StringMaker< bool > Struct Reference	189
6.99.1 Detailed Description	189
6.99.2 Member Function Documentation	189
6.99.2.1 convert()	189
6.100 Catch::StringMaker< Catch::Detail::Approx > Struct Reference	190
6.100.1 Detailed Description	190
6.100.2 Member Function Documentation	190
6.100.2.1 convert()	190
6.101 Catch::StringMaker< char * > Struct Reference	190
6.101.1 Detailed Description	190
6.101.2 Member Function Documentation	190
6.101.2.1 convert()	190
6.102 Catch::StringMaker< char > Struct Reference	191
6.102.1 Detailed Description	191
6.102.2 Member Function Documentation	191
6.102.2.1 convert()	191
6.103 Catch::StringMaker< char const * > Struct Reference	191
6.103.1 Detailed Description	191
6.103.2 Member Function Documentation	191
6.103.2.1 convert()	191
6.104 Catch::StringMaker< char[SZ]> Struct Template Reference	192
6.104.1 Detailed Description	192
6.104.2 Member Function Documentation	192
6.104.2.1 convert()	192
6.105 Catch::StringMaker< double > Struct Reference	192
6.105.1 Detailed Description	192
6.105.2 Member Function Documentation	193
6.105.2.1 convert()	193
6.105.3 Member Data Documentation	193
6.105.3.1 precision	193
6.106 Catch::StringMaker< float > Struct Reference	193
6.106.1 Detailed Description	193
6.106.2 Member Function Documentation	193
6.106.2.1 convert()	193
6.106.3 Member Data Documentation	194

6.106.3.1 precision	194
6.107 Catch::StringMaker< int > Struct Reference	194
6.107.1 Detailed Description	194
6.107.2 Member Function Documentation	194
6.107.2.1 convert()	194
6.108 Catch::StringMaker< long > Struct Reference	194
6.108.1 Detailed Description	195
6.108.2 Member Function Documentation	195
6.108.2.1 convert()	195
6.109 Catch::StringMaker< long long > Struct Reference	195
6.109.1 Detailed Description	195
6.109.2 Member Function Documentation	195
6.109.2.1 convert()	195
6.110 Catch::StringMaker< R C::* > Struct Template Reference	195
6.110.1 Detailed Description	196
6.110.2 Member Function Documentation	196
6.110.2.1 convert()	196
6.111 Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStreamInsertable< R >::value >::type > Struct Template Reference	196
6.111.1 Detailed Description	196
6.111.2 Member Function Documentation	197
6.111.2.1 convert()	197
6.112 Catch::StringMaker< signed char > Struct Reference	197
6.112.1 Detailed Description	197
6.112.2 Member Function Documentation	197
6.112.2.1 convert()	197
6.113 Catch::StringMaker< signed char[SZ]> Struct Template Reference	197
6.113.1 Detailed Description	198
6.113.2 Member Function Documentation	198
6.113.2.1 convert()	198
6.114 Catch::StringMaker< std::nullptr_t > Struct Reference	198
6.114.1 Detailed Description	198
6.114.2 Member Function Documentation	198
6.114.2.1 convert()	198
6.115 Catch::StringMaker< std::string > Struct Reference	199
6.115.1 Detailed Description	199
6.115.2 Member Function Documentation	199
6.115.2.1 convert()	199
6.116 Catch::StringMaker< std::wstring > Struct Reference	199
6.116.1 Detailed Description	199
6.116.2 Member Function Documentation	199
6.116.2.1 convert()	199

6.117 Catch::StringMaker< T * > Struct Template Reference	200
6.117.1 Detailed Description	200
6.117.2 Member Function Documentation	200
6.117.2.1 convert()	200
6.118 Catch::StringMaker< T[SZ]> Struct Template Reference	200
6.118.1 Detailed Description	200
6.118.2 Member Function Documentation	201
6.118.2.1 convert()	201
6.119 Catch::StringMaker< unsigned char > Struct Reference	201
6.119.1 Detailed Description	201
6.119.2 Member Function Documentation	201
6.119.2.1 convert()	201
6.120 Catch::StringMaker< unsigned char[SZ]> Struct Template Reference	201
6.120.1 Detailed Description	202
6.120.2 Member Function Documentation	202
6.120.2.1 convert()	202
6.121 Catch::StringMaker< unsigned int > Struct Reference	202
6.121.1 Detailed Description	202
6.121.2 Member Function Documentation	202
6.121.2.1 convert()	202
6.122 Catch::StringMaker< unsigned long > Struct Reference	203
6.122.1 Detailed Description	203
6.122.2 Member Function Documentation	203
6.122.2.1 convert()	203
6.123 Catch::StringMaker< unsigned long long > Struct Reference	203
6.123.1 Detailed Description	203
6.123.2 Member Function Documentation	203
6.123.2.1 convert()	203
6.124 Catch::StringMaker< wchar_t * > Struct Reference	204
6.124.1 Detailed Description	204
6.124.2 Member Function Documentation	204
6.124.2.1 convert()	204
6.125 Catch::StringMaker< wchar_t const * > Struct Reference	204
6.125.1 Detailed Description	204
6.125.2 Member Function Documentation	204
6.125.2.1 convert()	204
6.126 Catch::Matchers::StdString::StringMatcherBase Struct Reference	205
6.126.1 Detailed Description	206
6.126.2 Constructor & Destructor Documentation	206
6.126.2.1 StringMatcherBase()	206
6.126.3 Member Function Documentation	206
6.126.3.1 describe()	206

6.126.4 Member Data Documentation	206
6.126.4.1 m_comparator	206
6.126.4.2 m_operation	206
6.127 Catch::StringRef Class Reference	207
6.127.1 Detailed Description	207
6.127.2 Member Typedef Documentation	207
6.127.2.1 const_iterator	207
6.127.2.2 size_type	207
6.127.3 Constructor & Destructor Documentation	208
6.127.3.1 StringRef() [1/4]	208
6.127.3.2 StringRef() [2/4]	208
6.127.3.3 StringRef() [3/4]	208
6.127.3.4 StringRef() [4/4]	208
6.127.4 Member Function Documentation	208
6.127.4.1 begin()	208
6.127.4.2 c_str()	208
6.127.4.3 data()	208
6.127.4.4 empty()	209
6.127.4.5 end()	209
6.127.4.6 isNullTerminated()	209
6.127.4.7 operator std::string()	209
6.127.4.8 operator"!=()	209
6.127.4.9 operator==(())	209
6.127.4.10 operator[]()	209
6.127.4.11 size()	210
6.127.4.12 substr()	210
6.128 Catch::Generators::TakeGenerator< T > Class Template Reference	210
6.128.1 Detailed Description	211
6.128.2 Constructor & Destructor Documentation	211
6.128.2.1 TakeGenerator()	211
6.128.3 Member Function Documentation	211
6.128.3.1 get()	211
6.128.3.2 next()	211
6.129 Catch::TestCase Class Reference	212
6.129.1 Detailed Description	213
6.129.2 Constructor & Destructor Documentation	213
6.129.2.1 TestCase()	213
6.129.3 Member Function Documentation	213
6.129.3.1 getTestCaseInfo()	213
6.129.3.2 invoke()	213
6.129.3.3 operator<()	213
6.129.3.4 operator==(())	213

6.129.3.5 withName()	213
6.130 Catch::TestCaseInfo Struct Reference	214
6.130.1 Detailed Description	214
6.130.2 Member Enumeration Documentation	214
6.130.2.1 SpecialProperties	214
6.130.3 Constructor & Destructor Documentation	215
6.130.3.1 TestCaseInfo()	215
6.130.4 Member Function Documentation	215
6.130.4.1 expectedToFail()	215
6.130.4.2 isHidden()	215
6.130.4.3 okToFail()	215
6.130.4.4 tagsAsString()	215
6.130.4.5 throws()	215
6.130.5 Friends And Related Symbol Documentation	216
6.130.5.1 setTags	216
6.130.6 Member Data Documentation	216
6.130.6.1 className	216
6.130.6.2 description	216
6.130.6.3 lcaseTags	216
6.130.6.4 lineInfo	216
6.130.6.5 name	216
6.130.6.6 properties	216
6.130.6.7 tags	217
6.131 Catch::TestFailureException Struct Reference	217
6.131.1 Detailed Description	217
6.132 Catch::TestInvokerAsMethod< C > Class Template Reference	217
6.132.1 Detailed Description	218
6.132.2 Constructor & Destructor Documentation	218
6.132.2.1 TestInvokerAsMethod()	218
6.132.3 Member Function Documentation	218
6.132.3.1 invoke()	218
6.133 Catch::Timer Class Reference	218
6.133.1 Detailed Description	218
6.133.2 Member Function Documentation	219
6.133.2.1 getElapsedMicroseconds()	219
6.133.2.2 getElapsedMilliseconds()	219
6.133.2.3 getElapsedNanoseconds()	219
6.133.2.4 getElapsedSeconds()	219
6.133.2.5 start()	219
6.134 Catch::Totals Struct Reference	219
6.134.1 Detailed Description	220
6.134.2 Member Function Documentation	220

6.134.2.1 delta()	220
6.134.2.2 operator+=()	220
6.134.2.3 operator-()	220
6.134.3 Member Data Documentation	220
6.134.3.1 assertions	220
6.134.3.2 error	220
6.134.3.3 testCases	220
6.135 Catch::true_given< typename > Struct Template Reference	221
6.135.1 Detailed Description	221
6.136 Catch::UnaryExpr< LhsT > Class Template Reference	221
6.136.1 Detailed Description	222
6.136.2 Constructor & Destructor Documentation	222
6.136.2.1 UnaryExpr()	222
6.137 Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch > Struct Template Reference	222
6.137.1 Detailed Description	223
6.137.2 Constructor & Destructor Documentation	223
6.137.2.1 UnorderedEqualsMatcher()	223
6.137.3 Member Function Documentation	224
6.137.3.1 describe()	224
6.137.3.2 match()	224
6.138 Catch::UseColour Struct Reference	224
6.138.1 Detailed Description	224
6.138.2 Member Enumeration Documentation	224
6.138.2.1 YesOrNo	224
6.139 Var Class Reference	225
6.139.1 Detailed Description	225
6.139.2 Constructor & Destructor Documentation	225
6.139.2.1 Var()	225
6.139.3 Member Function Documentation	226
6.139.3.1 equals()	226
6.139.3.2 hasVariable()	226
6.139.3.3 interp()	226
6.139.3.4 pretty_print()	227
6.139.3.5 print()	227
6.139.3.6 subst()	227
6.139.4 Member Data Documentation	228
6.139.4.1 var	228
6.140 Catch::detail::void_type<... > Struct Template Reference	228
6.140.1 Detailed Description	228
6.140.2 Member Typedef Documentation	229
6.140.2.1 type	229

6.141	Catch::WaitForKeypress Struct Reference	229
6.141.1	Detailed Description	229
6.141.2	Member Enumeration Documentation	229
6.141.2.1	When	229
6.142	Catch::WarnAbout Struct Reference	230
6.142.1	Detailed Description	230
6.142.2	Member Enumeration Documentation	230
6.142.2.1	What	230
6.143	Catch::Matchers::Floating::WithinAbsMatcher Struct Reference	230
6.143.1	Detailed Description	231
6.143.2	Constructor & Destructor Documentation	231
6.143.2.1	WithinAbsMatcher()	231
6.143.3	Member Function Documentation	232
6.143.3.1	describe()	232
6.143.3.2	match()	232
6.144	Catch::Matchers::Floating::WithinRelMatcher Struct Reference	232
6.144.1	Detailed Description	233
6.144.2	Constructor & Destructor Documentation	233
6.144.2.1	WithinRelMatcher()	233
6.144.3	Member Function Documentation	233
6.144.3.1	describe()	233
6.144.3.2	match()	233
6.145	Catch::Matchers::Floating::WithinUlpMatcher Struct Reference	234
6.145.1	Detailed Description	235
6.145.2	Constructor & Destructor Documentation	235
6.145.2.1	WithinUlpMatcher()	235
6.145.3	Member Function Documentation	235
6.145.3.1	describe()	235
6.145.3.2	match()	235
7	File Documentation	237
7.1	/Users/samanthapope/msdscriptRepo/msdScript/catch.h File Reference	237
7.1.1	Macro Definition Documentation	251
7.1.1.1	AND_GIVEN	251
7.1.1.2	AND_THEN	251
7.1.1.3	AND_WHEN	251
7.1.1.4	ANON_TEST_CASE	251
7.1.1.5	CAPTURE	251
7.1.1.6	CATCH_CATCH_ALL	252
7.1.1.7	CATCH_CATCH_ANON	252
7.1.1.8	CATCH_CONFIG_COUNTER	252
7.1.1.9	CATCH_CONFIG_CPP11_TO_STRING	252

7.1.1.10 CATCH_CONFIG_DISABLE_EXCEPTIONS	252
7.1.1.11 CATCH_CONFIG_GLOBAL_NEXTAFTER	252
7.1.1.12 CATCH_CONFIG_POSIX_SIGNALS	252
7.1.1.13 CATCH_CONFIG_WCHAR	253
7.1.1.14 CATCH_DEFER	253
7.1.1.15 CATCH_EMPTY	253
7.1.1.16 CATCH_ENFORCE	253
7.1.1.17 CATCH_ERROR	253
7.1.1.18 CATCH_INTERNAL_CONFIG_COUNTER	253
7.1.1.19 CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER	253
7.1.1.20 CATCH_INTERNAL_CONFIG_POSIX_SIGNALS	254
7.1.1.21 CATCH_INTERNAL_ERROR	254
7.1.1.22 CATCH_INTERNAL_IGNORE_BUT_WARN	254
7.1.1.23 CATCH_INTERNAL_LINEINFO	254
7.1.1.24 CATCH_INTERNAL_START_WARNINGS_SUPPRESSION	254
7.1.1.25 CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION	254
7.1.1.26 CATCH_INTERNAL_STRINGIFY	254
7.1.1.27 CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS	255
7.1.1.28 CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS	255
7.1.1.29 CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS	255
7.1.1.30 CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS	255
7.1.1.31 CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS	255
7.1.1.32 CATCH_MAKE_MSG	255
7.1.1.33 CATCH_REC_END	255
7.1.1.34 CATCH_REC_GET_END	256
7.1.1.35 CATCH_REC_GET_END1	256
7.1.1.36 CATCH_REC_GET_END2	256
7.1.1.37 CATCH_REC_LIST	256
7.1.1.38 CATCH_REC_LIST0	256
7.1.1.39 CATCH_REC_LIST0_UD	256
7.1.1.40 CATCH_REC_LIST1	257
7.1.1.41 CATCH_REC_LIST1_UD	257
7.1.1.42 CATCH_REC_LIST2	257
7.1.1.43 CATCH_REC_LIST2_UD	257
7.1.1.44 CATCH_REC_LIST_UD	257
7.1.1.45 CATCH_REC_NEXT	258
7.1.1.46 CATCH_REC_NEXT0	258
7.1.1.47 CATCH_REC_NEXT1	258
7.1.1.48 CATCH_REC_OUT	258
7.1.1.49 CATCH_RECURSE	258
7.1.1.50 CATCH_RECURSION_LEVEL0	258
7.1.1.51 CATCH_RECURSION_LEVEL1	259

7.1.1.52 CATCH_RECURSION_LEVEL2	259
7.1.1.53 CATCH_RECURSION_LEVEL3	259
7.1.1.54 CATCH_RECURSION_LEVEL4	259
7.1.1.55 CATCH_RECURSION_LEVEL5	259
7.1.1.56 CATCH_REGISTER_ENUM	259
7.1.1.57 CATCH_REGISTER_TAG_ALIAS	260
7.1.1.58 CATCH_RUNTIME_ERROR	260
7.1.1.59 CATCH_TRANSLATE_EXCEPTION	260
7.1.1.60 CATCH_TRY	260
7.1.1.61 CATCH_VERSION_MAJOR	260
7.1.1.62 CATCH_VERSION_MINOR	260
7.1.1.63 CATCH_VERSION_PATCH	261
7.1.1.64 CHECK	261
7.1.1.65 CHECK_FALSE	261
7.1.1.66 CHECK_NOFAIL	261
7.1.1.67 CHECK_NOTHROW	261
7.1.1.68 CHECK_THAT	261
7.1.1.69 CHECK_THROWS	262
7.1.1.70 CHECK_THROWS_AS	262
7.1.1.71 CHECK_THROWS_MATCHES	262
7.1.1.72 CHECK_THROWS_WITH	262
7.1.1.73 CHECKED_ELSE	262
7.1.1.74 CHECKED_IF	262
7.1.1.75 DYNAMIC_SECTION	263
7.1.1.76 FAIL	263
7.1.1.77 FAIL_CHECK	263
7.1.1.78 GENERATE	263
7.1.1.79 GENERATE_COPY	263
7.1.1.80 GENERATE_REF	264
7.1.1.81 GIVEN	264
7.1.1.82 INFO	264
7.1.1.83 INTERNAL_CATCH_CAPTURE	264
7.1.1.84 INTERNAL_CATCH_CATCH	264
7.1.1.85 INTERNAL_CATCH_DECLARE_SIG_TEST	265
7.1.1.86 INTERNAL_CATCH_DECLARE_SIG_TEST0	265
7.1.1.87 INTERNAL_CATCH_DECLARE_SIG_TEST1	265
7.1.1.88 INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD	265
7.1.1.89 INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0	265
7.1.1.90 INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1	266
7.1.1.91 INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X	266
7.1.1.92 INTERNAL_CATCH_DECLARE_SIG_TEST_X	266
7.1.1.93 INTERNAL_CATCH_DEF	266

7.1.1.94 INTERNAL_CATCH_DEFINE_SIG_TEST	267
7.1.1.95 INTERNAL_CATCH_DEFINE_SIG_TEST0	267
7.1.1.96 INTERNAL_CATCH_DEFINE_SIG_TEST1	267
7.1.1.97 INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD	267
7.1.1.98 INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0	267
7.1.1.99 INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1	268
7.1.1.100 INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X	268
7.1.1.101 INTERNAL_CATCH_DEFINE_SIG_TEST_X	268
7.1.1.102 INTERNAL_CATCH_DYNAMIC_SECTION	268
7.1.1.103 INTERNAL_CATCH_ELSE	269
7.1.1.104 INTERNAL_CATCH_EXPAND1	269
7.1.1.105 INTERNAL_CATCH_EXPAND2	269
7.1.1.106 INTERNAL_CATCH_IF	269
7.1.1.107 INTERNAL_CATCH_INFO	269
7.1.1.108 INTERNAL_CATCH_MAKE_NAMESPACE	270
7.1.1.109 INTERNAL_CATCH_MAKE_NAMESPACE2	270
7.1.1.110 INTERNAL_CATCH_MAKE_TYPE_LIST	270
7.1.1.111 INTERNAL_CATCH_MAKE_TYPE_LIST2	270
7.1.1.112 INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES	270
7.1.1.113 INTERNAL_CATCH_METHOD_AS_TEST_CASE	270
7.1.1.114 INTERNAL_CATCH_MSG	271
7.1.1.115 INTERNAL_CATCH_NO_THROW	271
7.1.1.116 INTERNAL_CATCH_NOINTERNAL_CATCH_DEF	271
7.1.1.117 INTERNAL_CATCH_NTTP_0	271
7.1.1.118 INTERNAL_CATCH_NTTP_1	272
7.1.1.119 INTERNAL_CATCH_NTTP_GEN	272
7.1.1.120 INTERNAL_CATCH_NTTP_REG_GEN	272
7.1.1.121 INTERNAL_CATCH_NTTP_REG_METHOD_GEN	272
7.1.1.122 INTERNAL_CATCH_NTTP_REGISTER	273
7.1.1.123 INTERNAL_CATCH_NTTP_REGISTER0	273
7.1.1.124 INTERNAL_CATCH_NTTP_REGISTER_METHOD	273
7.1.1.125 INTERNAL_CATCH_NTTP_REGISTER_METHOD0	273
7.1.1.126 INTERNAL_CATCH_REACT	274
7.1.1.127 INTERNAL_CATCH_REGISTER_ENUM	274
7.1.1.128 INTERNAL_CATCH_REGISTER_TESTCASE	274
7.1.1.129 INTERNAL_CATCH_REMOVE_PARENS	274
7.1.1.130 INTERNAL_CATCH_REMOVE_PARENS_10_ARG	275
7.1.1.131 INTERNAL_CATCH_REMOVE_PARENS_11_ARG	275
7.1.1.132 INTERNAL_CATCH_REMOVE_PARENS_1_ARG	275
7.1.1.133 INTERNAL_CATCH_REMOVE_PARENS_2_ARG	275
7.1.1.134 INTERNAL_CATCH_REMOVE_PARENS_3_ARG	276
7.1.1.135 INTERNAL_CATCH_REMOVE_PARENS_4_ARG	276

7.1.1.136 INTERNAL_CATCH_REMOVE_PARENS_5_ARG	276
7.1.1.137 INTERNAL_CATCH_REMOVE_PARENS_6_ARG	276
7.1.1.138 INTERNAL_CATCH_REMOVE_PARENS_7_ARG	277
7.1.1.139 INTERNAL_CATCH_REMOVE_PARENS_8_ARG	277
7.1.1.140 INTERNAL_CATCH_REMOVE_PARENS_9_ARG	277
7.1.1.141 INTERNAL_CATCH_REMOVE_PARENS_GEN	277
7.1.1.142 INTERNAL_CATCH_SECTION	278
7.1.1.143 INTERNAL_CATCH_STRINGIZE	278
7.1.1.144 INTERNAL_CATCH_STRINGIZE2	278
7.1.1.145 INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS	278
7.1.1.146 INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE	278
7.1.1.147 INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_2	279
7.1.1.148 INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD	279
7.1.1.149 INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_2	279
7.1.1.150 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE	280
7.1.1.151 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2	280
7.1.1.152 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD	280
7.1.1.153 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2	281
7.1.1.154 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG	281
7.1.1.155 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG	281
7.1.1.156 INTERNAL_CATCH_TEMPLATE_TEST_CASE	281
7.1.1.157 INTERNAL_CATCH_TEMPLATE_TEST_CASE_2	282
7.1.1.158 INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD	282
7.1.1.159 INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2	282
7.1.1.160 INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG	283
7.1.1.161 INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG	283
7.1.1.162 INTERNAL_CATCH_TEST	284
7.1.1.163 INTERNAL_CATCH_TEST_CASE_METHOD	284
7.1.1.164 INTERNAL_CATCH_TEST_CASE_METHOD2	284
7.1.1.165 INTERNAL_CATCH_TESTCASE	284
7.1.1.166 INTERNAL_CATCH_TESTCASE2	285
7.1.1.167 INTERNAL_CATCH_THROWS	285
7.1.1.168 INTERNAL_CATCH_THROWS_AS	285
7.1.1.169 INTERNAL_CATCH_THROWS_MATCHES	286
7.1.1.170 INTERNAL_CATCH_THROWS_STR_MATCHES	286
7.1.1.171 INTERNAL_CATCH_TRANSLATE_EXCEPTION	286
7.1.1.172 INTERNAL_CATCH_TRANSLATE_EXCEPTION2	287
7.1.1.173 INTERNAL_CATCH_TRY	287
7.1.1.174 INTERNAL_CATCH_TYPE_GEN	287
7.1.1.175 INTERNAL_CATCH_UNIQUE_NAME	287
7.1.1.176 INTERNAL_CATCH_UNIQUE_NAME_LINE	287
7.1.1.177 INTERNAL_CATCH_UNIQUE_NAME_LINE2	287

7.1.1.178 INTERNAL_CATCH_UNSCOPED_INFO	288
7.1.1.179 INTERNAL_CATCH_VA_NARGS_IMPL	288
7.1.1.180 INTERNAL_CHECK_THAT	288
7.1.1.181 METHOD_AS_TEST_CASE	288
7.1.1.182 REGISTER_TEST_CASE	289
7.1.1.183 REQUIRE	289
7.1.1.184 REQUIRE_FALSE	289
7.1.1.185 REQUIRE_NOTHROW	289
7.1.1.186 REQUIRE_THAT	289
7.1.1.187 REQUIRE_THROWS	289
7.1.1.188 REQUIRE_THROWS_AS	290
7.1.1.189 REQUIRE_THROWS_MATCHES	290
7.1.1.190 REQUIRE_THROWS_WITH	290
7.1.1.191 SCENARIO	290
7.1.1.192 SCENARIO_METHOD	290
7.1.1.193 SECTION	290
7.1.1.194 STATIC_REQUIRE	291
7.1.1.195 STATIC_REQUIRE_FALSE	291
7.1.1.196 SUCCEED	291
7.1.1.197 TEMPLATE_LIST_TEST_CASE	291
7.1.1.198 TEMPLATE_LIST_TEST_CASE_METHOD	291
7.1.1.199 TEMPLATE_PRODUCT_TEST_CASE	291
7.1.1.200 TEMPLATE_PRODUCT_TEST_CASE_METHOD	292
7.1.1.201 TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG	292
7.1.1.202 TEMPLATE_PRODUCT_TEST_CASE_SIG	292
7.1.1.203 TEMPLATE_TEST_CASE	292
7.1.1.204 TEMPLATE_TEST_CASE_METHOD	292
7.1.1.205 TEMPLATE_TEST_CASE_METHOD_SIG	292
7.1.1.206 TEMPLATE_TEST_CASE_SIG	293
7.1.1.207 TEST_CASE	293
7.1.1.208 TEST_CASE_METHOD	293
7.1.1.209 THEN	293
7.1.1.210 UNSCOPED_INFO	293
7.1.1.211 WARN	293
7.1.1.212 WHEN	293
7.1.2 Function Documentation	294
7.1.2.1 operator""_catch_sr()	294
7.1.2.2 operator<<()	294
7.2 catch.h	294
7.3 /Users/samanthapope/msdscriptRepo/msdScript/cmake-build-debug/CMakeFiles/3.27.8/Compiler↵ IdC/CMakeCCompilerId.c File Reference	724
7.3.1 Macro Definition Documentation	724

7.3.1.1 <code>__has_include</code>	724
7.3.1.2 <code>ARCHITECTURE_ID</code>	724
7.3.1.3 <code>C_VERSION</code>	724
7.3.1.4 <code>COMPILER_ID</code>	725
7.3.1.5 <code>DEC</code>	725
7.3.1.6 <code>HEX</code>	725
7.3.1.7 <code>PLATFORM_ID</code>	725
7.3.1.8 <code>STRINGIFY</code>	725
7.3.1.9 <code>STRINGIFY_HELPER</code>	726
7.3.2 Function Documentation	726
7.3.2.1 <code>main()</code>	726
7.3.3 Variable Documentation	726
7.3.3.1 <code>info_arch</code>	726
7.3.3.2 <code>info_compiler</code>	726
7.3.3.3 <code>info_language_extensions_default</code>	726
7.3.3.4 <code>info_language_standard_default</code>	727
7.3.3.5 <code>info_platform</code>	727
7.4 <code>CMakeCCompilerId.c</code>	727
7.5 /Users/samanthapope/msdscripRepo/msdScript/cmake-build-debug/CMakeFiles/3.27.8/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference	737
7.5.1 Macro Definition Documentation	738
7.5.1.1 <code>__has_include</code>	738
7.5.1.2 <code>ARCHITECTURE_ID</code>	738
7.5.1.3 <code>COMPILER_ID</code>	738
7.5.1.4 <code>CXX_STD</code>	738
7.5.1.5 <code>DEC</code>	738
7.5.1.6 <code>HEX</code>	738
7.5.1.7 <code>PLATFORM_ID</code>	739
7.5.1.8 <code>STRINGIFY</code>	739
7.5.1.9 <code>STRINGIFY_HELPER</code>	739
7.5.2 Function Documentation	739
7.5.2.1 <code>main()</code>	739
7.5.3 Variable Documentation	739
7.5.3.1 <code>info_arch</code>	739
7.5.3.2 <code>info_compiler</code>	739
7.5.3.3 <code>info_language_extensions_default</code>	740
7.5.3.4 <code>info_language_standard_default</code>	740
7.5.3.5 <code>info_platform</code>	740
7.6 <code>CMakeCXXCompilerId.cpp</code>	740
7.7 /Users/samanthapope/msdscripRepo/msdScript/cmdline.cpp File Reference	750
7.7.1 Macro Definition Documentation	751
7.7.1.1 <code>CATCH_CONFIG_RUNNER</code>	751

7.7.2 Function Documentation	751
7.7.2.1 useArgs()	751
7.8 cmdline.cpp	751
7.9 /Users/samanthapope/msdscriptRepo/msdScript/cmdline.h File Reference	752
7.9.1 Function Documentation	752
7.9.1.1 useArgs()	752
7.10 cmdline.h	752
7.11 /Users/samanthapope/msdscriptRepo/msdScript/Expr.cpp File Reference	752
7.12 Expr.cpp	752
7.13 /Users/samanthapope/msdscriptRepo/msdScript/Expr.h File Reference	754
7.13.1 Enumeration Type Documentation	755
7.13.1.1 precedence_t	755
7.14 Expr.h	755
7.15 /Users/samanthapope/msdscriptRepo/msdScript/ExprTest.cpp File Reference	756
7.15.1 Function Documentation	757
7.15.1.1 TEST_CASE() [1/11]	757
7.15.1.2 TEST_CASE() [2/11]	757
7.15.1.3 TEST_CASE() [3/11]	757
7.15.1.4 TEST_CASE() [4/11]	757
7.15.1.5 TEST_CASE() [5/11]	757
7.15.1.6 TEST_CASE() [6/11]	758
7.15.1.7 TEST_CASE() [7/11]	758
7.15.1.8 TEST_CASE() [8/11]	758
7.15.1.9 TEST_CASE() [9/11]	758
7.15.1.10 TEST_CASE() [10/11]	758
7.15.1.11 TEST_CASE() [11/11]	758
7.16 ExprTest.cpp	759
7.17 /Users/samanthapope/msdscriptRepo/msdScript/main.cpp File Reference	763
7.17.1 Function Documentation	764
7.17.1.1 main()	764
7.18 main.cpp	764

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Catch	13
Catch::Detail	27
Catch::detail	30
Catch::Generators	30
Catch::Generators::pf	35
Catch::literals	36
Catch::Matchers	36
Catch::Matchers::Exception	40
Catch::Matchers::Floating	40
Catch::Matchers::Generic	40
Catch::Matchers::Generic::Detail	41
Catch::Matchers::Impl	41
Catch::Matchers::StdString	41
Catch::Matchers::Vector	41
mpl_	41

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Catch::Detail::Approx	48
Catch::Generators::as< T >	55
Catch::AssertionHandler	56
Catch::AssertionInfo	58
Catch::AssertionReaction	59
Catch::Capturer	64
Catch::Matchers::StdString::CasedString	65
Catch::CaseSensitive	66
Catch_global_namespace_dummy	67
Catch::Counts	75
Catch::Decomposer	77
Catch::Detail::EnumInfo	79
std::exception	
Catch::GeneratorException	96
Catch::ExceptionTranslatorRegistrar	86
Expr	87
Add	43
Mult	147
Num	153
Var	225
Catch::ExprLhs< LhsT >	89
std::false_type	
Catch::always_false< T >	48
Catch::detail::is_range_impl< T, typename >	121
Catch::is_range< T >	120
Catch::Generators::GeneratorUntypedBase	99
Catch::Generators::IGenerator< std::vector< T > >	107
Catch::Generators::ChunkGenerator< T >	68
Catch::Generators::IGenerator< Float >	107
Catch::Generators::RandomFloatingGenerator< Float >	162
Catch::Generators::IGenerator< Integer >	107
Catch::Generators::RandomIntegerGenerator< Integer >	164
Catch::Generators::IGenerator< T >	107
Catch::Generators::FilterGenerator< T, Predicate >	93

Catch::Generators::FixedValuesGenerator< T >	94
Catch::Generators::Generators< T >	97
Catch::Generators::IteratorGenerator< T >	123
Catch::Generators::MapGenerator< T, U, Func >	130
Catch::Generators::RangeGenerator< T >	166
Catch::Generators::RepeatGenerator< T >	170
Catch::Generators::SingleValueGenerator< T >	182
Catch::Generators::TakeGenerator< T >	210
Catch::Generators::GeneratorWrapper< T >	100
Catch::Generators::GeneratorWrapper< U >	100
Catch::IContext	104
Catch::IMutableContext	109
Catch::IExceptionTranslator	105
Catch::IExceptionTranslatorRegistry	106
Catch::IGeneratorTracker	108
Catch::IMutableEnumValuesRegistry	111
Catch::IMutableRegistryHub	112
Catch::IRegistryHub	113
Catch::IResultCapture	115
Catch::IRunner	118
Catch::is_callable< T >	119
Catch::is_callable< Fun(Args...)>	119
Catch::is_callable_tester	119
Catch::Detail::IsStreamInsertable< T >	122
Catch::IStream	122
Catch::ITestCaseRegistry	125
Catch::ITestInvoker	126
Catch::TestInvokerAsMethod< C >	217
Catch::ITransientExpression	126
Catch::BinaryExpr< LhsT, RhsT >	61
Catch::MatchExpr< ArgT, MatcherT >	140
Catch::UnaryExpr< LhsT >	221
Catch::LazyExpression	128
Catch::Matchers::Impl::MatcherMethod< ObjectT >	137
Catch::Matchers::Impl::MatcherBase< std::exception >	136
Catch::Matchers::Impl::MatcherBase< double >	136
Catch::Matchers::Impl::MatcherBase< ArgT >	136
Catch::Matchers::Impl::MatchAllOf< ArgT >	132
Catch::Matchers::Impl::MatchAnyOf< ArgT >	134
Catch::Matchers::Impl::MatchNotOf< ArgT >	141
Catch::Matchers::Impl::MatcherBase< std::string >	136
Catch::Matchers::Impl::MatcherMethod< ArgT >	137
Catch::Matchers::Impl::MatcherMethod< double >	137
Catch::Matchers::Impl::MatcherMethod< std::exception >	137
Catch::Matchers::Impl::MatcherMethod< std::string >	137
Catch::Matchers::Impl::MatcherMethod< T >	137
Catch::Matchers::Impl::MatcherBase< std::vector< T, AllocMatch > >	136
Catch::Matchers::Impl::MatcherBase< std::vector< T, Alloc > >	136
Catch::Matchers::Impl::MatcherBase< T >	136
Catch::Matchers::Exception::ExceptionMessageMatcher	85
Catch::Matchers::Floating::WithinAbsMatcher	230
Catch::Matchers::Floating::WithinRelMatcher	232
Catch::Matchers::Floating::WithinUlpsMatcher	234
Catch::Matchers::Generic::PredicateMatcher< T >	160
Catch::Matchers::StdString::RegexMatcher	168
Catch::Matchers::StdString::StringMatcherBase	205
Catch::Matchers::StdString::ContainsMatcher	72

Catch::Matchers::StdString::EndsWithMatcher	78
Catch::Matchers::StdString::EqualsMatcher	81
Catch::Matchers::StdString::StartsWithMatcher	186
Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >	52
Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >	69
Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >	73
Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >	82
Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >	222
Catch::Matchers::Impl::MatcherUntypedBase	138
Catch::Matchers::Impl::MatcherBase< std::exception >	136
Catch::Matchers::Impl::MatcherBase< double >	136
Catch::Matchers::Impl::MatcherBase< ArgT >	136
Catch::Matchers::Impl::MatcherBase< std::string >	136
Catch::Matchers::Impl::MatcherBase< std::vector< T, AllocMatch > >	136
Catch::Matchers::Impl::MatcherBase< std::vector< T, Alloc > >	136
Catch::Matchers::Impl::MatcherBase< T >	136
Catch::MessageInfo	145
Catch::MessageStream	146
Catch::MessageBuilder	143
Catch::NameAndTags	151
Catch::NonCopyable	152
Catch::AutoReg	60
Catch::IConfig	101
Catch::ReusableStringStream	173
Catch::Section	176
Catch::Option< T >	156
Catch::pluralise	159
Catch::RegistrarForTagAliases	169
Catch::ResultDisposition	171
Catch::ResultWas	172
Catch::RunTests	174
Catch::ScopedMessage	175
Catch::SectionEndInfo	177
Catch::SectionInfo	178
Catch::ShowDurations	179
Catch::SimplePcg32	180
Catch::SourceLineInfo	184
Catch::StreamEndStop	188
Catch::StringMaker< T, typename >	188
Catch::StringMaker< bool >	189
Catch::StringMaker< Catch::Detail::Approx >	190
Catch::StringMaker< char * >	190
Catch::StringMaker< char >	191
Catch::StringMaker< char const * >	191
Catch::StringMaker< char[SZ]>	192
Catch::StringMaker< double >	192
Catch::StringMaker< float >	193
Catch::StringMaker< int >	194
Catch::StringMaker< long >	194
Catch::StringMaker< long long >	195
Catch::StringMaker< R C::* >	195
Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStream← Insertable< R >::value >::type >	196
Catch::StringMaker< signed char >	197
Catch::StringMaker< signed char[SZ]>	197
Catch::StringMaker< std::nullptr_t >	198
Catch::StringMaker< std::string >	199

Catch::StringMaker< std::wstring >	199
Catch::StringMaker< T * >	200
Catch::StringMaker< T[SZ]>	200
Catch::StringMaker< unsigned char >	201
Catch::StringMaker< unsigned char[SZ]>	201
Catch::StringMaker< unsigned int >	202
Catch::StringMaker< unsigned long >	203
Catch::StringMaker< unsigned long long >	203
Catch::StringMaker< wchar_t * >	204
Catch::StringMaker< wchar_t const * >	204
Catch::StringRef	207
Catch::TestCaseInfo	214
Catch::TestCase	212
Catch::TestFailureException	217
Catch::Timer	218
Catch::Totals	219
std::true_type	
Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >()))>::type >	121
Catch::true_given< typename >	221
Catch::UseColour	224
Catch::detail::void_type<... >	228
Catch::WaitForKeypress	229
Catch::WarnAbout	230

Chapter 3

Class Index

3.1 Class List

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

Add	43
Catch::always_false< T >	48
Catch::Detail::Approx	48
Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >	52
Catch::Generators::as< T >	55
Catch::AssertionHandler	56
Catch::AssertionInfo	58
Catch::AssertionReaction	59
Catch::AutoReg	60
Catch::BinaryExpr< LhsT, RhstT >	61
Catch::Capturer	64
Catch::Matchers::StdString::CasedString	65
Catch::CaseSensitive	66
Catch_global_namespace_dummy	67
Catch::Generators::ChunkGenerator< T >	68
Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >	69
Catch::Matchers::StdString::ContainsMatcher	72
Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >	73
Catch::Counts	75
Catch::Decomposer	77
Catch::Matchers::StdString::EndsWithMatcher	78
Catch::Detail::EnumInfo	79
Catch::Matchers::StdString::EqualsMatcher	81
Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >	82
Catch::Matchers::Exception::ExceptionMessageMatcher	85
Catch::ExceptionTranslatorRegistrar	86
Expr	87
Catch::ExprLhs< LhsT >	89
Catch::Generators::FilterGenerator< T, Predicate >	93
Catch::Generators::FixedValuesGenerator< T >	94
Catch::GeneratorException	96
Catch::Generators::Generators< T >	97
Catch::Generators::GeneratorUntypedBase	99
Catch::Generators::GeneratorWrapper< T >	100
Catch::IConfig	101

Catch::IContext	104
Catch::IExceptionTranslator	105
Catch::IExceptionTranslatorRegistry	106
Catch::Generators::IGenerator< T >	107
Catch::IGeneratorTracker	108
Catch::IMutableContext	109
Catch::IMutableEnumValuesRegistry	111
Catch::IMutableRegistryHub	112
Catch::IRegistryHub	113
Catch::IResultCapture	115
Catch::IRunner	118
Catch::is_callable< T >	119
Catch::is_callable< Fun(Args...)>	119
Catch::is_callable_tester	119
Catch::is_range< T >	120
Catch::detail::is_range_impl< T, typename >	121
Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >()))>::type >	121
Catch::Detail::IsStreamInsertable< T >	122
Catch::IStream	122
Catch::Generators::IteratorGenerator< T >	123
Catch::ITestCaseRegistry	125
Catch::ITestInvoker	126
Catch::ITransientExpression	126
Catch::LazyExpression	128
Catch::Generators::MapGenerator< T, U, Func >	130
Catch::Matchers::Impl::MatchAllOf< ArgT >	132
Catch::Matchers::Impl::MatchAnyOf< ArgT >	134
Catch::Matchers::Impl::MatcherBase< T >	136
Catch::Matchers::Impl::MatcherMethod< ObjectT >	137
Catch::Matchers::Impl::MatcherUntypedBase	138
Catch::MatchExpr< ArgT, MatcherT >	140
Catch::Matchers::Impl::MatchNotOf< ArgT >	141
Catch::MessageBuilder	143
Catch::MessageInfo	145
Catch::MessageStream	146
Mult	147
Catch::NameAndTags	151
Catch::NonCopyable	152
Num	153
Catch::Option< T >	156
Catch::pluralise	159
Catch::Matchers::Generic::PredicateMatcher< T >	160
Catch::Generators::RandomFloatingGenerator< Float >	162
Catch::Generators::RandomIntegerGenerator< Integer >	164
Catch::Generators::RangeGenerator< T >	166
Catch::Matchers::StdString::RegexMatcher	168
Catch::RegistrarForTagAliases	169
Catch::Generators::RepeatGenerator< T >	170
Catch::ResultDisposition	171
Catch::ResultWas	172
Catch::ReusableStringStream	173
Catch::RunTests	174
Catch::ScopedMessage	175
Catch::Section	176
Catch::SectionEndInfo	177
Catch::SectionInfo	178
Catch::ShowDurations	179
Catch::SimplePcg32	180

Catch::Generators::SingleValueGenerator< T >	182
Catch::SourceLineInfo	184
Catch::Matchers::StdString::StartsWithMatcher	186
Catch::StreamEndStop	188
Catch::StringMaker< T, typename >	188
Catch::StringMaker< bool >	189
Catch::StringMaker< Catch::Detail::Approx >	190
Catch::StringMaker< char * >	190
Catch::StringMaker< char >	191
Catch::StringMaker< char const * >	191
Catch::StringMaker< char[SZ]>	192
Catch::StringMaker< double >	192
Catch::StringMaker< float >	193
Catch::StringMaker< int >	194
Catch::StringMaker< long >	194
Catch::StringMaker< long long >	195
Catch::StringMaker< R C::* >	195
Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStreamInsertable< R >::value >::value > >	196
Catch::StringMaker< signed char >	197
Catch::StringMaker< signed char[SZ]>	197
Catch::StringMaker< std::nullptr_t >	198
Catch::StringMaker< std::string >	199
Catch::StringMaker< std::wstring >	199
Catch::StringMaker< T * >	200
Catch::StringMaker< T[SZ]>	200
Catch::StringMaker< unsigned char >	201
Catch::StringMaker< unsigned char[SZ]>	201
Catch::StringMaker< unsigned int >	202
Catch::StringMaker< unsigned long >	203
Catch::StringMaker< unsigned long long >	203
Catch::StringMaker< wchar_t * >	204
Catch::StringMaker< wchar_t const * >	204
Catch::Matchers::StdString::StringMatcherBase	205
Catch::StringRef	207
Catch::Generators::TakeGenerator< T >	210
Catch::TestCase	212
Catch::TestCaseInfo	214
Catch::TestFailureException	217
Catch::TestInvokerAsMethod< C >	217
Catch::Timer	218
Catch::Totals	219
Catch::true_given< typename >	221
Catch::UnaryExpr< LhsT >	221
Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >	222
Catch::UseColour	224
Var	225
Catch::detail::void_type<... >	228
Catch::WaitForKeypress	229
Catch::WarnAbout	230
Catch::Matchers::Floating::WithinAbsMatcher	230
Catch::Matchers::Floating::WithinRelMatcher	232
Catch::Matchers::Floating::WithinUlpMatcher	234

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

/Users/samanthapope/msdscriptRepo/msdScript/catch.h	237
/Users/samanthapope/msdscriptRepo/msdScript/cmdline.cpp	750
/Users/samanthapope/msdscriptRepo/msdScript/cmdline.h	752
/Users/samanthapope/msdscriptRepo/msdScript/Expr.cpp	752
/Users/samanthapope/msdscriptRepo/msdScript/Expr.h	754
/Users/samanthapope/msdscriptRepo/msdScript/ExprTest.cpp	756
/Users/samanthapope/msdscriptRepo/msdScript/main.cpp	763
/Users/samanthapope/msdscriptRepo/msdScript/cmake-build-debug/CMakeFiles/3.27.8/CompilerIdC↵ C/CMakeCCompilerId.c	724
/Users/samanthapope/msdscriptRepo/msdScript/cmake-build-debug/CMakeFiles/3.27.8/CompilerIdC↵ CXX/CMakeCXXCompilerId.cpp	737

Chapter 5

Namespace Documentation

5.1 Catch Namespace Reference

Namespaces

- namespace [Detail](#)
- namespace [detail](#)
- namespace [Generators](#)
- namespace [literals](#)
- namespace [Matchers](#)

Classes

- struct [always_false](#)
- class [AssertionHandler](#)
- struct [AssertionInfo](#)
- struct [AssertionReaction](#)
- struct [AutoReg](#)
- class [BinaryExpr](#)
- class [Capturer](#)
- struct [CaseSensitive](#)
- struct [Counts](#)
- struct [Decomposer](#)
- class [ExceptionTranslatorRegistrar](#)
- class [ExprLhs](#)
- class [GeneratorException](#)
- struct [IConfig](#)
- struct [IContext](#)
- struct [IExceptionTranslator](#)
- struct [IExceptionTranslatorRegistry](#)
- struct [IGeneratorTracker](#)
- struct [IMutableContext](#)
- struct [IMutableEnumValuesRegistry](#)
- struct [IMutableRegistryHub](#)
- struct [IRegistryHub](#)
- struct [IResultCapture](#)
- struct [IRunner](#)

- struct [is_callable](#)
- struct [is_callable< Fun\(Args...\)>](#)
- struct [is_callable_tester](#)
- struct [is_range](#)
- struct [IStream](#)
- struct [ITestCaseRegistry](#)
- struct [ITestInvoker](#)
- struct [ITransientExpression](#)
- class [LazyExpression](#)
- class [MatchExpr](#)
- struct [MessageBuilder](#)
- struct [MessageInfo](#)
- struct [MessageStream](#)
- struct [NameAndTags](#)
- class [NonCopyable](#)
- class [Option](#)
- struct [pluralise](#)
- struct [RegistrarForTagAliases](#)
- struct [ResultDisposition](#)
- struct [ResultWas](#)
- class [ReusableStringStream](#)
- struct [RunTests](#)
- class [ScopedMessage](#)
- class [Section](#)
- struct [SectionEndInfo](#)
- struct [SectionInfo](#)
- struct [ShowDurations](#)
- class [SimplePcg32](#)
- struct [SourceLineInfo](#)
- struct [StreamEndStop](#)
- struct [StringMaker](#)
- struct [StringMaker< bool >](#)
- struct [StringMaker< Catch::Detail::Approx >](#)
- struct [StringMaker< char * >](#)
- struct [StringMaker< char >](#)
- struct [StringMaker< char const * >](#)
- struct [StringMaker< char\[SZ\]>](#)
- struct [StringMaker< double >](#)
- struct [StringMaker< float >](#)
- struct [StringMaker< int >](#)
- struct [StringMaker< long >](#)
- struct [StringMaker< long long >](#)
- struct [StringMaker< R C::* >](#)
- struct [StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStreamInsertable< R >::value >](#)
- struct [StringMaker< signed char >](#)
- struct [StringMaker< signed char\[SZ\]>](#)
- struct [StringMaker< std::nullptr_t >](#)
- struct [StringMaker< std::string >](#)
- struct [StringMaker< std::wstring >](#)
- struct [StringMaker< T * >](#)
- struct [StringMaker< T\[SZ\]>](#)
- struct [StringMaker< unsigned char >](#)
- struct [StringMaker< unsigned char\[SZ\]>](#)
- struct [StringMaker< unsigned int >](#)
- struct [StringMaker< unsigned long >](#)

- struct [StringMaker< unsigned long long >](#)
- struct [StringMaker< wchar_t * >](#)
- struct [StringMaker< wchar_t const * >](#)
- class [StringRef](#)
- class [TestCase](#)
- struct [TestCaseInfo](#)
- struct [TestFailureException](#)
- class [TestInvokerAsMethod](#)
- class [Timer](#)
- struct [Totals](#)
- struct [true_given](#)
- class [UnaryExpr](#)
- struct [UseColour](#)
- struct [WaitForKeypress](#)
- struct [WarnAbout](#)

Typedefs

- [template<typename Func , typename... U>](#)
[using FunctionReturnType = typename](#) [std::remove_reference<typename](#) [std::remove_cv<typename](#) [std::result_of<Func\(U...\)>::type>::type>::type](#)
- [using IReporterFactoryPtr = std::shared_ptr<IReporterFactory>](#)
- [using exceptionTranslateFunction = std::string\(*\)\(\)](#)
- [using ExceptionTranslators = std::vector<std::unique_ptr<IExceptionTranslator const>>](#)
- [using StringMatcher = Matchers::Impl::MatcherBase<std::string>](#)
- [using IConfigPtr = std::shared_ptr<IConfig const>](#)

Enumerations

- [enum class Verbosity { Quiet = 0 , Normal , High }](#)

Functions

- [unsigned int rngSeed \(\)](#)
- [std::ostream & operator<< \(std::ostream &os, SourceLineInfo const &info\)](#)
- [template<typename T >](#)
[T const & operator+ \(T const &value, StreamEndStop\)](#)
- [bool isThrowSafe \(TestCase const &testCase, IConfig const &config\)](#)
- [bool matchTest \(TestCase const &testCase, TestSpec const &testSpec, IConfig const &config\)](#)
- [std::vector< TestCase > filterTests \(std::vector< TestCase > const &testCases, TestSpec const &testSpec, IConfig const &config\)](#)
- [std::vector< TestCase > const & getAllTestCasesSorted \(IConfig const &config\)](#)
- [auto operator+= \(std::string &lhs, StringRef const &sr\) -> std::string &](#)
- [auto operator<< \(std::ostream &os, StringRef const &sr\) -> std::ostream &](#)
- [constexpr auto operator""_sr \(char const *rawChars, std::size_t size\) noexcept -> StringRef](#)
- [auto makeTestInvoker \(void\(*testAsFunction\)\(\)\) noexcept -> ITestInvoker *](#)
- [template<typename C >](#)
[auto makeTestInvoker \(void\(C::*testAsMethod\)\(\)\) noexcept -> ITestInvoker *](#)
- [bool isOk \(ResultWas::OfType resultType\)](#)
- [bool isJustInfo \(int flags\)](#)
- [ResultDisposition::Flags operator| \(ResultDisposition::Flags lhs, ResultDisposition::Flags rhs\)](#)
- [bool shouldContinueOnFailure \(int flags\)](#)

- [bool isFalseTest \(int flags\)](#)
- [bool shouldSuppressFailure \(int flags\)](#)
- [std::ostream & cout \(\)](#)
- [std::ostream & cerr \(\)](#)
- [std::ostream & clog \(\)](#)
- [auto makeStream \(StringRef const &filename\) -> IStream const *](#)
- [template<typename Range >
std::string rangeToString \(Range const &range\)](#)
- [template<typename Allocator >
std::string rangeToString \(std::vector< bool, Allocator > const &v\)](#)
- [void formatReconstructedExpression \(std::ostream &os, std::string const &lhs, StringRef op, std::string const &rhs\)](#)
- [template<typename LhsT, typename RhsT >
auto compareEqual \(LhsT const &lhs, RhsT const &rhs\) -> bool](#)
- [template<typename T >
auto compareEqual \(T *const &lhs, int rhs\) -> bool](#)
- [template<typename T >
auto compareEqual \(T *const &lhs, long rhs\) -> bool](#)
- [template<typename T >
auto compareEqual \(int lhs, T *const &rhs\) -> bool](#)
- [template<typename T >
auto compareEqual \(long lhs, T *const &rhs\) -> bool](#)
- [template<typename LhsT, typename RhsT >
auto compareNotEqual \(LhsT const &lhs, RhsT &&rhs\) -> bool](#)
- [template<typename T >
auto compareNotEqual \(T *const &lhs, int rhs\) -> bool](#)
- [template<typename T >
auto compareNotEqual \(T *const &lhs, long rhs\) -> bool](#)
- [template<typename T >
auto compareNotEqual \(int lhs, T *const &rhs\) -> bool](#)
- [template<typename T >
auto compareNotEqual \(long lhs, T *const &rhs\) -> bool](#)
- [void handleExpression \(ITransientExpression const &expr\)](#)
- [template<typename T >
void handleExpression \(ExprLhs< T > const &expr\)](#)
- [IResultCapture & getResultCapture \(\)](#)
- [void handleExceptionMatchExpr \(AssertionHandler &handler, std::string const &str, StringRef const &matcherString\)](#)
- [auto getCurrentNanosecondsSinceEpoch \(\) -> uint64_t](#)
- [auto getEstimatedClockResolution \(\) -> uint64_t](#)
- [IRegistryHub const & getRegistryHub \(\)](#)
- [IMutableRegistryHub & getMutableRegistryHub \(\)](#)
- [void cleanUp \(\)](#)
- [std::string translateActiveException \(\)](#)
- [bool startsWith \(std::string const &s, std::string const &prefix\)](#)
- [bool startsWith \(std::string const &s, char prefix\)](#)
- [bool endsWith \(std::string const &s, std::string const &suffix\)](#)
- [bool endsWith \(std::string const &s, char suffix\)](#)
- [bool contains \(std::string const &s, std::string const &infix\)](#)
- [void toLowerInPlace \(std::string &s\)](#)
- [std::string toLower \(std::string const &s\)](#)
- [std::string trim \(std::string const &str\)](#)
Returns a new string without whitespace at the start/end.
- [StringRef trim \(StringRef ref\)](#)
Returns a substring of the original ref without whitespace. Beware lifetimes!

- `std::vector< StringRef > splitStringRef (StringRef str, char delimiter)`
- `bool replaceInPlace (std::string &str, std::string const &replaceThis, std::string const &withThis)`
- `void handleExceptionMatchExpr (AssertionHandler &handler, StringMatcher const &matcher, StringRef const &matcherString)`
- `template<typename ArgT, typename MatcherT >
auto makeMatchExpr (ArgT const &arg, MatcherT const &matcher, StringRef const &matcherString) ->
MatchExpr< ArgT, MatcherT >`
- `void throw_exception (std::exception const &e)`
- `void throw_logic_error (std::string const &msg)`
- `void throw_domain_error (std::string const &msg)`
- `void throw_runtime_error (std::string const &msg)`
- `IMutableContext & getCurrentMutableContext ()`
- `IContext & getCurrentContext ()`
- `void cleanUpContext ()`
- `SimplePcg32 & rng ()`
- `TestCase makeTestCase (ITestInvoker *testCase, std::string const &className, NameAndTags const &nameAndTags, SourceLineInfo const &lineInfo)`

5.1.1 Typedef Documentation

5.1.1.1 exceptionTranslateFunction

```
using Catch::exceptionTranslateFunction = std::string(*)()
```

Definition at line 3005 of file `catch.h`.

5.1.1.2 ExceptionTranslators

```
using Catch::ExceptionTranslators = std::vector<std::unique_ptr<IExceptionTranslator const>>
```

Definition at line 3008 of file `catch.h`.

5.1.1.3 FunctionReturnType

```
template<typename Func, typename... U>  
using Catch::FunctionReturnType = typename std::remove_reference<typename std::remove_cv<typename  
std::result_of<Func(U...)>::type>::type>::type
```

Definition at line 949 of file `catch.h`.

5.1.1.4 IConfigPtr

```
typedef std::shared_ptr< IConfig const > Catch::IConfigPtr = std::shared_ptr<IConfig const>
```

Definition at line 4356 of file `catch.h`.

5.1.1.5 IReporterFactoryPtr

```
using Catch::IReporterFactoryPtr = std::shared_ptr<IReporterFactory>
```

Definition at line 2963 of file [catch.h](#).

5.1.1.6 StringMatcher

```
using Catch::StringMatcher = Matchers::Impl::MatcherBase<std::string>
```

Definition at line 3792 of file [catch.h](#).

5.1.2 Enumeration Type Documentation

5.1.2.1 Verbosity

```
enum class Catch::Verbosity [strong]
```

Enumerator

Quiet	
Normal	
High	

Definition at line 4476 of file [catch.h](#).

5.1.3 Function Documentation

5.1.3.1 cerr()

```
std::ostream & Catch::cerr ( )
```

5.1.3.2 cleanUp()

```
void Catch::cleanUp ( )
```

5.1.3.3 cleanUpContext()

```
void Catch::cleanUpContext ( )
```

5.1.3.4 clog()

```
std::ostream & Catch::clog ( )
```

5.1.3.5 compareEqual() [1/5]

```
template<typename T >
auto Catch::compareEqual (
    int lhs,
    T *const & rhs ) -> bool
```

Definition at line 2322 of file [catch.h](#).

5.1.3.6 compareEqual() [2/5]

```
template<typename LhsT , typename RhsT >
auto Catch::compareEqual (
    LhsT const & lhs,
    RhsT const & rhs ) -> bool
```

Definition at line 2316 of file [catch.h](#).

5.1.3.7 compareEqual() [3/5]

```
template<typename T >
auto Catch::compareEqual (
    long lhs,
    T *const & rhs ) -> bool
```

Definition at line 2324 of file [catch.h](#).

5.1.3.8 compareEqual() [4/5]

```
template<typename T >
auto Catch::compareEqual (
    T *const & lhs,
    int rhs ) -> bool
```

Definition at line 2318 of file [catch.h](#).

5.1.3.9 compareEqual() [5/5]

```
template<typename T >
auto Catch::compareEqual (
    T *const & lhs,
    long rhs ) -> bool
```

Definition at line 2320 of file [catch.h](#).

5.1.3.10 compareNotEqual() [1/5]

```
template<typename T >
auto Catch::compareNotEqual (
    int lhs,
    T *const & rhs ) -> bool
```

Definition at line 2333 of file [catch.h](#).

5.1.3.11 `compareNotEqual()` [2/5]

```
template<typename LhsT , typename RhsT >  
auto Catch::compareNotEqual (   
    LhsT const & lhs,  
    RhsT && rhs ) -> bool
```

Definition at line 2327 of file [catch.h](#).

5.1.3.12 `compareNotEqual()` [3/5]

```
template<typename T >  
auto Catch::compareNotEqual (   
    long lhs,  
    T *const & rhs ) -> bool
```

Definition at line 2335 of file [catch.h](#).

5.1.3.13 `compareNotEqual()` [4/5]

```
template<typename T >  
auto Catch::compareNotEqual (   
    T *const & lhs,  
    int rhs ) -> bool
```

Definition at line 2329 of file [catch.h](#).

5.1.3.14 `compareNotEqual()` [5/5]

```
template<typename T >  
auto Catch::compareNotEqual (   
    T *const & lhs,  
    long rhs ) -> bool
```

Definition at line 2331 of file [catch.h](#).

5.1.3.15 `contains()`

```
bool Catch::contains (   
    std::string const & s,  
    std::string const & infix )
```

5.1.3.16 `cout()`

```
std::ostream & Catch::cout ( )
```

5.1.3.17 endsWith() [1/2]

```
bool Catch::endsWith (
    std::string const & s,
    char suffix )
```

5.1.3.18 endsWith() [2/2]

```
bool Catch::endsWith (
    std::string const & s,
    std::string const & suffix )
```

5.1.3.19 filterTests()

```
std::vector< TestCase > Catch::filterTests (
    std::vector< TestCase > const & testCases,
    TestSpec const & testSpec,
    IConfig const & config )
```

5.1.3.20 formatReconstructedExpression()

```
void Catch::formatReconstructedExpression (
    std::ostream & os,
    std::string const & lhs,
    StringRef op,
    std::string const & rhs )
```

5.1.3.21 getAllTestCasesSorted()

```
std::vector< TestCase > const & Catch::getAllTestCasesSorted (
    IConfig const & config )
```

5.1.3.22 getCurrentContext()

```
IContext & Catch::getCurrentContext ( ) [inline]
```

Definition at line 4389 of file [catch.h](#).

5.1.3.23 getCurrentMutableContext()

```
IMutableContext & Catch::getCurrentMutableContext ( ) [inline]
```

Definition at line 4381 of file [catch.h](#).

5.1.3.24 getCurrentNanosecondsSinceEpoch()

```
auto Catch::getCurrentNanosecondsSinceEpoch ( ) -> uint64_t
```

5.1.3.25 getEstimatedClockResolution()

```
auto Catch::getEstimatedClockResolution ( ) -> uint64_t
```

5.1.3.26 getMutableRegistryHub()

```
IMutableRegistryHub & Catch::getMutableRegistryHub ( )
```

5.1.3.27 getRegistryHub()

```
IRegistryHub const & Catch::getRegistryHub ( )
```

5.1.3.28 getResultCapture()

```
IResultCapture & Catch::getResultCapture ( )
```

5.1.3.29 handleExceptionMatchExpr() [1/2]

```
void Catch::handleExceptionMatchExpr (
    AssertionHandler & handler,
    std::string const & str,
    StringRef const & matcherString )
```

5.1.3.30 handleExceptionMatchExpr() [2/2]

```
void Catch::handleExceptionMatchExpr (
    AssertionHandler & handler,
    StringMatcher const & matcher,
    StringRef const & matcherString )
```

5.1.3.31 handleExpression() [1/2]

```
template<typename T >
void Catch::handleExpression (
    ExprLhs< T > const & expr )
```

Definition at line 2410 of file [catch.h](#).

5.1.3.32 `handleExpression()` [2/2]

```
void Catch::handleExpression (
    ITransientExpression const & expr )
```

5.1.3.33 `isFalseTest()`

```
bool Catch::isFalseTest (
    int flags ) [inline]
```

Definition at line 1388 of file [catch.h](#).

5.1.3.34 `isJustInfo()`

```
bool Catch::isJustInfo (
    int flags )
```

5.1.3.35 `isOk()`

```
bool Catch::isOk (
    ResultWas::OfType resultType )
```

5.1.3.36 `isThrowSafe()`

```
bool Catch::isThrowSafe (
    TestCase const & testCase,
    IConfig const & config )
```

5.1.3.37 `makeMatchExpr()`

```
template<typename ArgT , typename MatcherT >
auto Catch::makeMatchExpr (
    ArgT const & arg,
    MatcherT const & matcher,
    StringRef const & matcherString ) -> MatchExpr<ArgT, MatcherT>
```

Definition at line 3797 of file [catch.h](#).

5.1.3.38 `makeStream()`

```
auto Catch::makeStream (
    StringRef const & filename ) -> IStream const *
```

5.1.3.39 makeTestCase()

```

TestCase Catch::makeTestCase (
    ITestInvoker * testCase,
    std::string const & className,
    NameAndTags const & nameAndTags,
    SourceLineInfo const & lineInfo )

```

5.1.3.40 makeTestInvoker() [1/2]

```

auto Catch::makeTestInvoker (
    void(*)() testAsFunction ) -> ITestInvoker * [noexcept]

```

5.1.3.41 makeTestInvoker() [2/2]

```

template<typename C >
auto Catch::makeTestInvoker (
    void(C::*)() testAsMethod ) -> ITestInvoker* [noexcept]

```

Definition at line 976 of file [catch.h](#).

5.1.3.42 matchTest()

```

bool Catch::matchTest (
    TestCase const & testCase,
    TestSpec const & testSpec,
    IConfig const & config )

```

5.1.3.43 operator""_sr()

```

constexpr auto Catch::operator""_sr (
    char const * rawChars,
    std::size_t size ) -> StringRef [constexpr], [noexcept]

```

Definition at line 680 of file [catch.h](#).

5.1.3.44 operator+()

```

template<typename T >
T const & Catch::operator+ (
    T const & value,
    StreamEndStop )

```

Definition at line 538 of file [catch.h](#).

5.1.3.45 operator+=()

```
auto Catch::operator+= (
    std::string & lhs,
    StringRef const & sr ) -> std::string &
```

5.1.3.46 operator<<() [1/2]

```
std::ostream & Catch::operator<< (
    std::ostream & os,
    SourceLineInfo const & info )
```

5.1.3.47 operator<<() [2/2]

```
auto Catch::operator<< (
    std::ostream & os,
    StringRef const & sr ) -> std::ostream &
```

5.1.3.48 operator" | ()

```
ResultDisposition::Flags Catch::operator| (
    ResultDisposition::Flags lhs,
    ResultDisposition::Flags rhs )
```

5.1.3.49 rangeToString() [1/2]

```
template<typename Range >
std::string Catch::rangeToString (
    Range const & range )
```

Definition at line 2018 of file [catch.h](#).

5.1.3.50 rangeToString() [2/2]

```
template<typename Allocator >
std::string Catch::rangeToString (
    std::vector< bool, Allocator > const & v )
```

Definition at line 2024 of file [catch.h](#).

5.1.3.51 replaceInPlace()

```
bool Catch::replaceInPlace (
    std::string & str,
    std::string const & replaceThis,
    std::string const & withThis )
```

5.1.3.52 rng()

```
SimplePcg32 & Catch::rng ( )
```

5.1.3.53 rngSeed()

```
unsigned int Catch::rngSeed ( )
```

5.1.3.54 shouldContinueOnFailure()

```
bool Catch::shouldContinueOnFailure (
    int flags )
```

5.1.3.55 shouldSuppressFailure()

```
bool Catch::shouldSuppressFailure (
    int flags )
```

5.1.3.56 splitStringRef()

```
std::vector< StringRef > Catch::splitStringRef (
    StringRef str,
    char delimiter )
```

5.1.3.57 startsWith() [1/2]

```
bool Catch::startsWith (
    std::string const & s,
    char prefix )
```

5.1.3.58 startsWith() [2/2]

```
bool Catch::startsWith (
    std::string const & s,
    std::string const & prefix )
```

5.1.3.59 throw_domain_error()

```
void Catch::throw_domain_error (
    std::string const & msg )
```

5.1.3.60 throw_exception()

```
void Catch::throw_exception (
    std::exception const & e )
```

5.1.3.61 throw_logic_error()

```
void Catch::throw_logic_error (
    std::string const & msg )
```

5.1.3.62 throw_runtime_error()

```
void Catch::throw_runtime_error (
    std::string const & msg )
```

5.1.3.63 toLower()

```
std::string Catch::toLower (
    std::string const & s )
```

5.1.3.64 toLowerInPlace()

```
void Catch::toLowerInPlace (
    std::string & s )
```

5.1.3.65 translateActiveException()

```
std::string Catch::translateActiveException ( )
```

5.1.3.66 trim() [1/2]

```
std::string Catch::trim (
    std::string const & str )
```

Returns a new string without whitespace at the start/end.

5.1.3.67 trim() [2/2]

```
StringRef Catch::trim (
    StringRef ref )
```

Returns a substring of the original ref without whitespace. Beware lifetimes!

5.2 Catch::Detail Namespace Reference

Classes

- class [Approx](#)
- struct [EnumInfo](#)
- class [IsStreamInsertable](#)

Functions

- `std::string rawMemoryToString (const void *object, std::size_t size)`
- `template<typename T >
std::string rawMemoryToString (const T &object)`
- `template<typename E >
std::string convertUnknownEnumToString (E e)`
- `template<typename T >
std::enable_if<!std::is_enum< T >::value &&!std::is_base_of< std::exception, T >::value, std::string >::type
convertUnstreamable (T const &)`
- `template<typename T >
std::enable_if<!std::is_enum< T >::value &&std::is_base_of< std::exception, T >::value, std::string >::type
convertUnstreamable (T const &ex)`
- `template<typename T >
std::enable_if< std::is_enum< T >::value, std::string >::type convertUnstreamable (T const &value)`
- `template<typename T >
std::string stringify (const T &e)`
- `template<typename InputIterator, typename Sentinel = InputIterator>
std::string rangeToString (InputIterator first, Sentinel last)`

Variables

- `const std::string unprintableString`

5.2.1 Function Documentation

5.2.1.1 convertUnknownEnumToString()

```
template<typename E >
std::string Catch::Detail::convertUnknownEnumToString (
    E e )
```

Definition at line 1649 of file [catch.h](#).

5.2.1.2 convertUnstreamable() [1/3]

```
template<typename T >
std::enable_if<!std::is_enum< T >::value &&!std::is_base_of< std::exception, T >::value,
std::string >::type Catch::Detail::convertUnstreamable (
    T const & )
```

Definition at line 1582 of file [catch.h](#).

5.2.1.3 convertUnstreamable() [2/3]

```
template<typename T >
std::enable_if<!std::is_enum< T >::value &&std::is_base_of< std::exception, T >::value, std::string >::type  
Catch::Detail::convertUnstreamable (
    T const & ex )
```

Definition at line 1588 of file [catch.h](#).

5.2.1.4 convertUnstreamable() [3/3]

```
template<typename T >
std::enable_if< std::is_enum< T >::value, std::string >::type Catch::Detail::convertUnstreamable
(
    T const & value )
```

Definition at line 1595 of file [catch.h](#).

5.2.1.5 rangeToString()

```
template<typename InputIterator , typename Sentinel = InputIterator>
std::string Catch::Detail::rangeToString (
    InputIterator first,
    Sentinel last )
```

Definition at line 1829 of file [catch.h](#).

5.2.1.6 rawMemoryToString() [1/2]

```
template<typename T >
std::string Catch::Detail::rawMemoryToString (
    const T & object )
```

Definition at line 1559 of file [catch.h](#).

5.2.1.7 rawMemoryToString() [2/2]

```
std::string Catch::Detail::rawMemoryToString (
    const void * object,
    std::size_t size )
```

5.2.1.8 stringify()

```
template<typename T >
std::string Catch::Detail::stringify (
    const T & e )
```

Definition at line 1644 of file [catch.h](#).

5.2.2 Variable Documentation

5.2.2.1 unprintableString

```
const std::string Catch::Detail::unprintableString [extern]
```

5.3 Catch::detail Namespace Reference

Classes

- struct [is_range_impl](#)
- struct [is_range_impl](#)< T, typename void_type< decltype(begin(std::declval< T >()))>::type >
- struct [void_type](#)

5.4 Catch::Generators Namespace Reference

Namespaces

- namespace [pf](#)

Classes

- struct [as](#)
- class [ChunkGenerator](#)
- class [FilterGenerator](#)
- class [FixedValuesGenerator](#)
- class [Generators](#)
- class [GeneratorUntypedBase](#)
- class [GeneratorWrapper](#)
- struct [IGenerator](#)
- class [IteratorGenerator](#)
- class [MapGenerator](#)
- class [RandomFloatingGenerator](#)
- class [RandomIntegerGenerator](#)
- class [RangeGenerator](#)
- class [RepeatGenerator](#)
- class [SingleValueGenerator](#)
- class [TakeGenerator](#)

Typedefs

- using [GeneratorBasePtr](#) = std::unique_ptr<[GeneratorUntypedBase](#)>

Functions

- `template<typename T >`
`GeneratorWrapper< T > value (T &&value)`
- `template<typename T >`
`GeneratorWrapper< T > values (std::initializer_list< T > values)`
- `template<typename... Ts>`
`GeneratorWrapper< std::tuple< Ts... > > table (std::initializer_list< std::tuple< typename std::decay< Ts >::type... > > tuples)`
- `template<typename T , typename... Gs>`
`auto makeGenerators (GeneratorWrapper< T > &&generator, Gs &&... moreGenerators) -> Generators< T >`
- `template<typename T >`
`auto makeGenerators (GeneratorWrapper< T > &&generator) -> Generators< T >`
- `template<typename T , typename... Gs>`
`auto makeGenerators (T &&val, Gs &&... moreGenerators) -> Generators< T >`
- `template<typename T , typename U , typename... Gs>`
`auto makeGenerators (as< T >, U &&val, Gs &&... moreGenerators) -> Generators< T >`
- `auto acquireGeneratorTracker (StringRef generatorName, SourceLineInfo const &lineInfo) -> IGeneratorTracker &`
- `template<typename L >`
`auto generate (StringRef generatorName, SourceLineInfo const &lineInfo, L const &generatorExpression) -> decltype(std::declval< decltype(generatorExpression())>().get())`
- `template<typename T >`
`GeneratorWrapper< T > take (size_t target, GeneratorWrapper< T > &&generator)`
- `template<typename T , typename Predicate >`
`GeneratorWrapper< T > filter (Predicate &&pred, GeneratorWrapper< T > &&generator)`
- `template<typename T >`
`GeneratorWrapper< T > repeat (size_t repeats, GeneratorWrapper< T > &&generator)`
- `template<typename Func , typename U , typename T = FunctionReturnType<Func, U>>`
`GeneratorWrapper< T > map (Func &&function, GeneratorWrapper< U > &&generator)`
- `template<typename T >`
`GeneratorWrapper< std::vector< T > > chunk (size_t size, GeneratorWrapper< T > &&generator)`
- `template<typename T >`
`std::enable_if< std::is_integral< T >::value &&!std::is_same< T, bool >::value, GeneratorWrapper< T > >::type random (T a, T b)`
- `template<typename T >`
`std::enable_if< std::is_floating_point< T >::value, GeneratorWrapper< T > >::type random (T a, T b)`
- `template<typename T >`
`GeneratorWrapper< T > range (T const &start, T const &end, T const &step)`
- `template<typename T >`
`GeneratorWrapper< T > range (T const &start, T const &end)`
- `template<typename InputIterator , typename InputSentinel , typename ResultType = typename std::iterator_traits<InputIterator>::value_type>`
`GeneratorWrapper< ResultType > from_range (InputIterator from, InputSentinel to)`
- `template<typename Container , typename ResultType = typename Container::value_type>`
`GeneratorWrapper< ResultType > from_range (Container const &cnt)`

5.4.1 Typedef Documentation

5.4.1.1 GeneratorBasePtr

```
using Catch::Generators::GeneratorBasePtr = std::unique_ptr<GeneratorUntypedBase>
```

Definition at line 3855 of file [catch.h](#).

5.4.2 Function Documentation

5.4.2.1 `acquireGeneratorTracker()`

```
auto Catch::Generators::acquireGeneratorTracker (
    StringRef generatorName,
    SourceLineInfo const & lineInfo ) -> IGeneratorTracker &
```

5.4.2.2 `chunk()`

```
template<typename T >
GeneratorWrapper< std::vector< T > > Catch::Generators::chunk (
    size_t size,
    GeneratorWrapper< T > && generator )
```

Definition at line 4333 of file [catch.h](#).

5.4.2.3 `filter()`

```
template<typename T , typename Predicate >
GeneratorWrapper< T > Catch::Generators::filter (
    Predicate && pred,
    GeneratorWrapper< T > && generator )
```

Definition at line 4195 of file [catch.h](#).

5.4.2.4 `from_range()` [1/2]

```
template<typename Container , typename ResultType = typename Container::value_type>
GeneratorWrapper< ResultType > Catch::Generators::from_range (
    Container const & cnt )
```

Definition at line 4746 of file [catch.h](#).

5.4.2.5 `from_range()` [2/2]

```
template<typename InputIterator , typename InputSentinel , typename ResultType = typename
std::iterator_traits<InputIterator>::value_type>
GeneratorWrapper< ResultType > Catch::Generators::from_range (
    InputIterator from,
    InputSentinel to )
```

Definition at line 4740 of file [catch.h](#).

5.4.2.6 generate()

```
template<typename L >
auto Catch::Generators::generate (
    StringRef generatorName,
    SourceLineInfo const & lineInfo,
    L const & generatorExpression ) -> decltype(std::declval<decltype(generatorExpression())>().get
```

Definition at line 4085 of file [catch.h](#).

5.4.2.7 makeGenerators() [1/4]

```
template<typename T , typename U , typename... Gs>
auto Catch::Generators::makeGenerators (
    as< T > ,
    U && val,
    Gs &&... moreGenerators ) -> Generators<T>
```

Definition at line 4075 of file [catch.h](#).

5.4.2.8 makeGenerators() [2/4]

```
template<typename T >
auto Catch::Generators::makeGenerators (
    GeneratorWrapper< T > && generator ) -> Generators<T>
```

Definition at line 4067 of file [catch.h](#).

5.4.2.9 makeGenerators() [3/4]

```
template<typename T , typename... Gs>
auto Catch::Generators::makeGenerators (
    GeneratorWrapper< T > && generator,
    Gs &&... moreGenerators ) -> Generators<T>
```

Definition at line 4063 of file [catch.h](#).

5.4.2.10 makeGenerators() [4/4]

```
template<typename T , typename... Gs>
auto Catch::Generators::makeGenerators (
    T && val,
    Gs &&... moreGenerators ) -> Generators<T>
```

Definition at line 4071 of file [catch.h](#).

5.4.2.11 map()

```
template<typename Func , typename U , typename T = FunctionReturnType<Func, U>>
GeneratorWrapper< T > Catch::Generators::map (
    Func && function,
    GeneratorWrapper< U > && generator )
```

Definition at line 4283 of file [catch.h](#).

5.4.2.12 random() [1/2]

```
template<typename T >
std::enable_if< std::is_integral< T >::value &&!std::is_same< T, bool >::value, GeneratorWrapper<
T > >::type Catch::Generators::random (
    T a,
    T b )
```

Definition at line 4651 of file [catch.h](#).

5.4.2.13 random() [2/2]

```
template<typename T >
std::enable_if< std::is_floating_point< T >::value, GeneratorWrapper< T > >::type Catch::↵
Generators::random (
    T a,
    T b )
```

Definition at line 4660 of file [catch.h](#).

5.4.2.14 range() [1/2]

```
template<typename T >
GeneratorWrapper< T > Catch::Generators::range (
    T const & start,
    T const & end )
```

Definition at line 4706 of file [catch.h](#).

5.4.2.15 range() [2/2]

```
template<typename T >
GeneratorWrapper< T > Catch::Generators::range (
    T const & start,
    T const & end,
    T const & step )
```

Definition at line 4700 of file [catch.h](#).

5.4.2.16 repeat()

```
template<typename T >
GeneratorWrapper< T > Catch::Generators::repeat (
    size_t repeats,
    GeneratorWrapper< T > && generator )
```

Definition at line 4251 of file [catch.h](#).

5.4.2.17 table()

```
template<typename... Ts>
GeneratorWrapper< std::tuple< Ts... > > Catch::Generators::table (
    std::initializer_list< std::tuple< typename std::decay< Ts >::type... > >
    tuples )
```

Definition at line 4054 of file [catch.h](#).

5.4.2.18 take()

```
template<typename T >
GeneratorWrapper< T > Catch::Generators::take (
    size_t target,
    GeneratorWrapper< T > && generator )
```

Definition at line 4151 of file [catch.h](#).

5.4.2.19 value()

```
template<typename T >
GeneratorWrapper< T > Catch::Generators::value (
    T && value )
```

Definition at line 4001 of file [catch.h](#).

5.4.2.20 values()

```
template<typename T >
GeneratorWrapper< T > Catch::Generators::values (
    std::initializer_list< T > values )
```

Definition at line 4005 of file [catch.h](#).

5.5 Catch::Generators::pf Namespace Reference

Functions

- template<typename T, typename... Args>
std::unique_ptr< T > [make_unique](#) (Args &&... args)

5.5.1 Function Documentation

5.5.1.1 `make_unique()`

```
template<typename T , typename... Args>
std::unique_ptr< T > Catch::Generators::pf::make_unique (
    Args &&... args )
```

Definition at line 3935 of file [catch.h](#).

5.6 `Catch::literals` Namespace Reference

Functions

- [Detail::Approx operator""_a](#) (long double val)
- [Detail::Approx operator""_a](#) (unsigned long long val)

5.6.1 Function Documentation

5.6.1.1 `operator""_a()` [1/2]

```
Detail::Approx Catch::literals::operator""_a (
    long double val )
```

5.6.1.2 `operator""_a()` [2/2]

```
Detail::Approx Catch::literals::operator""_a (
    unsigned long long val )
```

5.7 `Catch::Matchers` Namespace Reference

Namespaces

- namespace [Exception](#)
- namespace [Floating](#)
- namespace [Generic](#)
- namespace [Impl](#)
- namespace [StdString](#)
- namespace [Vector](#)

Functions

- [Exception::ExceptionMessageMatcher Message](#) (std::string const &message)
- [Floating::WithinUlpMatcher WithinULP](#) (double target, uint64_t maxUlpDiff)
- [Floating::WithinUlpMatcher WithinULP](#) (float target, uint64_t maxUlpDiff)
- [Floating::WithinAbsMatcher WithinAbs](#) (double target, double margin)
- [Floating::WithinRelMatcher WithinRel](#) (double target, double eps)
- [Floating::WithinRelMatcher WithinRel](#) (double target)
- [Floating::WithinRelMatcher WithinRel](#) (float target, float eps)
- [Floating::WithinRelMatcher WithinRel](#) (float target)
- [template<typename T > Generic::PredicateMatcher< T > Predicate](#) (std::function< bool(T const &)> const &predicate, std::string const &description="")
- [StdString::EqualsMatcher Equals](#) (std::string const &str, CaseSensitive::Choice caseSensitivity=CaseSensitive::Yes)
- [StdString::ContainsMatcher Contains](#) (std::string const &str, CaseSensitive::Choice caseSensitivity=CaseSensitive::Yes)
- [StdString::EndsWithMatcher EndsWith](#) (std::string const &str, CaseSensitive::Choice caseSensitivity=CaseSensitive::Yes)
- [StdString::StartsWithMatcher StartsWith](#) (std::string const &str, CaseSensitive::Choice caseSensitivity=CaseSensitive::Yes)
- [StdString::RegexMatcher Matches](#) (std::string const ®ex, CaseSensitive::Choice caseSensitivity=CaseSensitive::Yes)
- [template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp> Vector::ContainsMatcher< T, AllocComp, AllocMatch > Contains](#) (std::vector< T, AllocComp > const &comparator)
- [template<typename T, typename Alloc = std::allocator<T>> Vector::ContainsElementMatcher< T, Alloc > VectorContains](#) (T const &comparator)
- [template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp> Vector::EqualsMatcher< T, AllocComp, AllocMatch > Equals](#) (std::vector< T, AllocComp > const &comparator)
- [template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp> Vector::ApproxMatcher< T, AllocComp, AllocMatch > Approx](#) (std::vector< T, AllocComp > const &comparator)
- [template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp> Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch > UnorderedEquals](#) (std::vector< T, AllocComp > const &target)

5.7.1 Function Documentation

5.7.1.1 Approx()

```
template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
Vector::ApproxMatcher< T, AllocComp, AllocMatch > Catch::Matchers::Approx (
    std::vector< T, AllocComp > const & comparator )
```

Definition at line 3754 of file [catch.h](#).

5.7.1.2 Contains() [1/2]

```
StdString::ContainsMatcher Catch::Matchers::Contains (
    std::string const & str,
    CaseSensitive::Choice caseSensitivity = CaseSensitive::Yes )
```

5.7.1.3 Contains() [2/2]

```
template<typename T , typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
Vector::ContainsMatcher< T, AllocComp, AllocMatch > Catch::Matchers::Contains (
    std::vector< T, AllocComp > const & comparator )
```

Definition at line 3739 of file [catch.h](#).

5.7.1.4 EndsWith()

```
StdString::EndsWithMatcher Catch::Matchers::EndsWith (
    std::string const & str,
    CaseSensitive::Choice caseSensitivity = CaseSensitive::Yes )
```

5.7.1.5 Equals() [1/2]

```
StdString::EqualsMatcher Catch::Matchers::Equals (
    std::string const & str,
    CaseSensitive::Choice caseSensitivity = CaseSensitive::Yes )
```

5.7.1.6 Equals() [2/2]

```
template<typename T , typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
Vector::EqualsMatcher< T, AllocComp, AllocMatch > Catch::Matchers::Equals (
    std::vector< T, AllocComp > const & comparator )
```

Definition at line 3749 of file [catch.h](#).

5.7.1.7 Matches()

```
StdString::RegexMatcher Catch::Matchers::Matches (
    std::string const & regex,
    CaseSensitive::Choice caseSensitivity = CaseSensitive::Yes )
```

5.7.1.8 Message()

```
Exception::ExceptionMessageMatcher Catch::Matchers::Message (
    std::string const & message )
```

5.7.1.9 Predicate()

```
template<typename T >
Generic::PredicateMatcher< T > Catch::Matchers::Predicate (
    std::function< bool(T const &)> const & predicate,
    std::string const & description = "" )
```

Definition at line 3521 of file [catch.h](#).

5.7.1.10 StartsWith()

```
StdString::StartsWithMatcher Catch::Matchers::StartsWith (
    std::string const & str,
    CaseSensitive::Choice caseSensitivity = CaseSensitive::Yes )
```

5.7.1.11 UnorderedEquals()

```
template<typename T , typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch > Catch::Matchers::UnorderedEquals (
    std::vector< T, AllocComp > const & target )
```

Definition at line 3759 of file [catch.h](#).

5.7.1.12 VectorContains()

```
template<typename T , typename Alloc = std::allocator<T>>
Vector::ContainsElementMatcher< T, Alloc > Catch::Matchers::VectorContains (
    T const & comparator )
```

Definition at line 3744 of file [catch.h](#).

5.7.1.13 WithinAbs()

```
Floating::WithinAbsMatcher Catch::Matchers::WithinAbs (
    double target,
    double margin )
```

5.7.1.14 WithinRel() [1/4]

```
Floating::WithinRelMatcher Catch::Matchers::WithinRel (
    double target )
```

5.7.1.15 WithinRel() [2/4]

```
Floating::WithinRelMatcher Catch::Matchers::WithinRel (
    double target,
    double eps )
```

5.7.1.16 WithinRel() [3/4]

```
Floating::WithinRelMatcher Catch::Matchers::WithinRel (
    float target )
```

5.7.1.17 WithinRel() [4/4]

```
Floating::WithinRelMatcher Catch::Matchers::WithinRel (
    float target,
    float eps )
```

5.7.1.18 WithinULP() [1/2]

```
Floating::WithinUlpMatcher Catch::Matchers::WithinULP (
    double target,
    uint64_t maxUlpDiff )
```

5.7.1.19 WithinULP() [2/2]

```
Floating::WithinUlpMatcher Catch::Matchers::WithinULP (
    float target,
    uint64_t maxUlpDiff )
```

5.8 Catch::Matchers::Exception Namespace Reference**Classes**

- class [ExceptionMessageMatcher](#)

5.9 Catch::Matchers::Floating Namespace Reference**Classes**

- struct [WithinAbsMatcher](#)
- struct [WithinRelMatcher](#)
- struct [WithinUlpMatcher](#)

5.10 Catch::Matchers::Generic Namespace Reference**Namespaces**

- namespace [Detail](#)

Classes

- class [PredicateMatcher](#)

5.11 Catch::Matchers::Generic::Detail Namespace Reference

Functions

- std::string [finalizeDescription](#) (const std::string &desc)

5.11.1 Function Documentation

5.11.1.1 finalizeDescription()

```
std::string Catch::Matchers::Generic::Detail::finalizeDescription (
    const std::string & desc )
```

5.12 Catch::Matchers::Impl Namespace Reference

Classes

- struct [MatchAllOf](#)
- struct [MatchAnyOf](#)
- struct [MatcherBase](#)
- struct [MatcherMethod](#)
- class [MatcherUntypedBase](#)
- struct [MatchNotOf](#)

5.13 Catch::Matchers::StdString Namespace Reference

Classes

- struct [CasedString](#)
- struct [ContainsMatcher](#)
- struct [EndsWithMatcher](#)
- struct [EqualsMatcher](#)
- struct [RegexMatcher](#)
- struct [StartsWithMatcher](#)
- struct [StringMatcherBase](#)

5.14 Catch::Matchers::Vector Namespace Reference

Classes

- struct [ApproxMatcher](#)
- struct [ContainsElementMatcher](#)
- struct [ContainsMatcher](#)
- struct [EqualsMatcher](#)
- struct [UnorderedEqualsMatcher](#)

5.15 mpl_ Namespace Reference

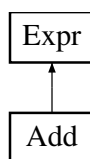
Chapter 6

Class Documentation

6.1 Add Class Reference

```
#include <Expr.h>
```

Inheritance diagram for Add:



Public Member Functions

- [Add](#) ([Expr](#) *lhs, [Expr](#) *rhs)
- bool [equals](#) ([Expr](#) *e) override
- int [interp](#) () override
- bool [hasVariable](#) () override
- [Expr](#) * [subst](#) (std::string stringInput, [Expr](#) *e) override
- void [print](#) (std::ostream &stream) override

Public Member Functions inherited from [Expr](#)

- virtual [~Expr](#) ()=default
- std::string [to_string](#) ()
- std::string [to_pp_string](#) ()
- void [pretty_print_at](#) (std::ostream &ot)

Public Attributes

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

Protected Member Functions

- void [pretty_print](#) (std::ostream &ot, [precedence_t](#) prec) override

6.1.1 Detailed Description

Definition at line 59 of file [Expr.h](#).

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Add()

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

Constructs an [Add](#) object with left and right expressions.

Parameters

<i>lhs</i>	Pointer to the left-hand side expression.
<i>rhs</i>	Pointer to the right-hand side expression.

Definition at line 17 of file [Expr.cpp](#).

6.1.3 Member Function Documentation

6.1.3.1 equals()

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

Checks if this [Add](#) object is equal to another expression.

Parameters

<i>e</i>	Pointer to the expression to compare with.
----------	--

Returns

true if the expressions are equivalent, false otherwise.

Implements [Expr](#).

Definition at line 54 of file [Expr.cpp](#).

6.1.3.2 hasVariable()

```
bool Add::hasVariable ( ) [override], [virtual]
```

Checks if the [Add](#) expression contains a variable.

Returns

true if either the left or right expressions contain a variable, false otherwise.

Implements [Expr](#).

Definition at line 125 of file [Expr.cpp](#).

6.1.3.3 interp()

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

Evaluates the addition expression.

Returns

The sum of the left and right expressions.

Implements [Expr](#).

Definition at line 93 of file [Expr.cpp](#).

6.1.3.4 pretty_print()

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

Pretty prints the [Add](#) expression with appropriate precedence.

Parameters

<i>ot</i>	The output stream to print to.
<i>prec</i>	The precedence context in which this expression is being printed.

Implements [Expr](#).

Definition at line 247 of file [Expr.cpp](#).

6.1.3.5 print()

```
void Add::print (
    std::ostream & stream ) [override], [virtual]
```

Prints the [Add](#) expression to a given output stream.

Parameters

<i>stream</i>	The output stream to print to.
---------------	--------------------------------

Implements [Expr](#).

Definition at line 205 of file [Expr.cpp](#).

6.1.3.6 subst()

```
Expr * Add::subst (
    std::string stringInput,
    Expr * e ) [override], [virtual]
```

Substitutes a variable with another expression in an [Add](#) object.

Parameters

<i>stringInput</i>	The name of the variable to substitute.
<i>e</i>	The expression to substitute the variable with.

Returns

A new [Add](#) object with the substituted expressions.

Implements [Expr](#).

Definition at line 161 of file [Expr.cpp](#).

6.1.4 Member Data Documentation**6.1.4.1 lhs**

```
Expr* Add::lhs
```

Definition at line 61 of file [Expr.h](#).

6.1.4.2 rhs

```
Expr* Add::rhs
```

Definition at line 62 of file [Expr.h](#).

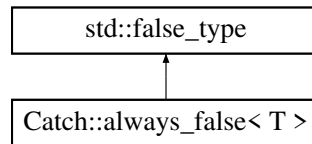
The documentation for this class was generated from the following files:

- [/Users/samanthapope/msdscriptRepo/msdScript/Expr.h](#)
- [/Users/samanthapope/msdscriptRepo/msdScript/Expr.cpp](#)

6.2 Catch::always_false< T > Struct Template Reference

```
#include <catch.h>
```

Inheritance diagram for Catch::always_false< T >:



6.2.1 Detailed Description

```
template<typename T>
struct Catch::always_false< T >
```

Definition at line 925 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.3 Catch::Detail::Approx Class Reference

```
#include <catch.h>
```

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) const
- [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)`

6.3.1 Detailed Description

Definition at line 3078 of file [catch.h](#).

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Approx() [1/2]

```
Catch::Detail::Approx::Approx (
    double value ) [explicit]
```

6.3.2.2 Approx() [2/2]

```
template<typename T, typename = typename std::enable_if<std::is_constructible<double, T>↵
::value>::type>
Catch::Detail::Approx::Approx (
    T const & value ) [inline], [explicit]
```

Definition at line 3105 of file [catch.h](#).

6.3.3 Member Function Documentation

6.3.3.1 custom()

```
static Approx Catch::Detail::Approx::custom ( ) [static]
```

6.3.3.2 epsilon()

```
template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>&
::value>::type>
Approx & Catch::Detail::Approx::epsilon (
    T const & newEpsilon ) [inline]
```

Definition at line 3150 of file [catch.h](#).

6.3.3.3 margin()

```
template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>&
::value>::type>
Approx & Catch::Detail::Approx::margin (
    T const & newMargin ) [inline]
```

Definition at line 3157 of file [catch.h](#).

6.3.3.4 operator>()

```
template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>&
::value>::type>
Approx Catch::Detail::Approx::operator() (
    T const & value ) const [inline]
```

Definition at line 3096 of file [catch.h](#).

6.3.3.5 operator-()

```
Approx Catch::Detail::Approx::operator- ( ) const
```

6.3.3.6 scale()

```
template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>&
::value>::type>
Approx & Catch::Detail::Approx::scale (
    T const & newScale ) [inline]
```

Definition at line 3164 of file [catch.h](#).

6.3.3.7 toString()

```
std::string Catch::Detail::Approx::toString ( ) const
```

6.3.4 Friends And Related Symbol Documentation

6.3.4.1 operator"!=" [1/2]

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

Definition at line 3125 of file [catch.h](#).

6.3.4.2 operator"!=" [2/2]

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

Definition at line 3120 of file [catch.h](#).

6.3.4.3 operator<= [1/2]

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

Definition at line 3135 of file [catch.h](#).

6.3.4.4 operator<= [2/2]

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

Definition at line 3130 of file [catch.h](#).

6.3.4.5 operator== [1/2]

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

Definition at line 3115 of file [catch.h](#).

6.3.4.6 operator== [2/2]

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

Definition at line 3109 of file [catch.h](#).

6.3.4.7 operator>= [1/2]

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

Definition at line 3145 of file [catch.h](#).

6.3.4.8 operator>= [2/2]

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

Definition at line 3140 of file [catch.h](#).

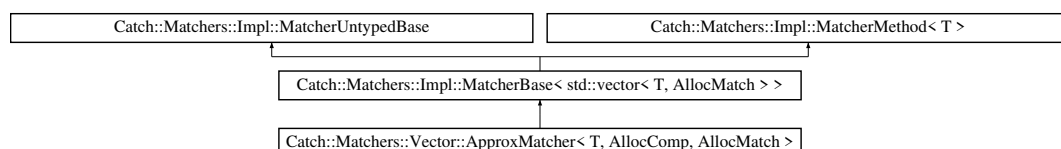
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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](#) ()=default
- [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 Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [virtual ~MatcherUntypedBase](#) ()

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

- [std::string m_cachedToString](#)

6.4.1 Detailed Description

```
template<typename T, typename AllocComp, typename AllocMatch>
struct Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >
```

Definition at line 3681 of file [catch.h](#).

6.4.2 Constructor & Destructor Documentation

6.4.2.1 ApproxMatcher()

```
template<typename T , typename AllocComp , typename AllocMatch >
Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >::ApproxMatcher (
    std::vector< T, AllocComp > const & comparator ) [inline]
```

Definition at line 3683 of file [catch.h](#).

6.4.3 Member Function Documentation

6.4.3.1 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](#).

Definition at line 3693 of file [catch.h](#).

6.4.3.2 epsilon()

```
template<typename T , typename AllocComp , typename AllocMatch >
template<typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
ApproxMatcher & Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >::epsilon (
    T const & newEpsilon ) [inline]
```

Definition at line 3697 of file [catch.h](#).

6.4.3.3 margin()

```
template<typename T , typename AllocComp , typename AllocMatch >
template<typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
ApproxMatcher & Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >::margin (
    T const & newMargin ) [inline]
```

Definition at line 3702 of file [catch.h](#).

6.4.3.4 match()

```
template<typename T , typename AllocComp , typename AllocMatch >
bool Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >::match (
    std::vector< T, AllocMatch > const & v ) const [inline], [override]
```

Definition at line 3685 of file [catch.h](#).

6.4.3.5 scale()

```
template<typename T , typename AllocComp , typename AllocMatch >
template<typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
ApproxMatcher & Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >::scale (
    T const & newScale ) [inline]
```

Definition at line 3707 of file [catch.h](#).

6.4.4 Member Data Documentation

6.4.4.1 approx

```
template<typename T , typename AllocComp , typename AllocMatch >
Catch::Detail::Approx Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >↔
::approx = Catch::Detail::Approx::custom() [mutable]
```

Definition at line 3713 of file [catch.h](#).

6.4.4.2 m_comparator

```
template<typename T , typename AllocComp , typename AllocMatch >
std::vector<T, AllocComp> const& Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch
>::m_comparator
```

Definition at line 3712 of file [catch.h](#).

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

6.5.1 Detailed Description

```
template<typename T>
struct Catch::Generators::as< T >
```

Definition at line 4060 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.6 Catch::AssertionHandler Class Reference

```
#include <catch.h>
```

Public Member Functions

- [AssertionHandler](#) ([StringRef](#) const ¯oName, [SourceLineInfo](#) const &lineInfo, [StringRef](#) capturedExpression, [ResultDisposition::Flags](#) resultDisposition)
- [~AssertionHandler](#) ()
- [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

6.6.1 Detailed Description

Definition at line 2548 of file [catch.h](#).

6.6.2 Constructor & Destructor Documentation

6.6.2.1 AssertionHandler()

```
Catch::AssertionHandler::AssertionHandler (
    StringRef const & macroName,
    SourceLineInfo const & lineInfo,
    StringRef capturedExpression,
    ResultDisposition::Flags resultDisposition )
```


6.6.2.2 ~AssertionHandler()

```
Catch::AssertionHandler::~~AssertionHandler ( ) [inline]
```

Definition at line 2560 of file [catch.h](#).

6.6.3 Member Function Documentation

6.6.3.1 allowThrows()

```
auto Catch::AssertionHandler::allowThrows ( ) const -> bool
```

6.6.3.2 complete()

```
void Catch::AssertionHandler::complete ( )
```

6.6.3.3 handleExceptionNotThrownAsExpected()

```
void Catch::AssertionHandler::handleExceptionNotThrownAsExpected ( )
```

6.6.3.4 handleExceptionThrownAsExpected()

```
void Catch::AssertionHandler::handleExceptionThrownAsExpected ( )
```

6.6.3.5 handleExpr() [1/2]

```
template<typename T >  
void Catch::AssertionHandler::handleExpr (   
    ExprLhs< T > const & expr ) [inline]
```

Definition at line 2567 of file [catch.h](#).

6.6.3.6 handleExpr() [2/2]

```
void Catch::AssertionHandler::handleExpr (   
    ITransientExpression const & expr )
```

6.6.3.7 handleMessage()

```
void Catch::AssertionHandler::handleMessage (   
    ResultWas::OfType resultType,   
   StringRef const & message )
```

6.6.3.8 handleThrowingCallSkipped()

```
void Catch::AssertionHandler::handleThrowingCallSkipped ( )
```

6.6.3.9 handleUnexpectedExceptionNotThrown()

```
void Catch::AssertionHandler::handleUnexpectedExceptionNotThrown ( )
```

6.6.3.10 handleUnexpectedInflightException()

```
void Catch::AssertionHandler::handleUnexpectedInflightException ( )
```

6.6.3.11 setCompleted()

```
void Catch::AssertionHandler::setCompleted ( )
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.7 Catch::AssertionInfo Struct Reference

```
#include <catch.h>
```

Public Attributes

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

6.7.1 Detailed Description

Definition at line [1396](#) of file [catch.h](#).

6.7.2 Member Data Documentation

6.7.2.1 capturedExpression

```
StringRef Catch::AssertionInfo::capturedExpression
```

Definition at line [1400](#) of file [catch.h](#).

6.7.2.2 lineInfo

`SourceLineInfo` `Catch::AssertionInfo::lineInfo`

Definition at line 1399 of file [catch.h](#).

6.7.2.3 macroName

`StringRef` `Catch::AssertionInfo::macroName`

Definition at line 1398 of file [catch.h](#).

6.7.2.4 resultDisposition

`ResultDisposition::Flags` `Catch::AssertionInfo::resultDisposition`

Definition at line 1401 of file [catch.h](#).

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.8 Catch::AssertionReaction Struct Reference

```
#include <catch.h>
```

Public Attributes

- `bool shouldDebugBreak = false`
- `bool shouldThrow = false`

6.8.1 Detailed Description

Definition at line 2543 of file [catch.h](#).

6.8.2 Member Data Documentation

6.8.2.1 shouldDebugBreak

`bool` `Catch::AssertionReaction::shouldDebugBreak = false`

Definition at line 2544 of file [catch.h](#).

6.8.2.2 shouldThrow

```
bool Catch::AssertionReaction::shouldThrow = false
```

Definition at line 2545 of file [catch.h](#).

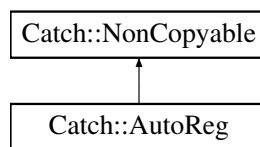
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.9 Catch::AutoReg Struct Reference

```
#include <catch.h>
```

Inheritance diagram for Catch::AutoReg:



Public Member Functions

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

Additional Inherited Members

Protected Member Functions inherited from [Catch::NonCopyable](#)

- [NonCopyable](#) ()
- [virtual ~NonCopyable](#) ()

6.9.1 Detailed Description

Definition at line 986 of file [catch.h](#).

6.9.2 Constructor & Destructor Documentation

6.9.2.1 AutoReg()

```

Catch::AutoReg::AutoReg (
    ITestInvoker * invoker,
    SourceLineInfo const & lineInfo,
    StringRef const & classOrMethod,
    NameAndTags const & nameAndTags ) [noexcept]
  
```

6.9.2.2 ~AutoReg()

```
Catch::AutoReg::~~AutoReg ( )
```

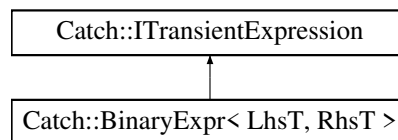
The documentation for this struct was generated from the following file:

- </Users/samanthapope/msdscriptRepo/msdScript/catch.h>

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

```
#include <catch.h>
```

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)
- [virtual ~ITransientExpression \(\)](#)

Additional Inherited Members

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

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

6.10.1 Detailed Description

```
template<typename LhsT, typename RhstT>
class Catch::BinaryExpr< LhsT, RhstT >
```

Definition at line 2224 of file [catch.h](#).

6.10.2 Constructor & Destructor Documentation

6.10.2.1 BinaryExpr()

```
template<typename LhsT , typename RhstT >
Catch::BinaryExpr< LhsT, RhstT >::BinaryExpr (
    bool comparisonResult,
    LhsT lhs,
    StringRef op,
    RhstT rhs ) [inline]
```

Definition at line 2235 of file [catch.h](#).

6.10.3 Member Function Documentation

6.10.3.1 operator"!==(

```
template<typename LhsT , typename RhstT >
template<typename T >
auto Catch::BinaryExpr< LhsT, RhstT >::operator!=(
    T ) const -> BinaryExpr<LhsT, RhstT const&> const [inline]
```

Definition at line 2264 of file [catch.h](#).

6.10.3.2 operator&&(

```
template<typename LhsT , typename RhstT >
template<typename T >
auto Catch::BinaryExpr< LhsT, RhstT >::operator&& (
    T ) const -> BinaryExpr<LhsT, RhstT const&> const [inline]
```

Definition at line 2243 of file [catch.h](#).

6.10.3.3 operator<()

```
template<typename LhsT , typename RhsT >
template<typename T >
auto Catch::BinaryExpr< LhsT, RhsT >::operator< (
    T ) const -> BinaryExpr<LhsT, RhsT const&> const    [inline]
```

Definition at line 2278 of file [catch.h](#).

6.10.3.4 operator<=()

```
template<typename LhsT , typename RhsT >
template<typename T >
auto Catch::BinaryExpr< LhsT, RhsT >::operator<= (
    T ) const -> BinaryExpr<LhsT, RhsT const&> const    [inline]
```

Definition at line 2292 of file [catch.h](#).

6.10.3.5 operator==(())

```
template<typename LhsT , typename RhsT >
template<typename T >
auto Catch::BinaryExpr< LhsT, RhsT >::operator==(
    T ) const -> BinaryExpr<LhsT, RhsT const&> const    [inline]
```

Definition at line 2257 of file [catch.h](#).

6.10.3.6 operator>()

```
template<typename LhsT , typename RhsT >
template<typename T >
auto Catch::BinaryExpr< LhsT, RhsT >::operator> (
    T ) const -> BinaryExpr<LhsT, RhsT const&> const    [inline]
```

Definition at line 2271 of file [catch.h](#).

6.10.3.7 operator>=()

```
template<typename LhsT , typename RhsT >
template<typename T >
auto Catch::BinaryExpr< LhsT, RhsT >::operator>= (
    T ) const -> BinaryExpr<LhsT, RhsT const&> const    [inline]
```

Definition at line 2285 of file [catch.h](#).

6.10.3.8 operator"|"|()

```
template<typename LhsT , typename RhsT >
template<typename T >
auto Catch::BinaryExpr< LhsT, RhsT >::operator|| (
    T ) const -> BinaryExpr<LhsT, RhsT const&> const [inline]
```

Definition at line 2250 of file [catch.h](#).

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.11 Catch::Capturer Class Reference

```
#include <catch.h>
```

Public Member Functions

- [Capturer](#) ([StringRef](#) macroName, [SourceLineInfo](#) const &lineInfo, [ResultWas::OfType](#) resultType, [StringRef](#) names)
- [~Capturer](#) ()
- [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)

6.11.1 Detailed Description

Definition at line 2652 of file [catch.h](#).

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Capturer()

```
Catch::Capturer::Capturer (
    StringRef macroName,
    SourceLineInfo const & lineInfo,
    ResultWas::OfType resultType,
    StringRef names )
```

6.11.2.2 ~Capturer()

```
Catch::Capturer::~Capturer ( )
```


6.11.3 Member Function Documentation

6.11.3.1 captureValue()

```
void Catch::Capturer::captureValue (
    size_t index,
    std::string const & value )
```

6.11.3.2 captureValues() [1/2]

```
template<typename T >
void Catch::Capturer::captureValues (
    size_t index,
    T const & value ) [inline]
```

Definition at line 2663 of file [catch.h](#).

6.11.3.3 captureValues() [2/2]

```
template<typename T , typename... Ts>
void Catch::Capturer::captureValues (
    size_t index,
    T const & value,
    Ts const &... values ) [inline]
```

Definition at line 2668 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.12 Catch::Matchers::StdString::CasedString Struct Reference

```
#include <catch.h>
```

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

6.12.1 Detailed Description

Definition at line 3538 of file [catch.h](#).

6.12.2 Constructor & Destructor Documentation

6.12.2.1 CasedString()

```
Catch::Matchers::StdString::CasedString::CasedString (
    std::string const & str,
    CaseSensitive::Choice caseSensitivity )
```

6.12.3 Member Function Documentation

6.12.3.1 adjustString()

```
std::string Catch::Matchers::StdString::CasedString::adjustString (
    std::string const & str ) const
```

6.12.3.2 caseSensitivitySuffix()

```
std::string Catch::Matchers::StdString::CasedString::caseSensitivitySuffix ( ) const
```

6.12.4 Member Data Documentation

6.12.4.1 m_caseSensitivity

```
CaseSensitive::Choice Catch::Matchers::StdString::CasedString::m_caseSensitivity
```

Definition at line 3544 of file [catch.h](#).

6.12.4.2 m_str

```
std::string Catch::Matchers::StdString::CasedString::m_str
```

Definition at line 3545 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.13 Catch::CaseSensitive Struct Reference

```
#include <catch.h>
```

Public Types

- enum [Choice](#) { [Yes](#) , [No](#) }

6.13.1 Detailed Description

Definition at line [486](#) of file [catch.h](#).

6.13.2 Member Enumeration Documentation

6.13.2.1 Choice

```
enum Catch::CaseSensitive::Choice
```

Enumerator

Yes	
No	

Definition at line [486](#) of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.14 Catch_global_namespace_dummy Struct Reference

```
#include <catch.h>
```

6.14.1 Detailed Description

Definition at line [481](#) of file [catch.h](#).

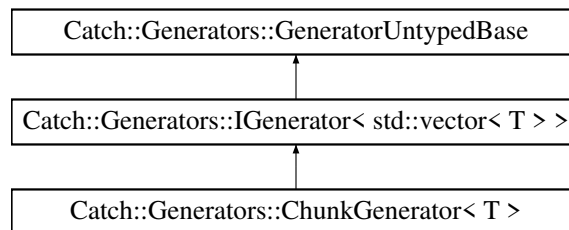
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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](#)

Public Member Functions inherited from

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

- [virtual](#) [~IGenerator](#) ()=[default](#)

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

- [GeneratorUntypedBase](#) ()=[default](#)
- [virtual](#) [~GeneratorUntypedBase](#) ()

Additional Inherited Members

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

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

6.15.1 Detailed Description

```
template<typename T>
class Catch::Generators::ChunkGenerator< T >
```

Definition at line [4297](#) of file [catch.h](#).

6.15.2 Constructor & Destructor Documentation

6.15.2.1 ChunkGenerator()

```
template<typename T >
Catch::Generators::ChunkGenerator< T >::ChunkGenerator (
    size_t size,
    GeneratorWrapper< T > generator ) [inline]
```

Definition at line 4303 of file [catch.h](#).

6.15.3 Member Function Documentation

6.15.3.1 get()

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

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

Definition at line 4317 of file [catch.h](#).

6.15.3.2 next()

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

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

Definition at line 4320 of file [catch.h](#).

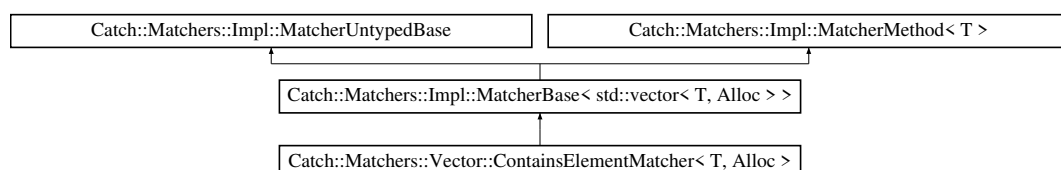
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.16 Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc > Struct Template Reference

```
#include <catch.h>
```

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](#) ()=default
- [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 Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- [virtual ~MatcherUntypedBase](#) ()

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

- [std::string m_cachedToString](#)

6.16.1 Detailed Description

```
template<typename T, typename Alloc>
struct Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >
```

Definition at line 3607 of file [catch.h](#).

6.16.2 Constructor & Destructor Documentation

6.16.2.1 ContainsElementMatcher()

```
template<typename T , typename Alloc >
Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >::ContainsElementMatcher (
    T const & comparator ) [inline]
```

Definition at line 3609 of file [catch.h](#).

6.16.3 Member Function Documentation

6.16.3.1 describe()

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

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

Definition at line 3620 of file [catch.h](#).

6.16.3.2 match()

```
template<typename T , typename Alloc >
bool Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >::match (
    std::vector< T, Alloc > const & v ) const [inline], [override]
```

Definition at line 3611 of file [catch.h](#).

6.16.4 Member Data Documentation

6.16.4.1 m_comparator

```
template<typename T , typename Alloc >
T const& Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >::m_comparator
```

Definition at line 3624 of file [catch.h](#).

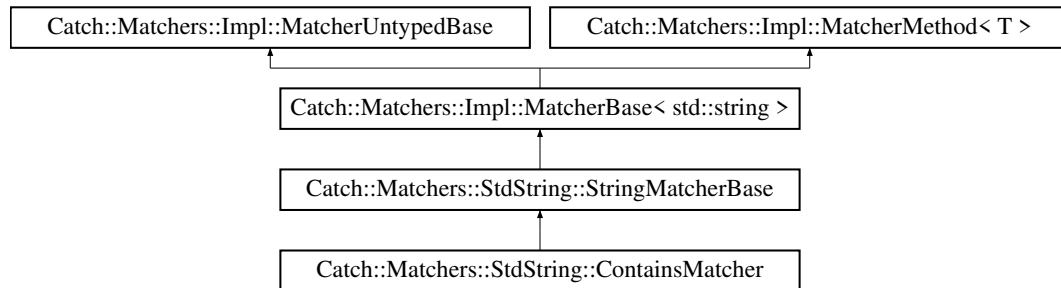
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.17 Catch::Matchers::StdString::ContainsMatcher Struct Reference

```
#include <catch.h>
```

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](#) ()=default
- [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 Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- [virtual ~MatcherUntypedBase\(\)](#)

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

- `std::string` [m_cachedToString](#)

6.17.1 Detailed Description

Definition at line 3560 of file [catch.h](#).

6.17.2 Constructor & Destructor Documentation

6.17.2.1 ContainsMatcher()

```
Catch::Matchers::StdString::ContainsMatcher::ContainsMatcher (
    CasedString const & comparator )
```

6.17.3 Member Function Documentation

6.17.3.1 match()

```
bool Catch::Matchers::StdString::ContainsMatcher::match (
    std::string const & source ) const [override]
```

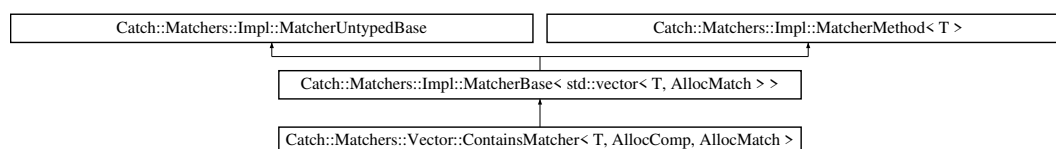
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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](#) ()=default
- [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 Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- [virtual ~MatcherUntypedBase](#) ()

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

- std::string [m_cachedToString](#)

6.18.1 Detailed Description

```
template<typename T, typename AllocComp, typename AllocMatch>
struct Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >
```

Definition at line 3628 of file [catch.h](#).

6.18.2 Constructor & Destructor Documentation

6.18.2.1 ContainsMatcher()

```
template<typename T , typename AllocComp , typename AllocMatch >
Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >::ContainsMatcher (
    std::vector< T, AllocComp > const & comparator ) [inline]
```

Definition at line 3630 of file [catch.h](#).

6.18.3 Member Function Documentation

6.18.3.1 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](#).

Definition at line 3650 of file [catch.h](#).

6.18.3.2 match()

```
template<typename T , typename AllocComp , typename AllocMatch >
bool Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >::match (
    std::vector< T, AllocMatch > const & v ) const [inline], [override]
```

Definition at line 3632 of file [catch.h](#).

6.18.4 Member Data Documentation

6.18.4.1 m_comparator

```
template<typename T , typename AllocComp , typename AllocMatch >
std::vector<T, AllocComp> const& Catch::Matchers::Vector::ContainsMatcher< T, AllocComp,
AllocMatch >::m_comparator
```

Definition at line 3654 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.19 Catch::Counts Struct Reference

```
#include <catch.h>
```

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

6.19.1 Detailed Description

Definition at line [2829](#) of file [catch.h](#).

6.19.2 Member Function Documentation

6.19.2.1 allOk()

```
bool Catch::Counts::allOk ( ) const
```

6.19.2.2 allPassed()

```
bool Catch::Counts::allPassed ( ) const
```

6.19.2.3 operator+=()

```
Counts & Catch::Counts::operator+= (  
    Counts const & other )
```

6.19.2.4 operator-()

```
Counts Catch::Counts::operator- (  
    Counts const & other ) const
```

6.19.2.5 total()

```
std::size_t Catch::Counts::total ( ) const
```

6.19.3 Member Data Documentation

6.19.3.1 failed

```
std::size_t Catch::Counts::failed = 0
```

Definition at line 2838 of file [catch.h](#).

6.19.3.2 failedButOk

```
std::size_t Catch::Counts::failedButOk = 0
```

Definition at line 2839 of file [catch.h](#).

6.19.3.3 passed

```
std::size_t Catch::Counts::passed = 0
```

Definition at line 2837 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.20 Catch::Decomposer Struct Reference

```
#include <catch.h>
```

Public Member Functions

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

6.20.1 Detailed Description

Definition at line 2414 of file [catch.h](#).

6.20.2 Member Function Documentation

6.20.2.1 operator<=() [1/2]

```
auto Catch::Decomposer::operator<= (
    bool value ) -> ExprLhs<bool>    [inline]
```

Definition at line 2420 of file [catch.h](#).

6.20.2.2 operator<=() [2/2]

```
template<typename T >
auto Catch::Decomposer::operator<= (
    T const & lhs ) -> ExprLhs<T const&>    [inline]
```

Definition at line 2416 of file [catch.h](#).

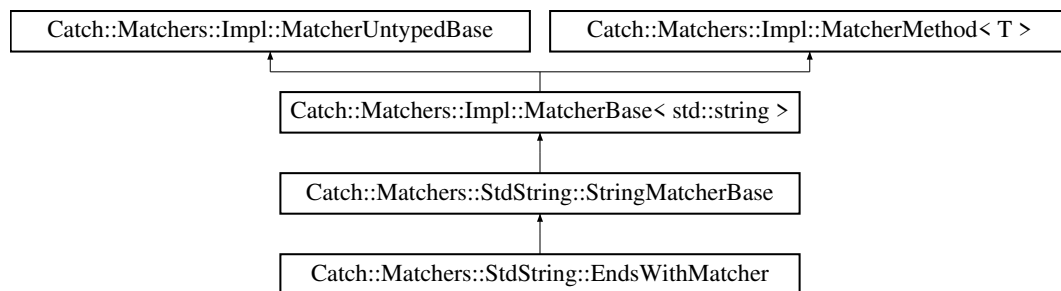
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.21 Catch::Matchers::StdString::EndsWithMatcher Struct Reference

```
#include <catch.h>
```

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](#) ()=default
- [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 Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [virtual ~MatcherUntypedBase](#) ()

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

- [std::string m_cachedToString](#)

6.21.1 Detailed Description

Definition at line 3568 of file [catch.h](#).

6.21.2 Constructor & Destructor Documentation**6.21.2.1 EndsWithMatcher()**

```
Catch::Matchers::StdString::EndsWithMatcher::EndsWithMatcher (
    CasedString const & comparator )
```

6.21.3 Member Function Documentation**6.21.3.1 match()**

```
bool Catch::Matchers::StdString::EndsWithMatcher::match (
    std::string const & source ) const [override]
```

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.22 Catch::Detail::EnumInfo Struct Reference

```
#include <catch.h>
```

Public Member Functions

- [~EnumInfo](#) ()
- [StringRef lookup](#) (int value) const

Public Attributes

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

6.22.1 Detailed Description

Definition at line [1466](#) of file [catch.h](#).

6.22.2 Constructor & Destructor Documentation

6.22.2.1 ~EnumInfo()

```
Catch::Detail::EnumInfo::~~EnumInfo ( )
```

6.22.3 Member Function Documentation

6.22.3.1 lookup()

```
StringRef Catch::Detail::EnumInfo::lookup (
    int value ) const
```

6.22.4 Member Data Documentation

6.22.4.1 m_name

```
StringRef Catch::Detail::EnumInfo::m_name
```

Definition at line [1467](#) of file [catch.h](#).

6.22.4.2 m_values

```
std::vector<std::pair<int, StringRef> > Catch::Detail::EnumInfo::m_values
```

Definition at line [1468](#) of file [catch.h](#).

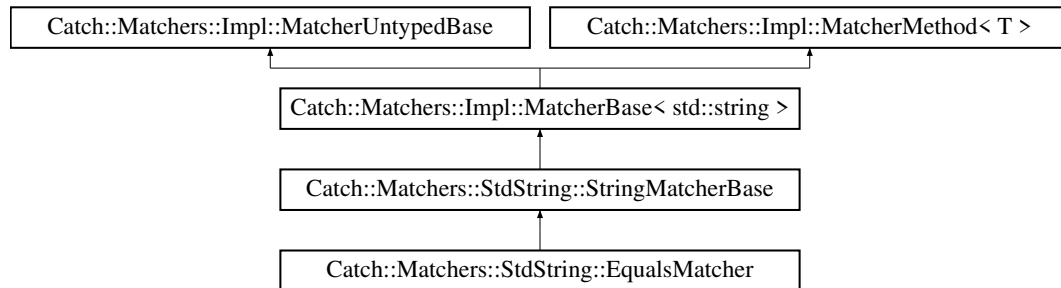
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.23 Catch::Matchers::StdString::EqualsMatcher Struct Reference

```
#include <catch.h>
```

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](#) ()=default
- [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 Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [virtual ~MatcherUntypedBase\(\)](#)

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

- `std::string` [m_cachedToString](#)

6.23.1 Detailed Description

Definition at line 3556 of file [catch.h](#).

6.23.2 Constructor & Destructor Documentation

6.23.2.1 EqualsMatcher()

```
Catch::Matchers::StdString::EqualsMatcher::EqualsMatcher (
    CasedString const & comparator )
```

6.23.3 Member Function Documentation

6.23.3.1 match()

```
bool Catch::Matchers::StdString::EqualsMatcher::match (
    std::string const & source ) const [override]
```

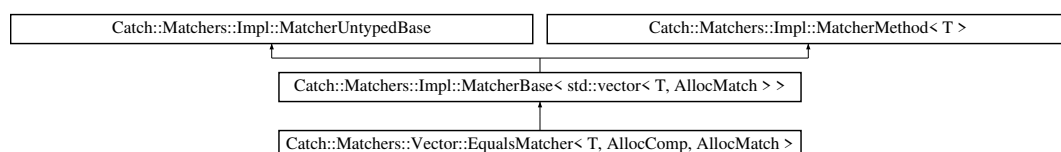
The documentation for this struct was generated from the following file:

- `/Users/samanthapope/msdscriptRepo/msdScript/catch.h`

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

```
#include <catch.h>
```

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](#) ()=default
- [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 Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)**

- [virtual ~MatcherUntypedBase](#) ()

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

- std::string [m_cachedToString](#)

6.24.1 Detailed Description

```
template<typename T, typename AllocComp, typename AllocMatch>
struct Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >
```

Definition at line 3658 of file [catch.h](#).

6.24.2 Constructor & Destructor Documentation

6.24.2.1 EqualsMatcher()

```
template<typename T , typename AllocComp , typename AllocMatch >
Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >::EqualsMatcher (
    std::vector< T, AllocComp > const & comparator ) [inline]
```

Definition at line 3660 of file [catch.h](#).

6.24.3 Member Function Documentation

6.24.3.1 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](#).

Definition at line 3674 of file [catch.h](#).

6.24.3.2 match()

```
template<typename T , typename AllocComp , typename AllocMatch >
bool Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >::match (
    std::vector< T, AllocMatch > const & v ) const [inline], [override]
```

Definition at line 3662 of file [catch.h](#).

6.24.4 Member Data Documentation

6.24.4.1 m_comparator

```
template<typename T , typename AllocComp , typename AllocMatch >
std::vector<T, AllocComp> const& Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch
>::m_comparator
```

Definition at line 3677 of file [catch.h](#).

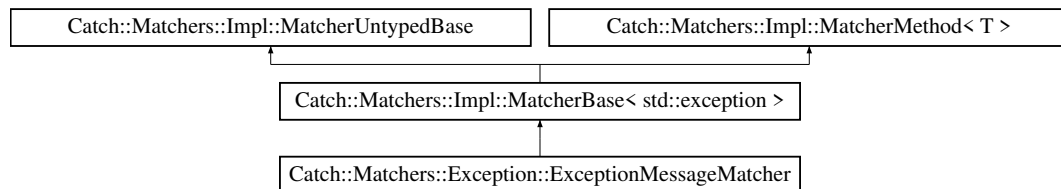
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.25 Catch::Matchers::Exception::ExceptionMessageMatcher Class Reference

```
#include <catch.h>
```

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](#) ()=default
- [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 ~MatcherUntypedBase](#) ()

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

- [std::string m_cachedToString](#)

6.25.1 Detailed Description

Definition at line 3399 of file [catch.h](#).

6.25.2 Constructor & Destructor Documentation

6.25.2.1 ExceptionMessageMatcher()

```
Catch::Matchers::Exception::ExceptionMessageMatcher::ExceptionMessageMatcher (
    std::string const & message ) [inline]
```

Definition at line 3403 of file [catch.h](#).

6.25.3 Member Function Documentation

6.25.3.1 describe()

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

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

6.25.3.2 match()

```
bool Catch::Matchers::Exception::ExceptionMessageMatcher::match (
    std::exception const & ex ) const [override]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.26 Catch::ExceptionTranslatorRegistrar Class Reference

```
#include <catch.h>
```

Public Member Functions

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

6.26.1 Detailed Description

Definition at line 3021 of file [catch.h](#).

6.26.2 Constructor & Destructor Documentation

6.26.2.1 ExceptionTranslatorRegistrar()

```
template<typename T >
Catch::ExceptionTranslatorRegistrar::ExceptionTranslatorRegistrar (
    std::string(*) (T &) translateFunction ) [inline]
```

Definition at line 3052 of file [catch.h](#).

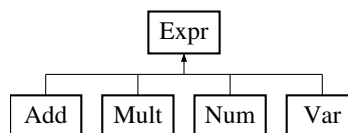
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.27 Expr Class Reference

```
#include <Expr.h>
```

Inheritance diagram for Expr:



Public Member Functions

- virtual [~Expr](#) ()=default
- virtual bool [equals](#) ([Expr](#) *e)=0
- virtual int [interp](#) ()=0
- virtual bool [hasVariable](#) ()=0
- virtual [Expr](#) * [subst](#) (std::string stringInput, [Expr](#) *e)=0
- std::string [to_string](#) ()
- std::string [to_pp_string](#) ()
- virtual void [print](#) (std::ostream &stream)=0
- virtual void [pretty_print](#) (std::ostream &ot, [precedence_t](#) prec)=0
- void [pretty_print_at](#) (std::ostream &ot)

6.27.1 Detailed Description

Definition at line 17 of file [Expr.h](#).

6.27.2 Constructor & Destructor Documentation

6.27.2.1 ~Expr()

```
virtual Expr::~Expr ( ) [virtual], [default]
```

6.27.3 Member Function Documentation

6.27.3.1 equals()

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

Implemented in [Num](#), [Add](#), [Mult](#), and [Var](#).

6.27.3.2 hasVariable()

```
virtual bool Expr::hasVariable ( ) [pure virtual]
```

Implemented in [Num](#), [Add](#), [Mult](#), and [Var](#).

6.27.3.3 interp()

```
virtual int Expr::interp ( ) [pure virtual]
```

Implemented in [Num](#), [Add](#), [Mult](#), and [Var](#).

6.27.3.4 pretty_print()

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

Implemented in [Num](#), [Add](#), [Mult](#), and [Var](#).

6.27.3.5 pretty_print_at()

```
void Expr::pretty_print_at (
    std::ostream & ot ) [inline]
```

Definition at line 41 of file [Expr.h](#).

6.27.3.6 print()

```
virtual void Expr::print (
    std::ostream & stream ) [pure virtual]
```

Implemented in [Num](#), [Add](#), [Mult](#), and [Var](#).

6.27.3.7 subst()

```
virtual Expr * Expr::subst (
    std::string stringInput,
    Expr * e ) [pure virtual]
```

Implemented in [Num](#), [Add](#), [Mult](#), and [Var](#).

6.27.3.8 to_pp_string()

```
std::string Expr::to_pp_string ( ) [inline]
```

Definition at line 31 of file [Expr.h](#).

6.27.3.9 to_string()

```
std::string Expr::to_string ( ) [inline]
```

Definition at line 25 of file [Expr.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/Expr.h](#)

6.28 Catch::ExprLhs< LhsT > Class Template Reference

```
#include <catch.h>
```

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 >](#)

6.28.1 Detailed Description

```
template<typename LhsT>
class Catch::ExprLhs< LhsT >
```

Definition at line 2338 of file [catch.h](#).

6.28.2 Constructor & Destructor Documentation

6.28.2.1 ExprLhs()

```
template<typename LhsT >
Catch::ExprLhs< LhsT >::ExprLhs (
    LhsT lhs ) [inline], [explicit]
```

Definition at line 2341 of file [catch.h](#).

6.28.3 Member Function Documentation

6.28.3.1 makeUnaryExpr()

```
template<typename LhsT >
auto Catch::ExprLhs< LhsT >::makeUnaryExpr ( ) const -> UnaryExpr<LhsT> [inline]
```

Definition at line 2402 of file [catch.h](#).

6.28.3.2 operator"!="() [1/2]

```
template<typename LhsT >
auto Catch::ExprLhs< LhsT >::operator!= (
    bool rhs ) -> BinaryExpr<LhsT, bool> const [inline]
```

Definition at line 2355 of file [catch.h](#).

6.28.3.3 operator"!="() [2/2]

```
template<typename LhsT >
template<typename RhsT >
auto Catch::ExprLhs< LhsT >::operator!= (
    RhsT const & rhs ) -> BinaryExpr<LhsT, RhsT const&> const [inline]
```

Definition at line 2352 of file [catch.h](#).

6.28.3.4 operator&()

```
template<typename LhsT >
template<typename RhsT >
auto Catch::ExprLhs< LhsT >::operator& (
    RhsT const & rhs ) -> BinaryExpr<LhsT, RhsT const&> const [inline]
```

Definition at line 2380 of file [catch.h](#).

6.28.3.5 operator&&()

```
template<typename LhsT >
template<typename RhsT >
auto Catch::ExprLhs< LhsT >::operator&& (
    RhsT const & ) -> BinaryExpr<LhsT, RhsT const&> const [inline]
```

Definition at line 2389 of file [catch.h](#).

6.28.3.6 operator<()

```
template<typename LhsT >
template<typename RhsT >
auto Catch::ExprLhs< LhsT >::operator< (
    RhsT const & rhs ) -> BinaryExpr<LhsT, RhsT const&> const [inline]
```

Definition at line 2364 of file [catch.h](#).

6.28.3.7 operator<=()

```
template<typename LhsT >
template<typename RhsT >
auto Catch::ExprLhs< LhsT >::operator<= (
    RhsT const & rhs ) -> BinaryExpr<LhsT, RhsT const&> const [inline]
```

Definition at line 2372 of file [catch.h](#).

6.28.3.8 operator==() [1/2]

```
template<typename LhsT >
auto Catch::ExprLhs< LhsT >::operator==(
    bool rhs ) -> BinaryExpr<LhsT, bool> const [inline]
```

Definition at line 2347 of file [catch.h](#).

6.28.3.9 operator==() [2/2]

```
template<typename LhsT >
template<typename RhsT >
auto Catch::ExprLhs< LhsT >::operator==(
    RhsT const & rhs ) -> BinaryExpr<LhsT, RhsT const&> const [inline]
```

Definition at line 2344 of file [catch.h](#).

6.28.3.10 operator>()

```
template<typename LhsT >
template<typename RhsT >
auto Catch::ExprLhs< LhsT >::operator> (
    RhsT const & rhs ) -> BinaryExpr<LhsT, RhsT const&> const [inline]
```

Definition at line 2360 of file [catch.h](#).

6.28.3.11 operator>=()

```
template<typename LhsT >
template<typename RhsT >
auto Catch::ExprLhs< LhsT >::operator>= (
    RhsT const & rhs ) -> BinaryExpr<LhsT, RhsT const&> const [inline]
```

Definition at line 2368 of file [catch.h](#).

6.28.3.12 operator^()

```
template<typename LhsT >
template<typename RhsT >
auto Catch::ExprLhs< LhsT >::operator^ (
    RhsT const & rhs ) -> BinaryExpr<LhsT, RhsT const&> const [inline]
```

Definition at line 2384 of file [catch.h](#).

6.28.3.13 operator" |()

```
template<typename LhsT >
template<typename RhsT >
auto Catch::ExprLhs< LhsT >::operator| (
    RhsT const & rhs ) -> BinaryExpr<LhsT, RhsT const&> const [inline]
```

Definition at line 2376 of file [catch.h](#).

6.28.3.14 operator" | " |()

```
template<typename LhsT >
template<typename RhsT >
auto Catch::ExprLhs< LhsT >::operator|| (
    RhsT const & ) -> BinaryExpr<LhsT, RhsT const&> const [inline]
```

Definition at line 2396 of file [catch.h](#).

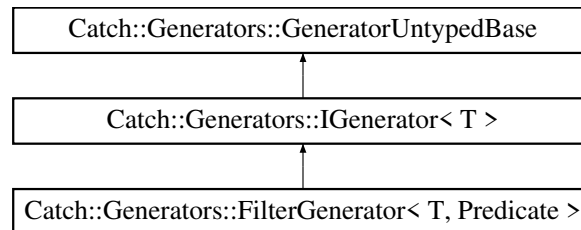
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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`

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

- `virtual ~IGenerator ()=default`

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

- `GeneratorUntypedBase ()=default`
- `virtual ~GeneratorUntypedBase ()`

Additional Inherited Members

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

- `using type = T`

6.29.1 Detailed Description

```
template<typename T, typename Predicate>
class Catch::Generators::FilterGenerator< T, Predicate >
```

Definition at line 4156 of file [catch.h](#).

6.29.2 Constructor & Destructor Documentation

6.29.2.1 FilterGenerator()

```
template<typename T , typename Predicate >
template<typename P = Predicate>
Catch::Generators::FilterGenerator< T, Predicate >::FilterGenerator (
    P && pred,
    GeneratorWrapper< T > && generator ) [inline]
```

Definition at line 4161 of file [catch.h](#).

6.29.3 Member Function Documentation

6.29.3.1 get()

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

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

Definition at line 4175 of file [catch.h](#).

6.29.3.2 next()

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

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

Definition at line 4179 of file [catch.h](#).

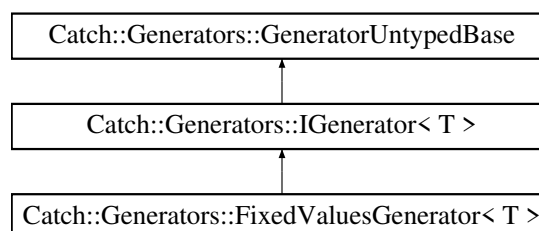
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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



Public Member Functions

- [FixedValuesGenerator](#) (std::initializer_list< T > values)
- [T const & get \(\) const](#) [override](#)
- [bool next \(\)](#) [override](#)

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

- [virtual ~IGenerator \(\)](#)=default

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

- [GeneratorUntypedBase \(\)](#)=default
- [virtual ~GeneratorUntypedBase \(\)](#)

Additional Inherited Members

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

- [using type = T](#)

6.30.1 Detailed Description

```
template<typename T>
class Catch::Generators::FixedValuesGenerator< T >
```

Definition at line 3967 of file [catch.h](#).

6.30.2 Constructor & Destructor Documentation

6.30.2.1 FixedValuesGenerator()

```
template<typename T >
Catch::Generators::FixedValuesGenerator< T >::FixedValuesGenerator (
    std::initializer_list< T > values ) [inline]
```

Definition at line 3974 of file [catch.h](#).

6.30.3 Member Function Documentation

6.30.3.1 get()

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

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

Definition at line 3976 of file [catch.h](#).

6.30.3.2 next()

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

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

Definition at line 3979 of file [catch.h](#).

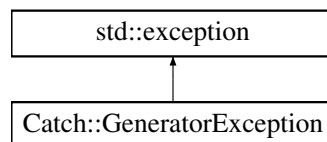
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.31 Catch::GeneratorException Class Reference

```
#include <catch.h>
```

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



Public Member Functions

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

6.31.1 Detailed Description

Definition at line 3919 of file [catch.h](#).

6.31.2 Constructor & Destructor Documentation

6.31.2.1 GeneratorException()

```
Catch::GeneratorException::GeneratorException (
    const char * msg ) [inline]
```

Definition at line 3923 of file [catch.h](#).

6.31.3 Member Function Documentation

6.31.3.1 what()

```
const char * Catch::GeneratorException::what ( ) const [final], [override], [noexcept]
```

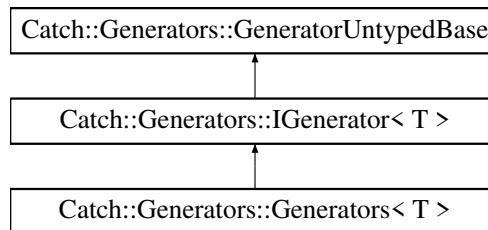
The documentation for this class was generated from the following file:

- /Users/samanthapope/msdscriptRepo/msdScript/catch.h

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

```
#include <catch.h>
```

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



Public Member Functions

- `template<typename... Gs>`
`Generators (Gs &&... moreGenerators)`
- `T const & get () const override`
- `bool next () override`

Public Member Functions inherited from `Catch::Generators::IGenerator< T >`

- `virtual ~IGenerator ()=default`

Public Member Functions inherited from `Catch::Generators::GeneratorUntypedBase`

- `GeneratorUntypedBase ()=default`
- `virtual ~GeneratorUntypedBase ()`

Additional Inherited Members

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

- `using type = T`

6.32.1 Detailed Description

```
template<typename T>
class Catch::Generators::Generators< T >
```

Definition at line 4010 of file [catch.h](#).

6.32.2 Constructor & Destructor Documentation

6.32.2.1 Generators()

```
template<typename T >
template<typename... Gs>
Catch::Generators::Generators< T >::Generators (
    Gs &&... moreGenerators ) [inline]
```

Definition at line 4032 of file [catch.h](#).

6.32.3 Member Function Documentation

6.32.3.1 get()

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

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

Definition at line 4037 of file [catch.h](#).

6.32.3.2 next()

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

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

Definition at line 4041 of file [catch.h](#).

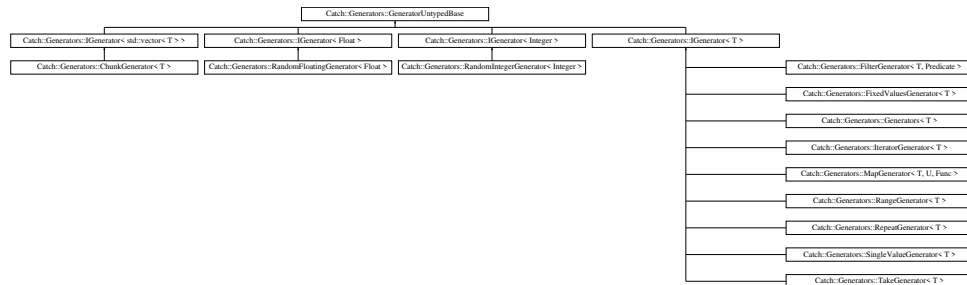
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.33 Catch::Generators::GeneratorUntypedBase Class Reference

```
#include <catch.h>
```

Inheritance diagram for Catch::Generators::GeneratorUntypedBase:



Public Member Functions

- [GeneratorUntypedBase\(\)](#)=default
- [virtual ~GeneratorUntypedBase\(\)](#)
- [virtual bool next\(\)](#)=0

6.33.1 Detailed Description

Definition at line 3845 of file [catch.h](#).

6.33.2 Constructor & Destructor Documentation

6.33.2.1 GeneratorUntypedBase()

```
Catch::Generators::GeneratorUntypedBase::GeneratorUntypedBase ( ) [default]
```

6.33.2.2 ~GeneratorUntypedBase()

```
virtual Catch::Generators::GeneratorUntypedBase::~~GeneratorUntypedBase ( ) [virtual]
```

6.33.3 Member Function Documentation

6.33.3.1 next()

```
virtual bool Catch::Generators::GeneratorUntypedBase::next ( ) [pure virtual]
```

Implemented in [Catch::Generators::SingleValueGenerator< T >](#), [Catch::Generators::FixedValuesGenerator< T >](#), [Catch::Generators::Generators< T >](#), [Catch::Generators::TakeGenerator< T >](#), [Catch::Generators::FilterGenerator< T, Predicate >](#), [Catch::Generators::RepeatGenerator< T >](#), [Catch::Generators::MapGenerator< T, U, Func >](#), [Catch::Generators::ChunkGenerator< T >](#), [Catch::Generators::RandomFloatingGenerator< Float >](#), [Catch::Generators::RandomIntegerGenerator< Integer >](#), [Catch::Generators::RangeGenerator< T >](#), and [Catch::Generators::IteratorGenerator< T >](#).

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

Public Member Functions

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

6.34.1 Detailed Description

```
template<typename T>  
class Catch::Generators::GeneratorWrapper< T >
```

Definition at line [3986](#) of file [catch.h](#).

6.34.2 Constructor & Destructor Documentation

6.34.2.1 GeneratorWrapper()

```
template<typename T >  
Catch::Generators::GeneratorWrapper< T >::GeneratorWrapper (  
    std::unique_ptr< IGenerator< T > > generator ) [inline]
```

Definition at line [3989](#) of file [catch.h](#).

6.34.3 Member Function Documentation

6.34.3.1 get()

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

Definition at line [3992](#) of file [catch.h](#).

6.34.3.2 next()

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

Definition at line [3995](#) of file [catch.h](#).

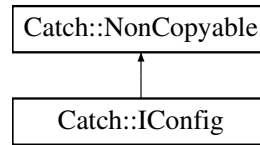
The documentation for this class was generated from the following file:

- /Users/samanthapope/msdscriptRepo/msdScript/[catch.h](#)

6.35 Catch::IConfig Struct Reference

```
#include <catch.h>
```

Inheritance diagram for Catch::IConfig:



Public Member Functions

- [virtual ~IConfig \(\)](#)
- [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](#)

Additional Inherited Members

Protected Member Functions inherited from [Catch::NonCopyable](#)

- [NonCopyable \(\)](#)
- [virtual ~NonCopyable \(\)](#)

6.35.1 Detailed Description

Definition at line [4512](#) of file [catch.h](#).

6.35.2 Constructor & Destructor Documentation

6.35.2.1 ~IConfig()

```
virtual Catch::IConfig::~~IConfig ( ) [virtual]
```

6.35.3 Member Function Documentation

6.35.3.1 abortAfter()

```
virtual int Catch::IConfig::abortAfter ( ) const [pure virtual]
```

6.35.3.2 allowThrows()

```
virtual bool Catch::IConfig::allowThrows ( ) const [pure virtual]
```

6.35.3.3 benchmarkConfidenceInterval()

```
virtual double Catch::IConfig::benchmarkConfidenceInterval ( ) const [pure virtual]
```

6.35.3.4 benchmarkNoAnalysis()

```
virtual bool Catch::IConfig::benchmarkNoAnalysis ( ) const [pure virtual]
```

6.35.3.5 benchmarkResamples()

```
virtual unsigned int Catch::IConfig::benchmarkResamples ( ) const [pure virtual]
```

6.35.3.6 benchmarkSamples()

```
virtual int Catch::IConfig::benchmarkSamples ( ) const [pure virtual]
```

6.35.3.7 benchmarkWarmupTime()

```
virtual std::chrono::milliseconds Catch::IConfig::benchmarkWarmupTime ( ) const [pure virtual]
```

6.35.3.8 getSectionsToRun()

```
virtual std::vector< std::string > const & Catch::IConfig::getSectionsToRun ( ) const [pure virtual]
```

6.35.3.9 getTestsOrTags()

```
virtual std::vector< std::string > const & Catch::IConfig::getTestsOrTags ( ) const [pure virtual]
```

6.35.3.10 hasTestFilters()

```
virtual bool Catch::IConfig::hasTestFilters ( ) const [pure virtual]
```

6.35.3.11 includeSuccessfulResults()

```
virtual bool Catch::IConfig::includeSuccessfulResults ( ) const [pure virtual]
```

6.35.3.12 minDuration()

```
virtual double Catch::IConfig::minDuration ( ) const [pure virtual]
```

6.35.3.13 name()

```
virtual std::string Catch::IConfig::name ( ) const [pure virtual]
```

6.35.3.14 rngSeed()

```
virtual unsigned int Catch::IConfig::rngSeed ( ) const [pure virtual]
```

6.35.3.15 runOrder()

```
virtual RunTests::InWhatOrder Catch::IConfig::runOrder ( ) const [pure virtual]
```

6.35.3.16 shouldDebugBreak()

```
virtual bool Catch::IConfig::shouldDebugBreak ( ) const [pure virtual]
```

6.35.3.17 showDurations()

```
virtual ShowDurations::OrNot Catch::IConfig::showDurations ( ) const [pure virtual]
```

6.35.3.18 showInvisibles()

```
virtual bool Catch::IConfig::showInvisibles ( ) const [pure virtual]
```

6.35.3.19 stream()

```
virtual std::ostream & Catch::IConfig::stream ( ) const [pure virtual]
```

6.35.3.20 testSpec()

```
virtual TestSpec const & Catch::IConfig::testSpec ( ) const [pure virtual]
```

6.35.3.21 useColour()

```
virtual UseColour::YesOrNo Catch::IConfig::useColour ( ) const [pure virtual]
```

6.35.3.22 verbosity()

```
virtual Verbosity Catch::IConfig::verbosity ( ) const [pure virtual]
```

6.35.3.23 warnAboutMissingAssertions()

```
virtual bool Catch::IConfig::warnAboutMissingAssertions ( ) const [pure virtual]
```

6.35.3.24 warnAboutNoTests()

```
virtual bool Catch::IConfig::warnAboutNoTests ( ) const [pure virtual]
```

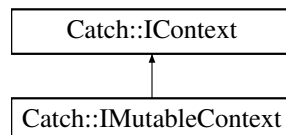
The documentation for this struct was generated from the following file:

- /Users/samanthapope/msdscripRepo/msdScript/[catch.h](#)

6.36 Catch::IContext Struct Reference

```
#include <catch.h>
```

Inheritance diagram for Catch::IContext:

**Public Member Functions**

- [virtual ~IContext \(\)](#)
- [virtual IResultCapture * getResultCapture \(\)=0](#)
- [virtual IRunner * getRunner \(\)=0](#)
- [virtual IConfigPtr const & getConfig \(\) const =0](#)

6.36.1 Detailed Description

Definition at line 4358 of file [catch.h](#).

6.36.2 Constructor & Destructor Documentation

6.36.2.1 ~IContext()

```
virtual Catch::IContext::~~IContext ( ) [virtual]
```

6.36.3 Member Function Documentation

6.36.3.1 getConfig()

```
virtual IConfigPtr const & Catch::IContext::getConfig ( ) const [pure virtual]
```

6.36.3.2 getResultCapture()

```
virtual IResultCapture * Catch::IContext::getResultCapture ( ) [pure virtual]
```

6.36.3.3 getRunner()

```
virtual IRunner * Catch::IContext::getRunner ( ) [pure virtual]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.37 Catch::IExceptionTranslator Struct Reference

```
#include <catch.h>
```

Public Member Functions

- [virtual ~IExceptionTranslator \(\)](#)
- [virtual std::string translate \(ExceptionTranslators::const_iterator it, ExceptionTranslators::const_iterator itEnd\) const =0](#)

6.37.1 Detailed Description

Definition at line 3010 of file [catch.h](#).

6.37.2 Constructor & Destructor Documentation

6.37.2.1 ~IExceptionTranslator()

```
virtual Catch::IExceptionTranslator::~IExceptionTranslator ( ) [virtual]
```

6.37.3 Member Function Documentation

6.37.3.1 translate()

```
virtual std::string Catch::IExceptionTranslator::translate (
    ExceptionTranslators::const_iterator it,
    ExceptionTranslators::const_iterator itEnd ) const [pure virtual]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.38 Catch::IExceptionTranslatorRegistry Struct Reference

```
#include <catch.h>
```

Public Member Functions

- [virtual ~IExceptionTranslatorRegistry \(\)](#)
- [virtual std::string translateActiveException \(\) const =0](#)

6.38.1 Detailed Description

Definition at line [3015](#) of file [catch.h](#).

6.38.2 Constructor & Destructor Documentation

6.38.2.1 ~IExceptionTranslatorRegistry()

```
virtual Catch::IExceptionTranslatorRegistry::~IExceptionTranslatorRegistry ( ) [virtual]
```

6.38.3 Member Function Documentation

6.38.3.1 translateActiveException()

```
virtual std::string Catch::IExceptionTranslatorRegistry::translateActiveException ( ) const
[pure virtual]
```

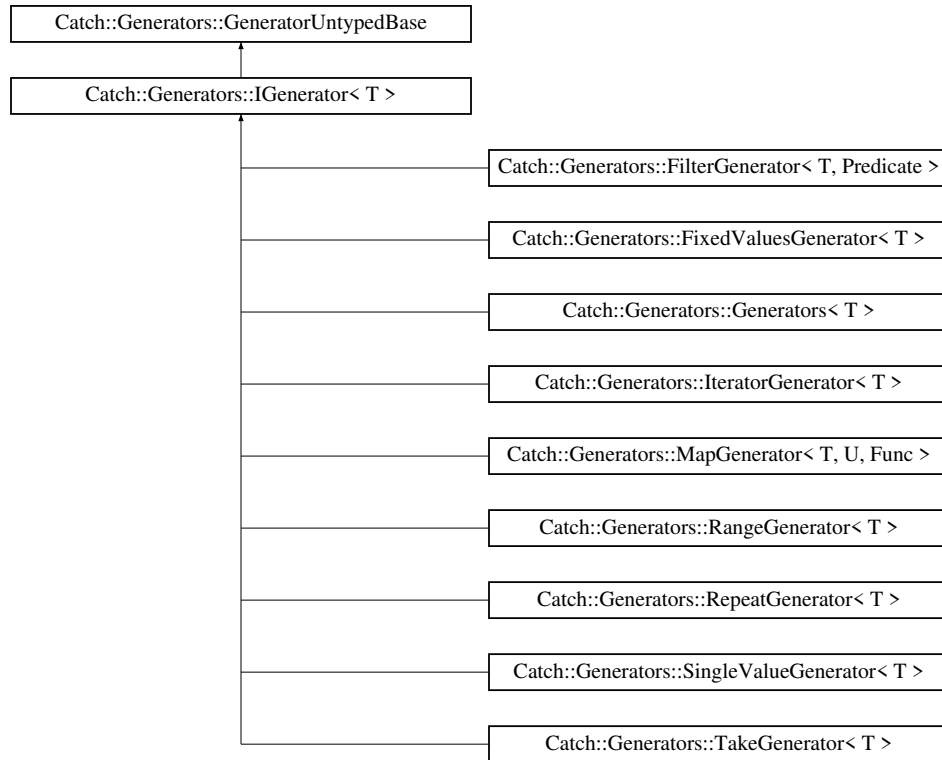
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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



Public Types

- `using type = T`

Public Member Functions

- `virtual ~IGenerator ()=default`
- `virtual T const & get () const =0`

Public Member Functions inherited from `Catch::Generators::GeneratorUntypedBase`

- `GeneratorUntypedBase ()=default`
- `virtual ~GeneratorUntypedBase ()`
- `virtual bool next ()=0`

6.39.1 Detailed Description

```
template<typename T>
struct Catch::Generators::IGenerator< T >
```

Definition at line 3941 of file `catch.h`.

6.39.2 Member Typedef Documentation

6.39.2.1 type

```
template<typename T >
using Catch::Generators::IGenerator< T >::type = T
```

Definition at line 3949 of file [catch.h](#).

6.39.3 Constructor & Destructor Documentation

6.39.3.1 ~IGenerator()

```
template<typename T >
virtual Catch::Generators::IGenerator< T >::~~IGenerator ( ) [virtual], [default]
```

6.39.4 Member Function Documentation

6.39.4.1 get()

```
template<typename T >
virtual T const & Catch::Generators::IGenerator< T >::get ( ) const [pure virtual]
```

Implemented in [Catch::Generators::SingleValueGenerator< T >](#), [Catch::Generators::FixedValuesGenerator< T >](#), [Catch::Generators::Generators< T >](#), [Catch::Generators::TakeGenerator< T >](#), [Catch::Generators::FilterGenerator< T, Predicate >](#), [Catch::Generators::RepeatGenerator< T >](#), [Catch::Generators::MapGenerator< T, U, Func >](#), [Catch::Generators::ChunkGenerator< T >](#), [Catch::Generators::RandomFloatingGenerator< Float >](#), [Catch::Generators::RandomIntegerGenerator< Integer >](#), [Catch::Generators::RangeGenerator< T >](#), and [Catch::Generators::IteratorGenerator< T >](#).

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.40 Catch::IGeneratorTracker Struct Reference

```
#include <catch.h>
```

Public Member Functions

- [virtual ~IGeneratorTracker \(\)](#)
- [virtual auto hasGenerator \(\) const -> bool=0](#)
- [virtual auto getGenerator \(\) const -> Generators::GeneratorBasePtr const &=0](#)
- [virtual void setGenerator \(Generators::GeneratorBasePtr &&generator\)=0](#)

6.40.1 Detailed Description

Definition at line 3859 of file [catch.h](#).

6.40.2 Constructor & Destructor Documentation

6.40.2.1 ~IGeneratorTracker()

```
virtual Catch::IGeneratorTracker::~~IGeneratorTracker ( ) [virtual]
```

6.40.3 Member Function Documentation

6.40.3.1 getGenerator()

```
virtual auto Catch::IGeneratorTracker::getGenerator ( ) const -> Generators::GeneratorBasePtr
const & [pure virtual]
```

6.40.3.2 hasGenerator()

```
virtual auto Catch::IGeneratorTracker::hasGenerator ( ) const -> bool [pure virtual]
```

6.40.3.3 setGenerator()

```
virtual void Catch::IGeneratorTracker::setGenerator (
    Generators::GeneratorBasePtr && generator ) [pure virtual]
```

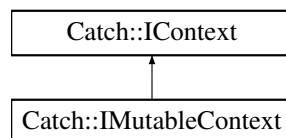
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.41 Catch::IMutableContext Struct Reference

```
#include <catch.h>
```

Inheritance diagram for Catch::IMutableContext:



Public Member Functions

- `virtual ~IMutableContext ()`
- `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 ~IContext \(\)](#)
- [virtual IResultCapture * getResultCapture \(\)=0](#)
- [virtual IRunner * getRunner \(\)=0](#)
- [virtual IConfigPtr const & getConfig \(\) const =0](#)

Friends

- [IMutableContext & getCurrentMutableContext \(\)](#)
- [void cleanUpContext \(\)](#)

6.41.1 Detailed Description

Definition at line 4367 of file [catch.h](#).

6.41.2 Constructor & Destructor Documentation

6.41.2.1 ~IMutableContext()

```
virtual Catch::IMutableContext::~~IMutableContext ( ) [virtual]
```

6.41.3 Member Function Documentation

6.41.3.1 setConfig()

```
virtual void Catch::IMutableContext::setConfig (
    IConfigPtr const & config ) [pure virtual]
```

6.41.3.2 setResultCapture()

```
virtual void Catch::IMutableContext::setResultCapture (
    IResultCapture * resultCapture ) [pure virtual]
```

6.41.3.3 setRunner()

```
virtual void Catch::IMutableContext::setRunner (
    IRunner * runner ) [pure virtual]
```

6.41.4 Friends And Related Symbol Documentation

6.41.4.1 cleanUpContext

```
void cleanUpContext ( ) [friend]
```

6.41.4.2 getCurrentMutableContext

```
ImmutableContext & getCurrentMutableContext ( ) [friend]
```

Definition at line 4381 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.42 Catch::ImmutableEnumValuesRegistry Struct Reference

```
#include <catch.h>
```

Public Member Functions

- [virtual ~ImmutableEnumValuesRegistry \(\)](#)
- [virtual Detail::EnumInfo const & registerEnum \(StringRef enumName, StringRef allEnums, std::vector< int > const &values\)=0](#)
- [template<typename E > Detail::EnumInfo const & registerEnum \(StringRef enumName, StringRef allEnums, std::initializer_list< E > values\)](#)

6.42.1 Detailed Description

Definition at line 1476 of file [catch.h](#).

6.42.2 Constructor & Destructor Documentation

6.42.2.1 ~ImmutableEnumValuesRegistry()

```
virtual Catch::ImmutableEnumValuesRegistry::~ImmutableEnumValuesRegistry ( ) [virtual]
```

6.42.3 Member Function Documentation

6.42.3.1 registerEnum() [1/2]

```
template<typename E >
Detail::EnumInfo const & Catch::ImmutableEnumValuesRegistry::registerEnum (
    StringRef enumName,
    StringRef allEnums,
    std::initializer_list< E > values ) [inline]
```

Definition at line 1482 of file [catch.h](#).

6.42.3.2 registerEnum() [2/2]

```
virtual Detail::EnumInfo const & Catch::IMutableEnumValuesRegistry::registerEnum (
    StringRef enumName,
    StringRef allEnums,
    std::vector< int > const & values ) [pure virtual]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.43 Catch::IMutableRegistryHub Struct Reference

```
#include <catch.h>
```

Public Member Functions

- [virtual ~IMutableRegistryHub \(\)](#)
- [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](#)

6.43.1 Detailed Description

Definition at line 2976 of file [catch.h](#).

6.43.2 Constructor & Destructor Documentation

6.43.2.1 ~IMutableRegistryHub()

```
virtual Catch::IMutableRegistryHub::~IMutableRegistryHub ( ) [virtual]
```

6.43.3 Member Function Documentation

6.43.3.1 getMutableEnumValuesRegistry()

```
virtual IMutableEnumValuesRegistry & Catch::IMutableRegistryHub::getMutableEnumValuesRegistry
( ) [pure virtual]
```


6.43.3.2 registerListener()

```
virtual void Catch::IMutableRegistryHub::registerListener (
    IReporterFactoryPtr const & factory ) [pure virtual]
```

6.43.3.3 registerReporter()

```
virtual void Catch::IMutableRegistryHub::registerReporter (
    std::string const & name,
    IReporterFactoryPtr const & factory ) [pure virtual]
```

6.43.3.4 registerStartupException()

```
virtual void Catch::IMutableRegistryHub::registerStartupException ( ) [pure virtual], [noexcept]
```

6.43.3.5 registerTagAlias()

```
virtual void Catch::IMutableRegistryHub::registerTagAlias (
    std::string const & alias,
    std::string const & tag,
    SourceLineInfo const & lineInfo ) [pure virtual]
```

6.43.3.6 registerTest()

```
virtual void Catch::IMutableRegistryHub::registerTest (
    TestCase const & testInfo ) [pure virtual]
```

6.43.3.7 registerTranslator()

```
virtual void Catch::IMutableRegistryHub::registerTranslator (
    const IExceptionTranslator * translator ) [pure virtual]
```

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

- /Users/samanthapope/msdscriptRepo/msdScript/[catch.h](#)

6.44 Catch::IRegistryHub Struct Reference

```
#include <catch.h>
```

Public Member Functions

- [virtual ~IRegistryHub \(\)](#)
- [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](#)

6.44.1 Detailed Description

Definition at line 2965 of file [catch.h](#).

6.44.2 Constructor & Destructor Documentation

6.44.2.1 ~IRegistryHub()

```
virtual Catch::IRegistryHub::~IRegistryHub ( ) [virtual]
```

6.44.3 Member Function Documentation

6.44.3.1 getExceptionTranslatorRegistry()

```
virtual IExceptionTranslatorRegistry const & Catch::IRegistryHub::getExceptionTranslatorRegistry ( ) const [pure virtual]
```

6.44.3.2 getReporterRegistry()

```
virtual IReporterRegistry const & Catch::IRegistryHub::getReporterRegistry ( ) const [pure virtual]
```

6.44.3.3 getStartupExceptionRegistry()

```
virtual StartupExceptionRegistry const & Catch::IRegistryHub::getStartupExceptionRegistry ( ) const [pure virtual]
```

6.44.3.4 getTagAliasRegistry()

```
virtual ITagAliasRegistry const & Catch::IRegistryHub::getTagAliasRegistry ( ) const [pure virtual]
```

6.44.3.5 getTestCaseRegistry()

```
virtual ITestCaseRegistry const & Catch::IRegistryHub::getTestCaseRegistry ( ) const [pure virtual]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.45 Catch::IResultCapture Struct Reference

```
#include <catch.h>
```

Public Member Functions

- [virtual ~IResultCapture \(\)](#)
- [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](#)

6.45.1 Detailed Description

Definition at line 2458 of file [catch.h](#).

6.45.2 Constructor & Destructor Documentation

6.45.2.1 ~IResultCapture()

```
virtual Catch::IResultCapture::~~IResultCapture ( ) [virtual]
```

6.45.3 Member Function Documentation

6.45.3.1 acquireGeneratorTracker()

```
virtual auto Catch::IResultCapture::acquireGeneratorTracker (
    StringRef generatorName,
    SourceLineInfo const & lineInfo ) -> IGeneratorTracker & [pure virtual]
```

6.45.3.2 assertionPassed()

```
virtual void Catch::IResultCapture::assertionPassed ( ) [pure virtual]
```

6.45.3.3 emplaceUnscopedMessage()

```
virtual void Catch::IResultCapture::emplaceUnscopedMessage (
    MessageBuilder const & builder ) [pure virtual]
```

6.45.3.4 exceptionEarlyReported()

```
virtual void Catch::IResultCapture::exceptionEarlyReported ( ) [pure virtual]
```

6.45.3.5 getCurrentTestName()

```
virtual std::string Catch::IResultCapture::getCurrentTestName ( ) const [pure virtual]
```

6.45.3.6 getLastResult()

```
virtual const AssertionResult * Catch::IResultCapture::getLastResult ( ) const [pure virtual]
```

6.45.3.7 handleExpr()

```
virtual void Catch::IResultCapture::handleExpr (
    AssertionInfo const & info,
    ITransientExpression const & expr,
    AssertionReaction & reaction ) [pure virtual]
```

6.45.3.8 handleFatalErrorCondition()

```
virtual void Catch::IResultCapture::handleFatalErrorCondition (
    StringRef message ) [pure virtual]
```

6.45.3.9 handleIncomplete()

```
virtual void Catch::IResultCapture::handleIncomplete (
    AssertionInfo const & info ) [pure virtual]
```

6.45.3.10 handleMessage()

```
virtual void Catch::IResultCapture::handleMessage (
    AssertionInfo const & info,
    ResultWas::OfType resultType,
    StringRef const & message,
    AssertionReaction & reaction ) [pure virtual]
```

6.45.3.11 handleNonExpr()

```
virtual void Catch::IResultCapture::handleNonExpr (
    AssertionInfo const & info,
    ResultWas::OfType resultType,
    AssertionReaction & reaction ) [pure virtual]
```

6.45.3.12 handleUnexpectedExceptionNotThrown()

```
virtual void Catch::IResultCapture::handleUnexpectedExceptionNotThrown (
    AssertionInfo const & info,
    AssertionReaction & reaction ) [pure virtual]
```

6.45.3.13 handleUnexpectedInflightException()

```
virtual void Catch::IResultCapture::handleUnexpectedInflightException (
    AssertionInfo const & info,
    std::string const & message,
    AssertionReaction & reaction ) [pure virtual]
```

6.45.3.14 lastAssertionPassed()

```
virtual bool Catch::IResultCapture::lastAssertionPassed ( ) [pure virtual]
```

6.45.3.15 popScopedMessage()

```
virtual void Catch::IResultCapture::popScopedMessage (
    MessageInfo const & message ) [pure virtual]
```

6.45.3.16 pushScopedMessage()

```
virtual void Catch::IResultCapture::pushScopedMessage (
    MessageInfo const & message ) [pure virtual]
```

6.45.3.17 sectionEnded()

```
virtual void Catch::IResultCapture::sectionEnded (
    SectionEndInfo const & endInfo ) [pure virtual]
```

6.45.3.18 sectionEndedEarly()

```
virtual void Catch::IResultCapture::sectionEndedEarly (
    SectionEndInfo const & endInfo ) [pure virtual]
```

6.45.3.19 sectionStarted()

```
virtual bool Catch::IResultCapture::sectionStarted (
    SectionInfo const & sectionInfo,
    Counts & assertions ) [pure virtual]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.46 Catch::IRunner Struct Reference

```
#include <catch.h>
```

Public Member Functions

- [virtual ~IRunner\(\)](#)
- [virtual bool aborting\(\) const](#) =0

6.46.1 Detailed Description

Definition at line [4840](#) of file [catch.h](#).

6.46.2 Constructor & Destructor Documentation**6.46.2.1 ~IRunner()**

```
virtual Catch::IRunner::~~IRunner ( ) [virtual]
```

6.46.3 Member Function Documentation

6.46.3.1 aborting()

```
virtual bool Catch::IRunner::aborting ( ) const [pure virtual]
```

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

- /Users/samanthapope/msdscriptRepo/msdScript/[catch.h](#)

6.47 Catch::is_callable< T > Struct Template Reference

6.47.1 Detailed Description

```
template<typename T>  
struct Catch::is_callable< T >
```

Definition at line 936 of file [catch.h](#).

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

- /Users/samanthapope/msdscriptRepo/msdScript/[catch.h](#)

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

```
#include <catch.h>
```

6.48.1 Detailed Description

```
template<typename Fun, typename... Args>  
struct Catch::is_callable< Fun(Args...)>
```

Definition at line 939 of file [catch.h](#).

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

- /Users/samanthapope/msdscriptRepo/msdScript/[catch.h](#)

6.49 Catch::is_callable_tester Struct Reference

```
#include <catch.h>
```

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 (...)`

6.49.1 Detailed Description

Definition at line 928 of file [catch.h](#).

6.49.2 Member Function Documentation

6.49.2.1 test() [1/2]

```
template<typename... >
static std::false_type Catch::is_callable_tester::test (
    ... ) [static]
```

6.49.2.2 test() [2/2]

```
template<typename Fun , typename... Args>
static true_given< decltype(std::declval< Fun >() (std::declval< Args >()...))> Catch::is_←
callable_tester::test (
    int ) [static]
```

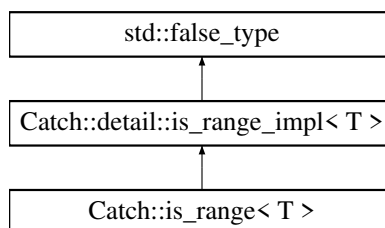
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.50 Catch::is_range< T > Struct Template Reference

```
#include <catch.h>
```

Inheritance diagram for `Catch::is_range< T >`:



6.50.1 Detailed Description

```
template<typename T>
struct Catch::is_range< T >
```

Definition at line 2007 of file [catch.h](#).

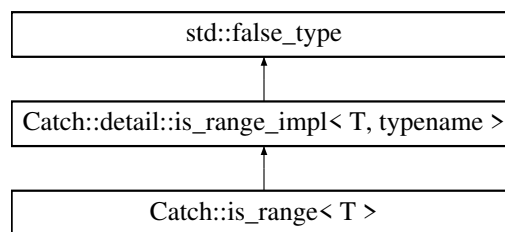
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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



6.51.1 Detailed Description

```
template<typename T, typename = void>
struct Catch::detail::is_range_impl< T, typename >
```

Definition at line 1998 of file [catch.h](#).

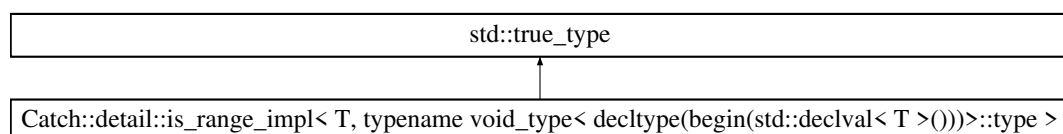
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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



6.52.1 Detailed Description

```
template<typename T>
struct Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >()))>::type
>
```

Definition at line 2002 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.53 Catch::Detail::IsStreamInsertable< T > Class Template Reference

```
#include <catch.h>
```

Static Public Attributes

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

6.53.1 Detailed Description

```
template<typename T>
class Catch::Detail::IsStreamInsertable< T >
```

Definition at line 1564 of file [catch.h](#).

6.53.2 Member Data Documentation

6.53.2.1 value

```
template<typename T >
const bool Catch::Detail::IsStreamInsertable< T >::value = decltype(test<std::ostream, const
T&>(0))::value [static]
```

Definition at line 1573 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.54 Catch::IStream Struct Reference

```
#include <catch.h>
```

Public Member Functions

- [virtual ~IStream \(\)](#)
- [virtual std::ostream & stream \(\) const =0](#)

6.54.1 Detailed Description

Definition at line [1433](#) of file [catch.h](#).

6.54.2 Constructor & Destructor Documentation**6.54.2.1 ~IStream()**

```
virtual Catch::IStream::~~IStream ( ) [virtual]
```

6.54.3 Member Function Documentation**6.54.3.1 stream()**

```
virtual std::ostream & Catch::IStream::stream ( ) const [pure virtual]
```

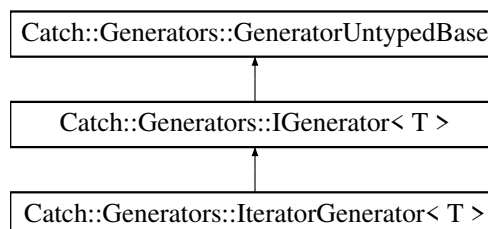
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.55 Catch::Generators::IteratorGenerator< T > Class Template Reference

```
#include <catch.h>
```

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](#)

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

- [virtual ~IGenerator \(\)=default](#)

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

- [GeneratorUntypedBase \(\)=default](#)
- [virtual ~GeneratorUntypedBase \(\)](#)

Additional Inherited Members

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

- [using type = T](#)

6.55.1 Detailed Description

```
template<typename T>
class Catch::Generators::IteratorGenerator< T >
```

Definition at line [4712](#) of file [catch.h](#).

6.55.2 Constructor & Destructor Documentation

6.55.2.1 IteratorGenerator()

```
template<typename T >
template<typename InputIterator , typename InputSentinel >
Catch::Generators::IteratorGenerator< T >::IteratorGenerator (
    InputIterator first,
    InputSentinel last ) [inline]
```

Definition at line [4721](#) of file [catch.h](#).

6.55.3 Member Function Documentation

6.55.3.1 get()

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

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

Definition at line [4727](#) of file [catch.h](#).

6.55.3.2 next()

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

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

Definition at line [4731](#) of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.56 Catch::ITestCaseRegistry Struct Reference

```
#include <catch.h>
```

Public Member Functions

- [virtual ~ITestCaseRegistry \(\)](#)
- [virtual std::vector< TestCase > const & getAllTests \(\) const =0](#)
- [virtual std::vector< TestCase > const & getAllTestsSorted \(IConfig const &config\) const =0](#)

6.56.1 Detailed Description

Definition at line [580](#) of file [catch.h](#).

6.56.2 Constructor & Destructor Documentation

6.56.2.1 ~ITestCaseRegistry()

```
virtual Catch::ITestCaseRegistry::~~ITestCaseRegistry ( ) [virtual]
```

6.56.3 Member Function Documentation

6.56.3.1 getAllTests()

```
virtual std::vector< TestCase > const & Catch::ITestCaseRegistry::getAllTests ( ) const [pure virtual]
```

6.56.3.2 getAllTestsSorted()

```
virtual std::vector< TestCase > const & Catch::ITestCaseRegistry::getAllTestsSorted (
    IConfig const & config ) const [pure virtual]
```

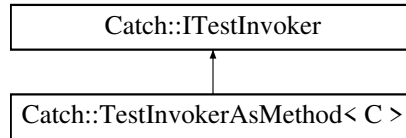
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.57 Catch::ITestInvoker Struct Reference

```
#include <catch.h>
```

Inheritance diagram for Catch::ITestInvoker:



Public Member Functions

- [virtual void invoke \(\) const =0](#)
- [virtual ~ITestInvoker \(\)](#)

6.57.1 Detailed Description

Definition at line [572](#) of file [catch.h](#).

6.57.2 Constructor & Destructor Documentation

6.57.2.1 ~ITestInvoker()

```
virtual Catch::ITestInvoker::~~ITestInvoker ( ) [virtual]
```

6.57.3 Member Function Documentation

6.57.3.1 invoke()

```
virtual void Catch::ITestInvoker::invoke ( ) const [pure virtual]
```

Implemented in [Catch::TestInvokerAsMethod< C >](#).

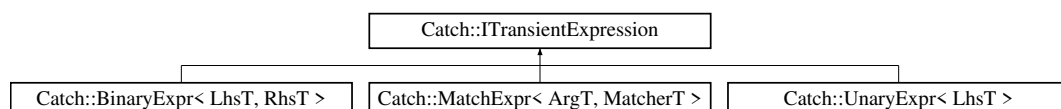
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.58 Catch::ITransientExpression Struct Reference

```
#include <catch.h>
```

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)`
- `virtual ~ITransientExpression ()`

Public Attributes

- `bool m_isBinaryExpression`
- `bool m_result`

6.58.1 Detailed Description

Definition at line 2202 of file [catch.h](#).

6.58.2 Constructor & Destructor Documentation

6.58.2.1 ITransientExpression()

```
Catch::ITransientExpression::ITransientExpression (
    bool isBinaryExpression,
    bool result ) [inline]
```

Definition at line 2207 of file [catch.h](#).

6.58.2.2 ~ITransientExpression()

```
virtual Catch::ITransientExpression::~~ITransientExpression ( ) [virtual]
```

6.58.3 Member Function Documentation

6.58.3.1 getResult()

```
auto Catch::ITransientExpression::getResult ( ) const -> bool [inline]
```

Definition at line 2204 of file [catch.h](#).

6.58.3.2 isBinaryExpression()

```
auto Catch::ITransientExpression::isBinaryExpression ( ) const -> bool [inline]
```

Definition at line 2203 of file [catch.h](#).

6.58.3.3 streamReconstructedExpression()

```
virtual void Catch::ITransientExpression::streamReconstructedExpression (
    std::ostream & os ) const [pure virtual]
```

Implemented in [Catch::MatchExpr< ArgT, MatcherT >](#).

6.58.4 Member Data Documentation

6.58.4.1 m_isBinaryExpression

```
bool Catch::ITransientExpression::m_isBinaryExpression
```

Definition at line 2216 of file [catch.h](#).

6.58.4.2 m_result

```
bool Catch::ITransientExpression::m_result
```

Definition at line 2217 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.59 Catch::LazyExpression Class Reference

```
#include <catch.h>
```

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 &

6.59.1 Detailed Description

Definition at line 2526 of file [catch.h](#).

6.59.2 Constructor & Destructor Documentation

6.59.2.1 LazyExpression() [1/2]

```
Catch::LazyExpression::LazyExpression (
    bool isNegated )
```

6.59.2.2 LazyExpression() [2/2]

```
Catch::LazyExpression::LazyExpression (
    LazyExpression const & other )
```

6.59.3 Member Function Documentation

6.59.3.1 operator bool()

```
Catch::LazyExpression::operator bool ( ) const [explicit]
```

6.59.3.2 operator=()

```
LazyExpression & Catch::LazyExpression::operator= (
    LazyExpression const & ) [delete]
```

6.59.4 Friends And Related Symbol Documentation

6.59.4.1 AssertionHandler

```
friend class AssertionHandler [friend]
```

Definition at line 2527 of file [catch.h](#).

6.59.4.2 AssertionStats

```
friend struct AssertionStats [friend]
```

Definition at line 2528 of file [catch.h](#).

6.59.4.3 operator<<

```
auto operator<< (
    std::ostream & os,
    LazyExpression const & lazyExpr ) -> std::ostream & [friend]
```

6.59.4.4 RunContext

```
friend class RunContext [friend]
```

Definition at line 2529 of file [catch.h](#).

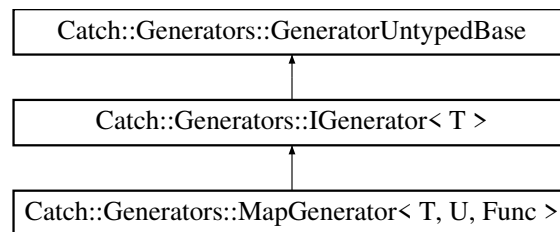
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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`

Public Member Functions inherited from `Catch::Generators::IGenerator< T >`

- `virtual ~IGenerator ()=default`

Public Member Functions inherited from `Catch::Generators::GeneratorUntypedBase`

- `GeneratorUntypedBase ()=default`
- `virtual ~GeneratorUntypedBase ()`

Additional Inherited Members

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

- `using type = T`

6.60.1 Detailed Description

```
template<typename T, typename U, typename Func>
class Catch::Generators::MapGenerator< T, U, Func >
```

Definition at line 4256 of file [catch.h](#).

6.60.2 Constructor & Destructor Documentation

6.60.2.1 MapGenerator()

```
template<typename T , typename U , typename Func >
template<typename F2 = Func>
Catch::Generators::MapGenerator< T, U, Func >::MapGenerator (
    F2 && function,
    GeneratorWrapper< U > && generator ) [inline]
```

Definition at line 4264 of file [catch.h](#).

6.60.3 Member Function Documentation

6.60.3.1 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 >](#).

Definition at line 4270 of file [catch.h](#).

6.60.3.2 next()

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

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

Definition at line 4273 of file [catch.h](#).

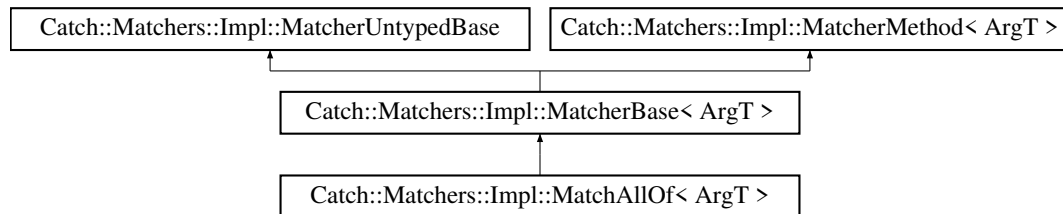
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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 \(\)=default](#)
- [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 Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [virtual ~MatcherUntypedBase \(\)](#)

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

- std::string [m_cachedToString](#)

6.61.1 Detailed Description

```
template<typename ArgT>
struct Catch::Matchers::Impl::MatchAllOf< ArgT >
```

Definition at line 3288 of file [catch.h](#).

6.61.2 Member Function Documentation

6.61.2.1 describe()

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

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

Definition at line 3296 of file [catch.h](#).

6.61.2.2 match()

```
template<typename ArgT >
bool Catch::Matchers::Impl::MatchAllOf< ArgT >::match (
    ArgT const & arg ) const [inline], [override]
```

Definition at line 3289 of file [catch.h](#).

6.61.2.3 operator&&()

```
template<typename ArgT >
MatchAllOf< ArgT > Catch::Matchers::Impl::MatchAllOf< ArgT >::operator&& (
    MatcherBase< ArgT > const & other ) [inline]
```

Definition at line 3312 of file [catch.h](#).

6.61.3 Member Data Documentation

6.61.3.1 m_matchers

```
template<typename ArgT >
std::vector<MatcherBase<ArgT> const*> Catch::Matchers::Impl::MatchAllOf< ArgT >::m_matchers
```

Definition at line 3318 of file [catch.h](#).

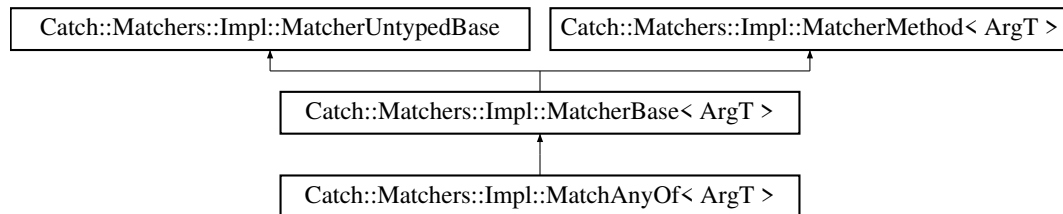
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.62 Catch::Matchers::Impl::MatchAnyOf< ArgT > Struct Template Reference

```
#include <catch.h>
```

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 \(\)](#)=default
- [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 Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [virtual ~MatcherUntypedBase \(\)](#)

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

- std::string [m_cachedToString](#)

6.62.1 Detailed Description

```
template<typename ArgT>
struct Catch::Matchers::Impl::MatchAnyOf< ArgT >
```

Definition at line 3321 of file [catch.h](#).

6.62.2 Member Function Documentation

6.62.2.1 describe()

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

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

Definition at line 3330 of file [catch.h](#).

6.62.2.2 match()

```
template<typename ArgT >
bool Catch::Matchers::Impl::MatchAnyOf< ArgT >::match (
    ArgT const & arg ) const [inline], [override]
```

Definition at line 3323 of file [catch.h](#).

6.62.2.3 operator" | " |()

```
template<typename ArgT >
MatchAnyOf< ArgT > Catch::Matchers::Impl::MatchAnyOf< ArgT >::operator|| (
    MatcherBase< ArgT > const & other ) [inline]
```

Definition at line 3346 of file [catch.h](#).

6.62.3 Member Data Documentation

6.62.3.1 m_matchers

```
template<typename ArgT >
std::vector<MatcherBase<ArgT> const*> Catch::Matchers::Impl::MatchAnyOf< ArgT >::m_matchers
```

Definition at line 3352 of file [catch.h](#).

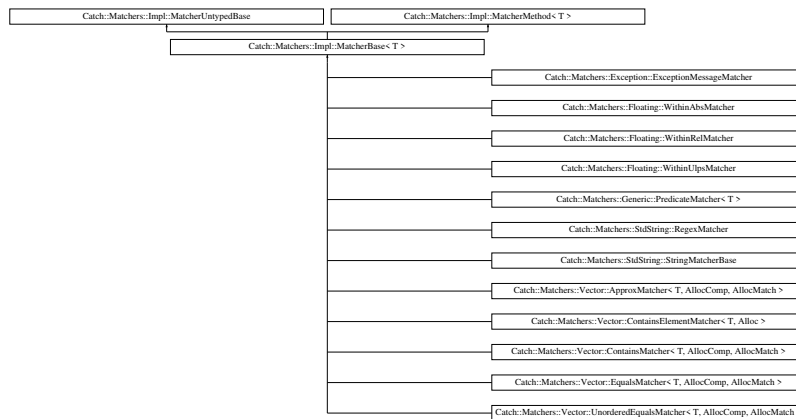
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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

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

- [std::string m_cachedToString](#)

6.63.1 Detailed Description

```
template<typename T>
struct Catch::Matchers::Impl::MatcherBase< T >
```

Definition at line 3280 of file [catch.h](#).

6.63.2 Member Function Documentation

6.63.2.1 operator"!")()

```
template<typename T >
MatchNotOf< T > Catch::Matchers::Impl::MatcherBase< T >::operator! ( ) const
```

Definition at line 3379 of file [catch.h](#).

6.63.2.2 operator&&()

```
template<typename T >
MatchAllOf< T > Catch::Matchers::Impl::MatcherBase< T >::operator&& (
    MatcherBase< T > const & other ) const
```

Definition at line 3371 of file [catch.h](#).

6.63.2.3 operator" | " | ()

```
template<typename T >
MatchAnyOf< T > Catch::Matchers::Impl::MatcherBase< T >::operator|| (
    MatcherBase< T > const & other ) const
```

Definition at line 3375 of file [catch.h](#).

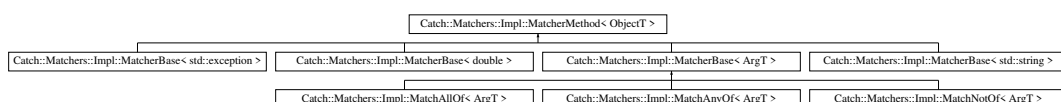
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.64 Catch::Matchers::Impl::MatcherMethod< ObjectT > Struct Template Reference

```
#include <catch.h>
```

Inheritance diagram for Catch::Matchers::Impl::MatcherMethod< ObjectT >:



Protected Attributes

- `std::string m_cachedToString`

6.65.1 Detailed Description

Definition at line 3243 of file [catch.h](#).

6.65.2 Constructor & Destructor Documentation

6.65.2.1 MatcherUntypedBase() [1/2]

```
Catch::Matchers::Impl::MatcherUntypedBase::MatcherUntypedBase ( ) [default]
```

6.65.2.2 MatcherUntypedBase() [2/2]

```
Catch::Matchers::Impl::MatcherUntypedBase::MatcherUntypedBase (
    MatcherUntypedBase const & ) [default]
```

6.65.2.3 ~MatcherUntypedBase()

```
virtual Catch::Matchers::Impl::MatcherUntypedBase::~~MatcherUntypedBase ( ) [protected], [virtual]
```

6.65.3 Member Function Documentation

6.65.3.1 describe()

```
virtual std::string Catch::Matchers::Impl::MatcherUntypedBase::describe ( ) const [protected],
[pure virtual]
```

Implemented in [Catch::Matchers::Impl::MatchAllOf< ArgT >](#), [Catch::Matchers::Impl::MatchAnyOf< ArgT >](#), [Catch::Matchers::Impl::MatchNotOf< ArgT >](#), [Catch::Matchers::Exception::ExceptionMessageMatcher](#), [Catch::Matchers::Floating::WithinUlpMatcher](#), [Catch::Matchers::Floating::WithinRelMatcher](#), [Catch::Matchers::Generic::PredicateMatcher](#), [Catch::Matchers::StdString::StringMatcherBase](#), [Catch::Matchers::StdString::RegexMatcher](#), [Catch::Matchers::Vector::ContainsElementMatcher](#), [Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >](#), [Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >](#), [Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >](#), and [Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >](#).

6.65.3.2 operator=()

```
MatcherUntypedBase & Catch::Matchers::Impl::MatcherUntypedBase::operator= (
    MatcherUntypedBase const & ) [delete]
```

6.65.3.3 toString()

```
std::string Catch::Matchers::Impl::MatcherUntypedBase::toString ( ) const
```

6.65.4 Member Data Documentation

6.65.4.1 m_cachedToString

`std::string Catch::Matchers::Impl::MatcherUntypedBase::m_cachedToString [mutable], [protected]`

Definition at line 3253 of file [catch.h](#).

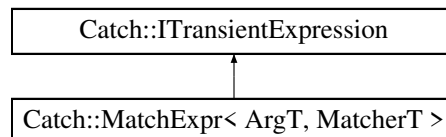
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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](#))
- [virtual ~ITransientExpression](#) ()

Additional Inherited Members

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

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

6.66.1 Detailed Description

```
template<typename ArgT, typename MatcherT>
class Catch::MatchExpr< ArgT, MatcherT >
```

Definition at line 3770 of file [catch.h](#).

6.66.2 Constructor & Destructor Documentation

6.66.2.1 MatchExpr()

```
template<typename ArgT , typename MatcherT >
Catch::MatchExpr< ArgT, MatcherT >::MatchExpr (
    ArgT const & arg,
    MatcherT const & matcher,
    StringRef const & matcherString ) [inline]
```

Definition at line 3775 of file [catch.h](#).

6.66.3 Member Function Documentation

6.66.3.1 streamReconstructedExpression()

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

Implements [Catch::ITransientExpression](#).

Definition at line 3782 of file [catch.h](#).

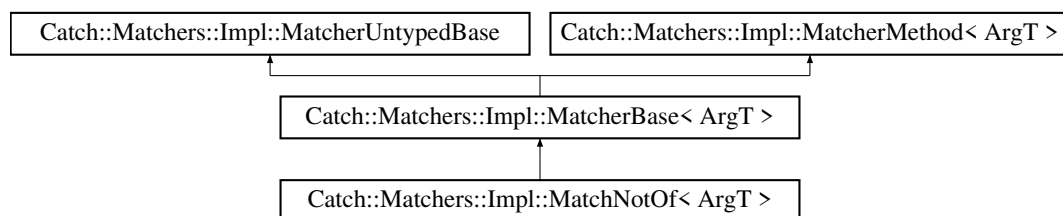
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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 \(\)=default](#)
- [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 Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- [virtual ~MatcherUntypedBase \(\)](#)

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

- [std::string m_cachedToString](#)

6.67.1 Detailed Description

```
template<typename ArgT>
struct Catch::Matchers::Impl::MatchNotOf< ArgT >
```

Definition at line 3356 of file [catch.h](#).

6.67.2 Constructor & Destructor Documentation

6.67.2.1 MatchNotOf()

```
template<typename ArgT >
Catch::Matchers::Impl::MatchNotOf< ArgT >::MatchNotOf (
    MatcherBase< ArgT > const & underlyingMatcher ) [inline]
```

Definition at line 3358 of file [catch.h](#).

6.67.3 Member Function Documentation

6.67.3.1 describe()

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

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

Definition at line 3364 of file [catch.h](#).

6.67.3.2 match()

```
template<typename ArgT >
bool Catch::Matchers::Impl::MatchNotOf< ArgT >::match (
    ArgT const & arg ) const [inline], [override]
```

Definition at line 3360 of file [catch.h](#).

6.67.4 Member Data Documentation

6.67.4.1 m_underlyingMatcher

```
template<typename ArgT >
MatcherBase<ArgT> const& Catch::Matchers::Impl::MatchNotOf< ArgT >::m_underlyingMatcher
```

Definition at line 3367 of file [catch.h](#).

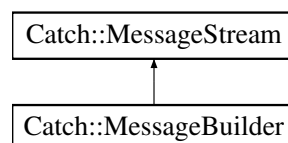
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.68 Catch::MessageBuilder Struct Reference

```
#include <catch.h>
```

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](#)

6.68.1 Detailed Description

Definition at line 2627 of file [catch.h](#).

6.68.2 Constructor & Destructor Documentation

6.68.2.1 MessageBuilder()

```
Catch::MessageBuilder::MessageBuilder (
    StringRef const & macroName,
    SourceLineInfo const & lineInfo,
    ResultWas::OfType type )
```

6.68.3 Member Function Documentation

6.68.3.1 operator<<()

```
template<typename T >
MessageBuilder & Catch::MessageBuilder::operator<< (
    T const & value ) [inline]
```

Definition at line 2633 of file [catch.h](#).

6.68.4 Member Data Documentation

6.68.4.1 m_info

[MessageInfo](#) Catch::MessageBuilder::m_info

Definition at line 2638 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.69 Catch::MessageInfo Struct Reference

```
#include <catch.h>
```

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

6.69.1 Detailed Description

Definition at line 2599 of file [catch.h](#).

6.69.2 Constructor & Destructor Documentation

6.69.2.1 MessageInfo()

```
Catch::MessageInfo::MessageInfo (
    StringRef const & _macroName,
    SourceLineInfo const & _lineInfo,
    ResultWas::OfType _type )
```

6.69.3 Member Function Documentation

6.69.3.1 operator<()

```
bool Catch::MessageInfo::operator< (
    MessageInfo const & other ) const
```

6.69.3.2 operator==()

```
bool Catch::MessageInfo::operator== (
    MessageInfo const & other ) const
```

6.69.4 Member Data Documentation

6.69.4.1 lineInfo

`SourceLineInfo` `Catch::MessageInfo::lineInfo`

Definition at line 2606 of file [catch.h](#).

6.69.4.2 macroName

`StringRef` `Catch::MessageInfo::macroName`

Definition at line 2604 of file [catch.h](#).

6.69.4.3 message

`std::string` `Catch::MessageInfo::message`

Definition at line 2605 of file [catch.h](#).

6.69.4.4 sequence

`unsigned int` `Catch::MessageInfo::sequence`

Definition at line 2608 of file [catch.h](#).

6.69.4.5 type

`ResultWas::OfType` `Catch::MessageInfo::type`

Definition at line 2607 of file [catch.h](#).

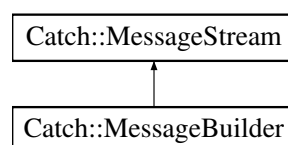
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.70 Catch::MessageStream Struct Reference

```
#include <catch.h>
```

Inheritance diagram for `Catch::MessageStream`:



Public Member Functions

- `template<typename T >`
`MessageStream & operator<< (T const &value)`

Public Attributes

- `ReusableStringStream m_stream`

6.70.1 Detailed Description

Definition at line 2616 of file [catch.h](#).

6.70.2 Member Function Documentation

6.70.2.1 `operator<<()`

```
template<typename T >
MessageStream & Catch::MessageStream::operator<< (
    T const & value ) [inline]
```

Definition at line 2619 of file [catch.h](#).

6.70.3 Member Data Documentation

6.70.3.1 `m_stream`

`ReusableStringStream` `Catch::MessageStream::m_stream`

Definition at line 2624 of file [catch.h](#).

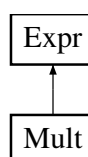
The documentation for this struct was generated from the following file:

- `/Users/samanthapope/msdscriptRepo/msdScript/catch.h`

6.71 Mult Class Reference

```
#include <Expr.h>
```

Inheritance diagram for Mult:



Public Member Functions

- [Mult](#) ([Expr](#) *lhs, [Expr](#) *rhs)
- bool [equals](#) ([Expr](#) *e) override
- int [interp](#) () override
- bool [hasVariable](#) () override
- [Expr](#) * [subst](#) (std::string stringInput, [Expr](#) *e) override
- void [print](#) (std::ostream &stream) override

Public Member Functions inherited from [Expr](#)

- virtual [~Expr](#) ()=default
- std::string [to_string](#) ()
- std::string [to_pp_string](#) ()
- void [pretty_print_at](#) (std::ostream &ot)

Public Attributes

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

Protected Member Functions

- void [pretty_print](#) (std::ostream &ot, [precedence_t](#) prec) override

6.71.1 Detailed Description

Definition at line 75 of file [Expr.h](#).

6.71.2 Constructor & Destructor Documentation

6.71.2.1 Mult()

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

Constructs a [Mult](#) object with left and right expressions.

Parameters

<i>lhs</i>	Pointer to the left-hand side expression.
<i>rhs</i>	Pointer to the right-hand side expression.

Definition at line 27 of file [Expr.cpp](#).

6.71.3 Member Function Documentation

6.71.3.1 equals()

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

Checks if this [Mult](#) object is equal to another expression.

Parameters

<i>e</i>	Pointer to the expression to compare with.
----------	--

Returns

true if the expressions are equivalent, false otherwise.

Implements [Expr](#).

Definition at line 65 of file [Expr.cpp](#).

6.71.3.2 hasVariable()

```
bool Mult::hasVariable ( ) [override], [virtual]
```

Checks if the [Mult](#) expression contains a variable.

Returns

true if either the left or right expressions contain a variable, false otherwise.

Implements [Expr](#).

Definition at line 133 of file [Expr.cpp](#).

6.71.3.3 interp()

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

Evaluates the multiplication expression.

Returns

The product of the left and right expressions.

Implements [Expr](#).

Definition at line 101 of file [Expr.cpp](#).

6.71.3.4 pretty_print()

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

Pretty prints the [Mult](#) expression with appropriate precedence.

Parameters

<i>ot</i>	The output stream to print to.
<i>prec</i>	The precedence context in which this expression is being printed.

Implements [Expr](#).

Definition at line 261 of file [Expr.cpp](#).

6.71.3.5 print()

```
void Mult::print (
    std::ostream & stream ) [override], [virtual]
```

Prints the [Mult](#) expression to a given output stream.

Parameters

<i>stream</i>	The output stream to print to.
---------------	--------------------------------

Implements [Expr](#).

Definition at line 217 of file [Expr.cpp](#).

6.71.3.6 subst()

```
Expr * Mult::subst (
    std::string stringInput,
    Expr * e ) [override], [virtual]
```

Substitutes a variable with another expression in a [Mult](#) object.

Parameters

<i>stringInput</i>	The name of the variable to substitute.
<i>e</i>	The expression to substitute the variable with.

Returns

A new [Mult](#) object with the substituted expressions.

Implements [Expr](#).

Definition at line 173 of file [Expr.cpp](#).

6.71.4 Member Data Documentation

6.71.4.1 lhs

```
Expr* Mult::lhs
```

Definition at line 77 of file [Expr.h](#).

6.71.4.2 rhs

`Expr* Mult::rhs`

Definition at line 78 of file [Expr.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/Expr.h](#)
- [/Users/samanthapope/msdscriptRepo/msdScript/Expr.cpp](#)

6.72 Catch::NameAndTags Struct Reference

```
#include <catch.h>
```

Public Member Functions

- [NameAndTags](#) ([StringRef](#) const &name__{__}=[StringRef](#)(), [StringRef](#) const &tags__{__}=[StringRef](#)()) [noexcept](#)

Public Attributes

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

6.72.1 Detailed Description

Definition at line 980 of file [catch.h](#).

6.72.2 Constructor & Destructor Documentation

6.72.2.1 NameAndTags()

```
Catch::NameAndTags::NameAndTags (
    StringRef const & name___ = StringRef(),
    StringRef const & tags___ = StringRef() ) [noexcept]
```

6.72.3 Member Data Documentation

6.72.3.1 name

[StringRef](#) [Catch::NameAndTags::name](#)

Definition at line 982 of file [catch.h](#).

6.72.3.2 tags

`StringRef` `Catch::NameAndTags::tags`

Definition at line 983 of file [catch.h](#).

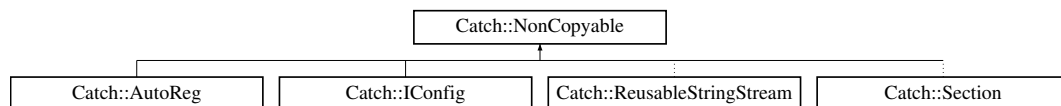
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.73 Catch::NonCopyable Class Reference

```
#include <catch.h>
```

Inheritance diagram for `Catch::NonCopyable`:



Protected Member Functions

- [NonCopyable](#) ()
- [virtual ~NonCopyable](#) ()

6.73.1 Detailed Description

Definition at line 491 of file [catch.h](#).

6.73.2 Constructor & Destructor Documentation

6.73.2.1 NonCopyable()

```
Catch::NonCopyable::NonCopyable ( ) [protected]
```

6.73.2.2 ~NonCopyable()

```
virtual Catch::NonCopyable::~~NonCopyable ( ) [protected], [virtual]
```

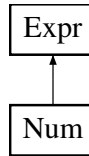
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.74 Num Class Reference

```
#include <Expr.h>
```

Inheritance diagram for Num:



Public Member Functions

- [Num](#) (int [val](#))
- bool [equals](#) ([Expr](#) *e) override
- int [interp](#) () override
- bool [hasVariable](#) () override
- [Expr](#) * [subst](#) (std::string stringInput, [Expr](#) *e) override
- void [print](#) (std::ostream &stream) override

Public Member Functions inherited from [Expr](#)

- virtual [~Expr](#) ()=default
- std::string [to_string](#) ()
- std::string [to_pp_string](#) ()
- void [pretty_print_at](#) (std::ostream &ot)

Public Attributes

- int [val](#)

Protected Member Functions

- void [pretty_print](#) (std::ostream &ot, [precedence_t](#) prec) override

6.74.1 Detailed Description

Definition at line 46 of file [Expr.h](#).

6.74.2 Constructor & Destructor Documentation

6.74.2.1 Num()

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

Constructs a [Num](#) object with a given integer value.

Parameters

<i>val</i>	The integer value to initialize the Num object with.
------------	--

Definition at line 8 of file [Expr.cpp](#).

6.74.3 Member Function Documentation

6.74.3.1 equals()

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

Checks if this [Num](#) object is equal to another expression.

Parameters

<i>e</i>	Pointer to the expression to compare with.
----------	--

Returns

true if the expressions are equivalent, false otherwise.

Implements [Expr](#).

Definition at line 44 of file [Expr.cpp](#).

6.74.3.2 hasVariable()

```
bool Num::hasVariable ( ) [override], [virtual]
```

Checks if the [Num](#) expression contains a variable.

Returns

false because [Num](#) does not contain a variable.

Implements [Expr](#).

Definition at line 117 of file [Expr.cpp](#).

6.74.3.3 interp()

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

Evaluates the numeric expression.

Returns

The integer value of the [Num](#) object.

Implements [Expr](#).

Definition at line 85 of file [Expr.cpp](#).

6.74.3.4 pretty_print()

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

Pretty prints the [Num](#) expression with appropriate precedence.

Parameters

<i>ot</i>	The output stream to print to.
<i>prec</i>	The precedence context in which this expression is being printed.

Implements [Expr](#).

Definition at line 238 of file [Expr.cpp](#).

6.74.3.5 print()

```
void Num::print (
    std::ostream & stream ) [override], [virtual]
```

Prints the [Num](#) expression to a given output stream.

Parameters

<i>stream</i>	The output stream to print to.
---------------	--------------------------------

Implements [Expr](#).

Definition at line 197 of file [Expr.cpp](#).

6.74.3.6 subst()

```
Expr * Num::subst (
    std::string stringInput,
    Expr * e ) [override], [virtual]
```

Substitutes a variable with another expression in a [Num](#) object.

Parameters

<i>stringInput</i>	The name of the variable to substitute.
<i>e</i>	The expression to substitute the variable with.

Returns

A new [Num](#) object with the same value, since no substitution is needed.

Implements [Expr](#).

Definition at line 151 of file [Expr.cpp](#).

6.74.4 Member Data Documentation

6.74.4.1 val

```
int Num::val
```

Definition at line 48 of file [Expr.h](#).

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

- [/Users/samanthapope/msdscripRepo/msdScript/Expr.h](#)
- [/Users/samanthapope/msdscripRepo/msdScript/Expr.cpp](#)

6.75 Catch::Option< T > Class Template Reference

```
#include <catch.h>
```

Public Member Functions

- [Option \(\)](#)
- [Option \(T const &_value\)](#)
- [Option \(Option const &_other\)](#)
- [~Option \(\)](#)
- [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](#)

6.75.1 Detailed Description

```
template<typename T>
class Catch::Option< T >
```

Definition at line 4409 of file [catch.h](#).

6.75.2 Constructor & Destructor Documentation

6.75.2.1 Option() [1/3]

```
template<typename T >  
Catch::Option< T >::Option ( ) [inline]
```

Definition at line 4411 of file [catch.h](#).

6.75.2.2 Option() [2/3]

```
template<typename T >  
Catch::Option< T >::Option (   
    T const & _value ) [inline]
```

Definition at line 4412 of file [catch.h](#).

6.75.2.3 Option() [3/3]

```
template<typename T >  
Catch::Option< T >::Option (   
    Option< T > const & _other ) [inline]
```

Definition at line 4415 of file [catch.h](#).

6.75.2.4 ~Option()

```
template<typename T >  
Catch::Option< T >::~~Option ( ) [inline]
```

Definition at line 4419 of file [catch.h](#).

6.75.3 Member Function Documentation

6.75.3.1 none()

```
template<typename T >  
bool Catch::Option< T >::none ( ) const [inline]
```

Definition at line 4453 of file [catch.h](#).

6.75.3.2 operator bool()

```
template<typename T >  
Catch::Option< T >::operator bool ( ) const [inline], [explicit]
```

Definition at line 4456 of file [catch.h](#).

6.75.3.3 operator"!()

```
template<typename T >
bool Catch::Option< T >::operator! ( ) const [inline]
```

Definition at line 4455 of file [catch.h](#).

6.75.3.4 operator*() [1/2]

```
template<typename T >
T & Catch::Option< T >::operator* ( ) [inline]
```

Definition at line 4443 of file [catch.h](#).

6.75.3.5 operator*() [2/2]

```
template<typename T >
T const & Catch::Option< T >::operator* ( ) const [inline]
```

Definition at line 4444 of file [catch.h](#).

6.75.3.6 operator->() [1/2]

```
template<typename T >
T * Catch::Option< T >::operator-> ( ) [inline]
```

Definition at line 4445 of file [catch.h](#).

6.75.3.7 operator->() [2/2]

```
template<typename T >
const T * Catch::Option< T >::operator-> ( ) const [inline]
```

Definition at line 4446 of file [catch.h](#).

6.75.3.8 operator=() [1/2]

```
template<typename T >
Option & Catch::Option< T >::operator= (
    Option< T > const & _other ) [inline]
```

Definition at line 4423 of file [catch.h](#).

6.75.3.9 operator=() [2/2]

```
template<typename T >
Option & Catch::Option< T >::operator= (
    T const & _value ) [inline]
```

Definition at line 4431 of file [catch.h](#).

6.75.3.10 reset()

```
template<typename T >
void Catch::Option< T >::reset ( ) [inline]
```

Definition at line 4437 of file [catch.h](#).

6.75.3.11 some()

```
template<typename T >
bool Catch::Option< T >::some ( ) const [inline]
```

Definition at line 4452 of file [catch.h](#).

6.75.3.12 valueOr()

```
template<typename T >
T Catch::Option< T >::valueOr (
    T const & defaultValue ) const [inline]
```

Definition at line 4448 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.76 Catch::pluralise Struct Reference

```
#include <catch.h>
```

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](#))

6.76.1 Detailed Description

Definition at line 3216 of file [catch.h](#).

6.76.2 Constructor & Destructor Documentation

6.76.2.1 pluralise()

```
Catch::pluralise::pluralise (
    std::size_t count,
    std::string const & label )
```

6.76.3 Friends And Related Symbol Documentation

6.76.3.1 operator<<

```
std::ostream & operator<< (
    std::ostream & os,
    pluralise const & pluraliser ) [friend]
```

6.76.4 Member Data Documentation

6.76.4.1 m_count

```
std::size_t Catch::pluralise::m_count
```

Definition at line 3221 of file [catch.h](#).

6.76.4.2 m_label

```
std::string Catch::pluralise::m_label
```

Definition at line 3222 of file [catch.h](#).

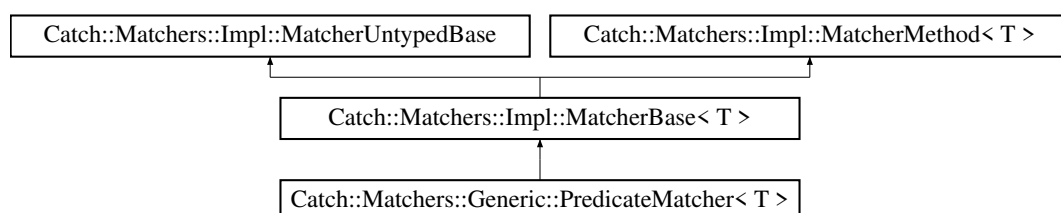
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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

Additional Inherited Members

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

- [virtual ~MatcherUntypedBase](#) ()

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

- [std::string m_cachedToString](#)

6.77.1 Detailed Description

```
template<typename T>
class Catch::Matchers::Generic::PredicateMatcher< T >
```

Definition at line 3495 of file [catch.h](#).

6.77.2 Constructor & Destructor Documentation

6.77.2.1 PredicateMatcher()

```
template<typename T >
Catch::Matchers::Generic::PredicateMatcher< T >::PredicateMatcher (
    std::function< bool(T const &)> const & elem,
    std::string const & descr ) [inline]
```

Definition at line 3500 of file [catch.h](#).

6.77.3 Member Function Documentation

6.77.3.1 describe()

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

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

Definition at line 3509 of file [catch.h](#).

6.77.3.2 match()

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

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

Definition at line 3505 of file [catch.h](#).

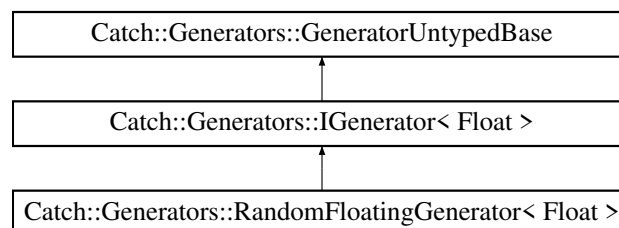
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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



Public Member Functions

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

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

- [virtual ~IGenerator \(\)=default](#)

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

- [GeneratorUntypedBase \(\)=default](#)
- [virtual ~GeneratorUntypedBase \(\)](#)

Additional Inherited Members

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

- [using type](#)

6.78.1 Detailed Description

```
template<typename Float>
class Catch::Generators::RandomFloatingGenerator< Float >
```

Definition at line 4603 of file [catch.h](#).

6.78.2 Constructor & Destructor Documentation

6.78.2.1 RandomFloatingGenerator()

```
template<typename Float >
Catch::Generators::RandomFloatingGenerator< Float >::RandomFloatingGenerator (
    Float a,
    Float b ) [inline]
```

Definition at line 4609 of file [catch.h](#).

6.78.3 Member Function Documentation

6.78.3.1 get()

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

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

Definition at line 4615 of file [catch.h](#).

6.78.3.2 next()

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

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

Definition at line 4618 of file [catch.h](#).

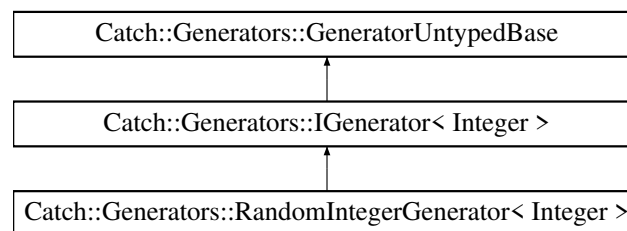
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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



Public Member Functions

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

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

- [virtual ~IGenerator \(\)=default](#)

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

- [GeneratorUntypedBase \(\)=default](#)
- [virtual ~GeneratorUntypedBase \(\)](#)

Additional Inherited Members

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

- [using type](#)

6.79.1 Detailed Description

```
template<typename Integer>
class Catch::Generators::RandomIntegerGenerator< Integer >
```

Definition at line 4625 of file [catch.h](#).

6.79.2 Constructor & Destructor Documentation

6.79.2.1 RandomIntegerGenerator()

```
template<typename Integer >
Catch::Generators::RandomIntegerGenerator< Integer >::RandomIntegerGenerator (
    Integer a,
    Integer b ) [inline]
```

Definition at line 4631 of file [catch.h](#).

6.79.3 Member Function Documentation

6.79.3.1 get()

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

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

Definition at line 4637 of file [catch.h](#).

6.79.3.2 next()

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

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

Definition at line 4640 of file [catch.h](#).

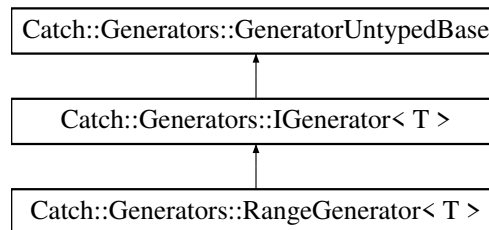
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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](#)

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

- [virtual](#) [~IGenerator](#) ()=[default](#)

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

- [GeneratorUntypedBase](#) ()=[default](#)
- [virtual](#) [~GeneratorUntypedBase](#) ()

Additional Inherited Members

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

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

6.80.1 Detailed Description

```
template<typename T>
class Catch::Generators::RangeGenerator< T >
```

Definition at line [4667](#) of file [catch.h](#).

6.80.2 Constructor & Destructor Documentation

6.80.2.1 RangeGenerator() [1/2]

```
template<typename T >
Catch::Generators::RangeGenerator< T >::RangeGenerator (
    T const & start,
    T const & end,
    T const & step ) [inline]
```

Definition at line 4674 of file [catch.h](#).

6.80.2.2 RangeGenerator() [2/2]

```
template<typename T >
Catch::Generators::RangeGenerator< T >::RangeGenerator (
    T const & start,
    T const & end ) [inline]
```

Definition at line 4685 of file [catch.h](#).

6.80.3 Member Function Documentation

6.80.3.1 get()

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

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

Definition at line 4689 of file [catch.h](#).

6.80.3.2 next()

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

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

Definition at line 4693 of file [catch.h](#).

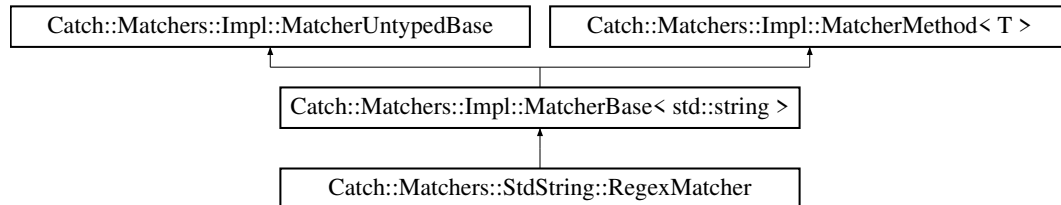
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.81 Catch::Matchers::StdString::RegexMatcher Struct Reference

```
#include <catch.h>
```

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](#) ()=default
- [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 ~MatcherUntypedBase](#) ()

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

- [std::string m_cachedToString](#)

6.81.1 Detailed Description

Definition at line 3573 of file [catch.h](#).

6.81.2 Constructor & Destructor Documentation

6.81.2.1 RegexMatcher()

```
Catch::Matchers::StdString::RegexMatcher::RegexMatcher (
    std::string regex,
    CaseSensitive::Choice caseSensitivity )
```

6.81.3 Member Function Documentation

6.81.3.1 describe()

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

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

6.81.3.2 match()

```
bool Catch::Matchers::StdString::RegexMatcher::match (
    std::string const & matchee ) const [override]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.82 Catch::RegistrarForTagAliases Struct Reference

```
#include <catch.h>
```

Public Member Functions

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

6.82.1 Detailed Description

Definition at line 549 of file [catch.h](#).

6.82.2 Constructor & Destructor Documentation

6.82.2.1 RegistrarForTagAliases()

```
Catch::RegistrarForTagAliases::RegistrarForTagAliases (
    char const * alias,
    char const * tag,
    SourceLineInfo const & lineInfo )
```

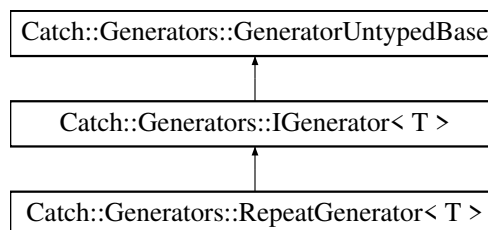
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.83 Catch::Generators::RepeatGenerator< T > Class Template Reference

```
#include <catch.h>
```

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

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

- [virtual ~IGenerator \(\)](#)=default

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

- [GeneratorUntypedBase \(\)](#)=default
- [virtual ~GeneratorUntypedBase \(\)](#)

Additional Inherited Members

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

- [using type](#) = T

6.83.1 Detailed Description

```
template<typename T>
class Catch::Generators::RepeatGenerator< T >
```

Definition at line 4200 of file [catch.h](#).

6.83.2 Constructor & Destructor Documentation

6.83.2.1 RepeatGenerator()

```
template<typename T >
Catch::Generators::RepeatGenerator< T >::RepeatGenerator (
    size_t repeats,
    GeneratorWrapper< T > && generator ) [inline]
```

Definition at line 4210 of file [catch.h](#).

6.83.3 Member Function Documentation

6.83.3.1 get()

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

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

Definition at line 4217 of file [catch.h](#).

6.83.3.2 next()

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

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

Definition at line 4225 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.84 Catch::ResultDisposition Struct Reference

```
#include <catch.h>
```

Public Types

- enum [Flags](#) { [Normal](#) = 0x01 , [ContinueOnFailure](#) = 0x02 , [FalseTest](#) = 0x04 , [SuppressFail](#) = 0x08 }

6.84.1 Detailed Description

Definition at line [1377](#) of file [catch.h](#).

6.84.2 Member Enumeration Documentation

6.84.2.1 Flags

```
enum Catch::ResultDisposition::Flags
```

Enumerator

Normal	
ContinueOnFailure	
FalseTest	
SuppressFail	

Definition at line [1377](#) of file [catch.h](#).

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

- [./Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.85 Catch::ResultWas Struct Reference

```
#include <catch.h>
```

Public Types

- enum [OfType](#) {
[Unknown](#) = -1 , [Ok](#) = 0 , [Info](#) = 1 , [Warning](#) = 2 ,
[FailureBit](#) = 0x10 , [ExpressionFailed](#) = [FailureBit](#) | 1 , [ExplicitFailure](#) = [FailureBit](#) | 2 , [Exception](#) = 0x100 |
[FailureBit](#) ,
[ThrewException](#) = [Exception](#) | 1 , [DidntThrowException](#) = [Exception](#) | 2 , [FatalErrorCondition](#) = 0x200 |
[FailureBit](#) }

6.85.1 Detailed Description

Definition at line [1353](#) of file [catch.h](#).

6.85.2 Member Enumeration Documentation

6.85.2.1 OfType

```
enum Catch::ResultWas::OfType
```

Enumerator

Unknown	
Ok	
Info	
Warning	
FailureBit	
ExpressionFailed	
ExplicitFailure	
Exception	
ThrowException	
DidntThrowException	
FatalErrorCondition	

Definition at line 1353 of file [catch.h](#).

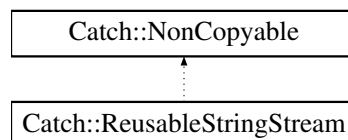
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.86 Catch::ReusableStringStream Class Reference

```
#include <catch.h>
```

Inheritance diagram for Catch::ReusableStringStream:



Public Member Functions

- [ReusableStringStream \(\)](#)
- [~ReusableStringStream \(\)](#)
- [auto str \(\) const](#) -> `std::string`
- `template<typename T>`
[auto operator<< \(T const &value\)](#) -> [ReusableStringStream &](#)
- [auto get \(\)](#) -> `std::ostream &`

6.86.1 Detailed Description

Definition at line 1440 of file [catch.h](#).

6.86.2 Constructor & Destructor Documentation

6.86.2.1 ReusableStringStream()

```
Catch::ReusableStringStream::ReusableStringStream ( )
```

6.86.2.2 ~ReusableStringStream()

```
Catch::ReusableStringStream::~~ReusableStringStream ( )
```

6.86.3 Member Function Documentation

6.86.3.1 get()

```
auto Catch::ReusableStringStream::get ( ) -> std::ostream& [inline]
```

Definition at line 1454 of file [catch.h](#).

6.86.3.2 operator<<()

```
template<typename T >  
auto Catch::ReusableStringStream::operator<< (   
    T const & value ) -> ReusableStringStream& [inline]
```

Definition at line 1450 of file [catch.h](#).

6.86.3.3 str()

```
auto Catch::ReusableStringStream::str ( ) const -> std::string
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.87 Catch::RunTests Struct Reference

```
#include <catch.h>
```

Public Types

- enum [InWhatOrder](#) { [InDeclarationOrder](#) , [InLexicographicalOrder](#) , [InRandomOrder](#) }

6.87.1 Detailed Description

Definition at line 4493 of file [catch.h](#).

6.87.2 Member Enumeration Documentation

6.87.2.1 InWhatOrder

```
enum Catch::RunTests::InWhatOrder
```

Enumerator

InDeclarationOrder	
InLexicographicalOrder	
InRandomOrder	

Definition at line 4493 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.88 Catch::ScopedMessage Class Reference

```
#include <catch.h>
```

Public Member Functions

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

Public Attributes

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

6.88.1 Detailed Description

Definition at line 2641 of file [catch.h](#).

6.88.2 Constructor & Destructor Documentation

6.88.2.1 ScopedMessage() [1/3]

```
Catch::ScopedMessage::ScopedMessage (
    MessageBuilder const & builder ) [explicit]
```

6.88.2.2 ScopedMessage() [2/3]

```
Catch::ScopedMessage::ScopedMessage (
    ScopedMessage & duplicate ) [delete]
```

6.88.2.3 ScopedMessage() [3/3]

```
Catch::ScopedMessage::ScopedMessage (
    ScopedMessage && old )
```

6.88.2.4 ~ScopedMessage()

```
Catch::ScopedMessage::~~ScopedMessage ( )
```

6.88.3 Member Data Documentation

6.88.3.1 m_info

```
MessageInfo Catch::ScopedMessage::m_info
```

Definition at line 2648 of file [catch.h](#).

6.88.3.2 m_moved

```
bool Catch::ScopedMessage::m_moved
```

Definition at line 2649 of file [catch.h](#).

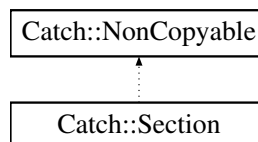
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.89 Catch::Section Class Reference

```
#include <catch.h>
```

Inheritance diagram for Catch::Section:



Public Member Functions

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

6.89.1 Detailed Description

Definition at line 2911 of file [catch.h](#).

6.89.2 Constructor & Destructor Documentation

6.89.2.1 Section()

```
Catch::Section::Section (
    SectionInfo const & info )
```

6.89.2.2 ~Section()

```
Catch::Section::~~Section ( )
```

6.89.3 Member Function Documentation

6.89.3.1 operator bool()

```
Catch::Section::operator bool ( ) const [explicit]
```

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.90 Catch::SectionEndInfo Struct Reference

```
#include <catch.h>
```

Public Attributes

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

6.90.1 Detailed Description

Definition at line 2876 of file [catch.h](#).

6.90.2 Member Data Documentation

6.90.2.1 durationInSeconds

`double` `Catch::SectionEndInfo::durationInSeconds`

Definition at line 2879 of file [catch.h](#).

6.90.2.2 prevAssertions

`Counts` `Catch::SectionEndInfo::prevAssertions`

Definition at line 2878 of file [catch.h](#).

6.90.2.3 sectionInfo

`SectionInfo` `Catch::SectionEndInfo::sectionInfo`

Definition at line 2877 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.91 Catch::SectionInfo Struct Reference

```
#include <catch.h>
```

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](#)

6.91.1 Detailed Description

Definition at line 2860 of file [catch.h](#).

6.91.2 Constructor & Destructor Documentation

6.91.2.1 SectionInfo() [1/2]

```
Catch::SectionInfo::SectionInfo (
    SourceLineInfo const & _lineInfo,
    std::string const & _name )
```

6.91.2.2 SectionInfo() [2/2]

```
Catch::SectionInfo::SectionInfo (
    SourceLineInfo const & _lineInfo,
    std::string const & _name,
    std::string const & ) [inline]
```

Definition at line 2866 of file [catch.h](#).

6.91.3 Member Data Documentation

6.91.3.1 description

```
std::string Catch::SectionInfo::description
```

Definition at line 2872 of file [catch.h](#).

6.91.3.2 lineInfo

```
SourceLineInfo Catch::SectionInfo::lineInfo
```

Definition at line 2873 of file [catch.h](#).

6.91.3.3 name

```
std::string Catch::SectionInfo::name
```

Definition at line 2871 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.92 Catch::ShowDurations Struct Reference

```
#include <catch.h>
```

Public Types

- enum [OrNot](#) { [DefaultForReporter](#) , [Always](#) , [Never](#) }

6.92.1 Detailed Description

Definition at line [4488](#) of file [catch.h](#).

6.92.2 Member Enumeration Documentation

6.92.2.1 OrNot

```
enum Catch::ShowDurations::OrNot
```

Enumerator

DefaultForReporter	
Always	
Never	

Definition at line [4488](#) of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.93 Catch::SimplePcg32 Class Reference

```
#include <catch.h>
```

Public Types

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

Public Member Functions

- [SimplePcg32](#) ()
- [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`)

6.93.1 Detailed Description

Definition at line 4558 of file [catch.h](#).

6.93.2 Member Typedef Documentation

6.93.2.1 `result_type`

```
using Catch::SimplePcg32::result_type = std::uint32_t
```

Definition at line 4561 of file [catch.h](#).

6.93.3 Constructor & Destructor Documentation

6.93.3.1 `SimplePcg32()` [1/2]

```
Catch::SimplePcg32::SimplePcg32 ( ) [inline]
```

Definition at line 4570 of file [catch.h](#).

6.93.3.2 `SimplePcg32()` [2/2]

```
Catch::SimplePcg32::SimplePcg32 (
    result_type seed_ ) [explicit]
```

6.93.4 Member Function Documentation

6.93.4.1 `discard()`

```
void Catch::SimplePcg32::discard (
    uint64_t skip )
```

6.93.4.2 `max()`

```
static constexpr result_type Catch::SimplePcg32::max ( ) [inline], [static], [constexpr]
```

Definition at line 4565 of file [catch.h](#).

6.93.4.3 min()

```
static constexpr result_type Catch::SimplePcg32::min ( ) [inline], [static], [constexpr]
```

Definition at line 4562 of file [catch.h](#).

6.93.4.4 operator()()

```
result_type Catch::SimplePcg32::operator() ( )
```

6.93.4.5 seed()

```
void Catch::SimplePcg32::seed (
    result_type seed_ )
```

6.93.5 Friends And Related Symbol Documentation

6.93.5.1 operator"!="

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

6.93.5.2 operator==

```
bool operator== (
    SimplePcg32 const & lhs,
    SimplePcg32 const & rhs ) [friend]
```

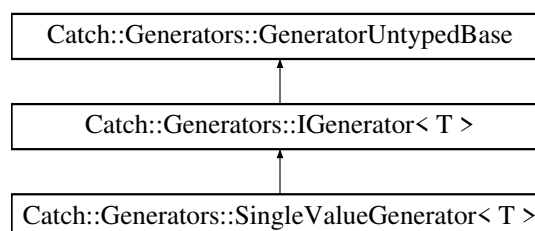
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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



Public Member Functions

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

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

- [virtual ~IGenerator \(\)=default](#)

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

- [GeneratorUntypedBase \(\)=default](#)
- [virtual ~GeneratorUntypedBase \(\)](#)

Additional Inherited Members

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

- [using type = T](#)

6.94.1 Detailed Description

```
template<typename T>
class Catch::Generators::SingleValueGenerator< T >
```

Definition at line 3953 of file [catch.h](#).

6.94.2 Constructor & Destructor Documentation

6.94.2.1 SingleValueGenerator()

```
template<typename T >
Catch::Generators::SingleValueGenerator< T >::SingleValueGenerator (
    T && value ) [inline]
```

Definition at line 3956 of file [catch.h](#).

6.94.3 Member Function Documentation

6.94.3.1 get()

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

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

Definition at line 3958 of file [catch.h](#).

6.94.3.2 next()

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

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

Definition at line [3961](#) of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.95 Catch::SourceLineInfo Struct Reference

```
#include <catch.h>
```

Public Member Functions

- [SourceLineInfo \(\)=delete](#)
- [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](#)

6.95.1 Detailed Description

Definition at line [502](#) of file [catch.h](#).

6.95.2 Constructor & Destructor Documentation

6.95.2.1 SourceLineInfo() [1/4]

```
Catch::SourceLineInfo::SourceLineInfo ( ) [delete]
```


6.95.2.2 SourceLineInfo() [2/4]

```
Catch::SourceLineInfo::SourceLineInfo (
    char const * _file,
    std::size_t _line ) [inline], [noexcept]
```

Definition at line 505 of file [catch.h](#).

6.95.2.3 SourceLineInfo() [3/4]

```
Catch::SourceLineInfo::SourceLineInfo (
    SourceLineInfo const & other ) [default]
```

6.95.2.4 SourceLineInfo() [4/4]

```
Catch::SourceLineInfo::SourceLineInfo (
    SourceLineInfo && ) [default], [noexcept]
```

6.95.3 Member Function Documentation

6.95.3.1 empty()

```
bool Catch::SourceLineInfo::empty ( ) const [inline], [noexcept]
```

Definition at line 515 of file [catch.h](#).

6.95.3.2 operator<()

```
bool Catch::SourceLineInfo::operator< (
    SourceLineInfo const & other ) const [noexcept]
```

6.95.3.3 operator=() [1/2]

```
SourceLineInfo & Catch::SourceLineInfo::operator= (
    SourceLineInfo && ) [default], [noexcept]
```

6.95.3.4 operator=() [2/2]

```
SourceLineInfo & Catch::SourceLineInfo::operator= (
    SourceLineInfo const & ) [default]
```

6.95.3.5 operator==()

```
bool Catch::SourceLineInfo::operator== (
    SourceLineInfo const & other ) const [noexcept]
```

6.95.4 Member Data Documentation

6.95.4.1 file

`char const* Catch::SourceLineInfo::file`

Definition at line 519 of file [catch.h](#).

6.95.4.2 line

`std::size_t Catch::SourceLineInfo::line`

Definition at line 520 of file [catch.h](#).

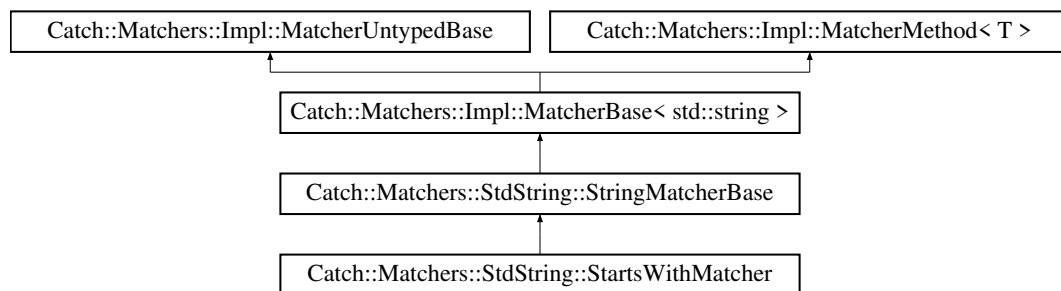
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.96 Catch::Matchers::StdString::StartsWithMatcher Struct Reference

```
#include <catch.h>
```

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](#) ()=default
- [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 Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- virtual ~[MatcherUntypedBase](#) ()

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

- std::string [m_cachedToString](#)

6.96.1 Detailed Description

Definition at line 3564 of file [catch.h](#).

6.96.2 Constructor & Destructor Documentation**6.96.2.1 StartsWithMatcher()**

```
Catch::Matchers::StdString::StartsWithMatcher::StartsWithMatcher (
    CasedString const & comparator )
```

6.96.3 Member Function Documentation**6.96.3.1 match()**

```
bool Catch::Matchers::StdString::StartsWithMatcher::match (
    std::string const & source ) const [override]
```

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

- /Users/samanthapope/msdscriptRepo/msdScript/[catch.h](#)

6.97 Catch::StreamEndStop Struct Reference

```
#include <catch.h>
```

Public Member Functions

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

6.97.1 Detailed Description

Definition at line 534 of file [catch.h](#).

6.97.2 Member Function Documentation

6.97.2.1 operator+()

```
std::string Catch::StreamEndStop::operator+ ( ) const
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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)`

6.98.1 Detailed Description

```
template<typename T, typename = void>  
struct Catch::StringMaker< T, typename >
```

Definition at line 1615 of file [catch.h](#).

6.98.2 Member Function Documentation

6.98.2.1 convert() [1/2]

```
template<typename T , typename = void>
template<typename Fake = T>
static std::enable_if<::Catch::Detail::IsStreamInsertable< Fake >::value, std::string >::type
Catch::StringMaker< T, typename >::convert (
    const Fake & value ) [inline], [static]
```

Definition at line 1619 of file [catch.h](#).

6.98.2.2 convert() [2/2]

```
template<typename T , typename = void>
template<typename Fake = T>
static std::enable_if<!::Catch::Detail::IsStreamInsertable< Fake >::value, std::string >↵
::type Catch::StringMaker< T, typename >::convert (
    const Fake & value ) [inline], [static]
```

Definition at line 1630 of file [catch.h](#).

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.99 Catch::StringMaker< bool > Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

- [static](#) std::string [convert](#) (bool b)

6.99.1 Detailed Description

Definition at line 1761 of file [catch.h](#).

6.99.2 Member Function Documentation

6.99.2.1 convert()

```
static std::string Catch::StringMaker< bool >::convert (
    bool b ) [static]
```

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.100 `Catch::StringMaker< Catch::Detail::Approx >` Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

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

6.100.1 Detailed Description

Definition at line 3185 of file [catch.h](#).

6.100.2 Member Function Documentation

6.100.2.1 `convert()`

```
static std::string Catch::StringMaker< Catch::Detail::Approx >::convert (
    Catch::Detail::Approx const & value ) [static]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.101 `Catch::StringMaker< char * >` Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

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

6.101.1 Detailed Description

Definition at line 1681 of file [catch.h](#).

6.101.2 Member Function Documentation

6.101.2.1 `convert()`

```
static std::string Catch::StringMaker< char * >::convert (
    char * str ) [static]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.102 Catch::StringMaker< char > Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

- [static](#) std::string [convert](#) ([char](#) c)

6.102.1 Detailed Description

Definition at line [1766](#) of file [catch.h](#).

6.102.2 Member Function Documentation

6.102.2.1 convert()

```
static std::string Catch::StringMaker< char >::convert (  
    char c ) [static]
```

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

- /Users/samanthapope/msdscripRepo/msdScript/[catch.h](#)

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

```
#include <catch.h>
```

Static Public Member Functions

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

6.103.1 Detailed Description

Definition at line [1677](#) of file [catch.h](#).

6.103.2 Member Function Documentation

6.103.2.1 convert()

```
static std::string Catch::StringMaker< char const * >::convert (  
    char const * str ) [static]
```

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

- /Users/samanthapope/msdscripRepo/msdScript/[catch.h](#)

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

```
#include <catch.h>
```

Static Public Member Functions

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

6.104.1 Detailed Description

```
template<int SZ>
struct Catch::StringMaker< char[SZ]>
```

Definition at line 1711 of file [catch.h](#).

6.104.2 Member Function Documentation

6.104.2.1 convert()

```
template<int SZ>
static std::string Catch::StringMaker< char[SZ]>::convert (
    char const * str ) [inline], [static]
```

Definition at line 1712 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.105 Catch::StringMaker< double > Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

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

Static Public Attributes

- `static int precision`

6.105.1 Detailed Description

Definition at line 1790 of file [catch.h](#).

6.105.2 Member Function Documentation

6.105.2.1 convert()

```
static std::string Catch::StringMaker< double >::convert (
    double value ) [static]
```

6.105.3 Member Data Documentation

6.105.3.1 precision

```
int Catch::StringMaker< double >::precision [static]
```

Definition at line 1792 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.106 Catch::StringMaker< float > Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

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

Static Public Attributes

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

6.106.1 Detailed Description

Definition at line 1784 of file [catch.h](#).

6.106.2 Member Function Documentation

6.106.2.1 convert()

```
static std::string Catch::StringMaker< float >::convert (
    float value ) [static]
```

6.106.3 Member Data Documentation

6.106.3.1 precision

```
int Catch::StringMaker< float >::precision [static]
```

Definition at line 1786 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.107 Catch::StringMaker< int > Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

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

6.107.1 Detailed Description

Definition at line 1736 of file [catch.h](#).

6.107.2 Member Function Documentation

6.107.2.1 convert()

```
static std::string Catch::StringMaker< int >::convert (  
    int value ) [static]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.108 Catch::StringMaker< long > Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

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

6.108.1 Detailed Description

Definition at line 1740 of file [catch.h](#).

6.108.2 Member Function Documentation

6.108.2.1 convert()

```
static std::string Catch::StringMaker< long >::convert (
    long value ) [static]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.109 Catch::StringMaker< long long > Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

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

6.109.1 Detailed Description

Definition at line 1744 of file [catch.h](#).

6.109.2 Member Function Documentation

6.109.2.1 convert()

```
static std::string Catch::StringMaker< long long >::convert (
    long long value ) [static]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

Static Public Member Functions

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

6.110.1 Detailed Description

```
template<typename R, typename C>
struct Catch::StringMaker< R C::* >
```

Definition at line 1808 of file [catch.h](#).

6.110.2 Member Function Documentation

6.110.2.1 convert()

```
template<typename R , typename C >
static std::string Catch::StringMaker< R C::* >::convert (
    R C::* p ) [inline], [static]
```

Definition at line 1809 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

Static Public Member Functions

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

6.111.1 Detailed Description

```
template<typename R>
struct Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStreamInsertable< R >::value >::type >
```

Definition at line 2040 of file [catch.h](#).

6.111.2 Member Function Documentation

6.111.2.1 convert()

```
template<typename R >
static std::string Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::
R >::value >::type >::convert (
    R const & range ) [inline], [static]
```

Definition at line 2041 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.112 Catch::StringMaker< signed char > Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

- [static](#) std::string [convert](#) ([signed char c](#))

6.112.1 Detailed Description

Definition at line 1770 of file [catch.h](#).

6.112.2 Member Function Documentation

6.112.2.1 convert()

```
static std::string Catch::StringMaker< signed char >::convert (
    signed char c ) [static]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

Static Public Member Functions

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

6.113.1 Detailed Description

```
template<int SZ>
struct Catch::StringMaker< signed char[SZ]>
```

Definition at line 1717 of file [catch.h](#).

6.113.2 Member Function Documentation

6.113.2.1 convert()

```
template<int SZ>
static std::string Catch::StringMaker< signed char[SZ]>::convert (
    signed char const * str ) [inline], [static]
```

Definition at line 1718 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

Static Public Member Functions

- `static std::string convert (std::nullptr_t)`

6.114.1 Detailed Description

Definition at line 1779 of file [catch.h](#).

6.114.2 Member Function Documentation

6.114.2.1 convert()

```
static std::string Catch::StringMaker< std::nullptr_t >::convert (
    std::nullptr_t ) [static]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

Static Public Member Functions

- `static std::string convert (const std::string &str)`

6.115.1 Detailed Description

Definition at line 1665 of file [catch.h](#).

6.115.2 Member Function Documentation

6.115.2.1 convert()

```
static std::string Catch::StringMaker< std::string >::convert (  
    const std::string & str ) [static]
```

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

Static Public Member Functions

- `static std::string convert (const std::wstring &wstr)`

6.116.1 Detailed Description

Definition at line 1687 of file [catch.h](#).

6.116.2 Member Function Documentation

6.116.2.1 convert()

```
static std::string Catch::StringMaker< std::wstring >::convert (  
    const std::wstring & wstr ) [static]
```

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

Static Public Member Functions

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

6.117.1 Detailed Description

```
template<typename T>  
struct Catch::StringMaker< T * >
```

Definition at line 1796 of file [catch.h](#).

6.117.2 Member Function Documentation

6.117.2.1 convert()

```
template<typename T >  
template<typename U >  
static std::string Catch::StringMaker< T * >::convert (  
    U * p ) [inline], [static]
```

Definition at line 1798 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

Static Public Member Functions

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

6.118.1 Detailed Description

```
template<typename T, int SZ>  
struct Catch::StringMaker< T[SZ]>
```

Definition at line 2047 of file [catch.h](#).

6.118.2 Member Function Documentation

6.118.2.1 convert()

```
template<typename T , int SZ>
static std::string Catch::StringMaker< T[SZ]>::convert (
    T const(&) arr[SZ] ) [inline], [static]
```

Definition at line 2048 of file [catch.h](#).

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.119 Catch::StringMaker< unsigned char > Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

- [static](#) std::string [convert](#) (unsigned char c)

6.119.1 Detailed Description

Definition at line 1774 of file [catch.h](#).

6.119.2 Member Function Documentation

6.119.2.1 convert()

```
static std::string Catch::StringMaker< unsigned char >::convert (
    unsigned char c ) [static]
```

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

Static Public Member Functions

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

6.120.1 Detailed Description

```
template<int SZ>
struct Catch::StringMaker< unsigned char[SZ]>
```

Definition at line 1723 of file [catch.h](#).

6.120.2 Member Function Documentation

6.120.2.1 convert()

```
template<int SZ>
static std::string Catch::StringMaker< unsigned char[SZ]>::convert (
    unsigned char const * str ) [inline], [static]
```

Definition at line 1724 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.121 Catch::StringMaker< unsigned int > Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

- `static std::string convert (unsigned int value)`

6.121.1 Detailed Description

Definition at line 1748 of file [catch.h](#).

6.121.2 Member Function Documentation

6.121.2.1 convert()

```
static std::string Catch::StringMaker< unsigned int >::convert (
    unsigned int value ) [static]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.122 Catch::StringMaker< unsigned long > Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

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

6.122.1 Detailed Description

Definition at line 1752 of file [catch.h](#).

6.122.2 Member Function Documentation

6.122.2.1 convert()

```
static std::string Catch::StringMaker< unsigned long >::convert (
    unsigned long value ) [static]
```

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

Static Public Member Functions

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

6.123.1 Detailed Description

Definition at line 1756 of file [catch.h](#).

6.123.2 Member Function Documentation

6.123.2.1 convert()

```
static std::string Catch::StringMaker< unsigned long long >::convert (
    unsigned long long value ) [static]
```

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.124 Catch::StringMaker< wchar_t * > Struct Reference

```
#include <catch.h>
```

Static Public Member Functions

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

6.124.1 Detailed Description

Definition at line 1703 of file [catch.h](#).

6.124.2 Member Function Documentation

6.124.2.1 convert()

```
static std::string Catch::StringMaker< wchar_t * >::convert (
    wchar_t * str ) [static]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

Static Public Member Functions

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

6.125.1 Detailed Description

Definition at line 1699 of file [catch.h](#).

6.125.2 Member Function Documentation

6.125.2.1 convert()

```
static std::string Catch::StringMaker< wchar_t const * >::convert (
    wchar_t const * str ) [static]
```

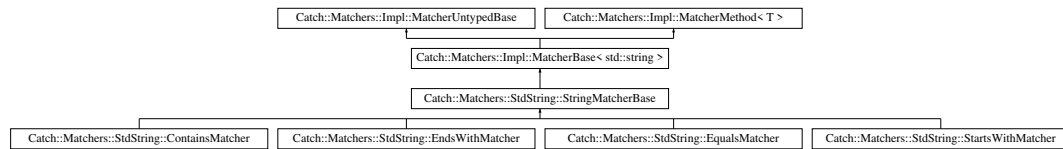
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.126 Catch::Matchers::StdString::StringMatcherBase Struct Reference

```
#include <catch.h>
```

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](#) ()=default
- [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 Member Functions inherited from [Catch::Matchers::Impl::MatcherUntypedBase](#)

- virtual [~MatcherUntypedBase](#) ()

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

- `std::string m_cachedToString`

6.126.1 Detailed Description

Definition at line 3548 of file [catch.h](#).

6.126.2 Constructor & Destructor Documentation

6.126.2.1 StringMatcherBase()

```
Catch::Matchers::StdString::StringMatcherBase::StringMatcherBase (
    std::string const & operation,
    CasedString const & comparator )
```

6.126.3 Member Function Documentation

6.126.3.1 describe()

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

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

6.126.4 Member Data Documentation

6.126.4.1 m_comparator

```
CasedString Catch::Matchers::StdString::StringMatcherBase::m_comparator
```

Definition at line 3552 of file [catch.h](#).

6.126.4.2 m_operation

```
std::string Catch::Matchers::StdString::StringMatcherBase::m_operation
```

Definition at line 3553 of file [catch.h](#).

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

- `/Users/samanthapope/msdscriptRepo/msdScript/catch.h`

6.127 Catch::StringRef Class Reference

```
#include <catch.h>
```

Public Types

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

Public Member Functions

- `constexpr StringRef () noexcept=default`
- `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`

6.127.1 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.

Definition at line 606 of file [catch.h](#).

6.127.2 Member Typedef Documentation

6.127.2.1 const_iterator

```
using Catch::StringRef::const_iterator = const char*
```

Definition at line 609 of file [catch.h](#).

6.127.2.2 size_type

```
using Catch::StringRef::size_type = std::size_t
```

Definition at line 608 of file [catch.h](#).

6.127.3 Constructor & Destructor Documentation

6.127.3.1 StringRef() [1/4]

```
constexpr Catch::StringRef::StringRef ( ) [constexpr], [default], [noexcept]
```

6.127.3.2 StringRef() [2/4]

```
Catch::StringRef::StringRef (
    char const * rawChars ) [noexcept]
```

6.127.3.3 StringRef() [3/4]

```
constexpr Catch::StringRef::StringRef (
    char const * rawChars,
    size_type size ) [inline], [constexpr], [noexcept]
```

Definition at line 622 of file [catch.h](#).

6.127.3.4 StringRef() [4/4]

```
Catch::StringRef::StringRef (
    std::string const & stdString ) [inline], [noexcept]
```

Definition at line 627 of file [catch.h](#).

6.127.4 Member Function Documentation

6.127.4.1 begin()

```
constexpr const_iterator Catch::StringRef::begin ( ) const [inline], [constexpr]
```

Definition at line 673 of file [catch.h](#).

6.127.4.2 c_str()

```
auto Catch::StringRef::c_str ( ) const -> char const *
```

6.127.4.3 data()

```
auto Catch::StringRef::data ( ) const -> char const * [noexcept]
```


6.127.4.4 empty()

```
constexpr auto Catch::StringRef::empty ( ) const -> bool    [inline], [constexpr], [noexcept]
```

Definition at line 648 of file [catch.h](#).

6.127.4.5 end()

```
constexpr const_iterator Catch::StringRef::end ( ) const    [inline], [constexpr]
```

Definition at line 674 of file [catch.h](#).

6.127.4.6 isNullTerminated()

```
constexpr auto Catch::StringRef::isNullTerminated ( ) const -> bool    [inline], [constexpr],  
[noexcept]
```

Definition at line 668 of file [catch.h](#).

6.127.4.7 operator std::string()

```
Catch::StringRef::operator std::string ( ) const    [inline], [explicit]
```

Definition at line 632 of file [catch.h](#).

6.127.4.8 operator"!=()

```
auto Catch::StringRef::operator!= (   
    StringRef const & other ) const -> bool    [inline], [noexcept]
```

Definition at line 638 of file [catch.h](#).

6.127.4.9 operator==(())

```
auto Catch::StringRef::operator==(   
    StringRef const & other ) const -> bool    [noexcept]
```

6.127.4.10 operator[]()

```
auto Catch::StringRef::operator[] (   
    size_type index ) const -> char    [inline], [noexcept]
```

Definition at line 642 of file [catch.h](#).

6.127.4.11 size()

```
constexpr auto Catch::StringRef::size ( ) const -> size_type [inline], [constexpr], [noexcept]
```

Definition at line 651 of file [catch.h](#).

6.127.4.12 substr()

```
auto Catch::StringRef::substr (
    size_type start,
    size_type length ) const -> StringRef [noexcept]
```

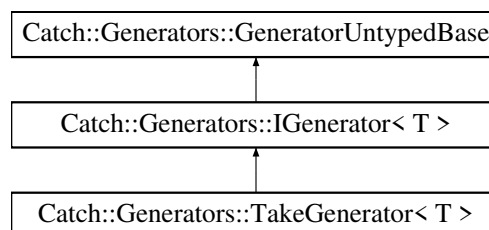
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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

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

- [virtual](#) [~IGenerator](#) ()=default

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

- [GeneratorUntypedBase](#) ()=default
- [virtual](#) [~GeneratorUntypedBase](#) ()

Additional Inherited Members

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

- [using type = T](#)

6.128.1 Detailed Description

```
template<typename T>
class Catch::Generators::TakeGenerator< T >
```

Definition at line 4120 of file [catch.h](#).

6.128.2 Constructor & Destructor Documentation

6.128.2.1 TakeGenerator()

```
template<typename T >
Catch::Generators::TakeGenerator< T >::TakeGenerator (
    size_t target,
    GeneratorWrapper< T > && generator ) [inline]
```

Definition at line 4125 of file [catch.h](#).

6.128.3 Member Function Documentation

6.128.3.1 get()

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

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

Definition at line 4131 of file [catch.h](#).

6.128.3.2 next()

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

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

Definition at line 4134 of file [catch.h](#).

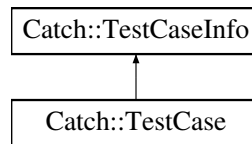
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.129 Catch::TestCase Class Reference

```
#include <catch.h>
```

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

6.129.1 Detailed Description

Definition at line 4807 of file [catch.h](#).

6.129.2 Constructor & Destructor Documentation

6.129.2.1 TestCase()

```
Catch::TestCase::TestCase (
    ITestInvoker * testCase,
    TestCaseInfo && info )
```

6.129.3 Member Function Documentation

6.129.3.1 getTestCaseInfo()

```
TestCaseInfo const & Catch::TestCase::getTestCaseInfo ( ) const
```

6.129.3.2 invoke()

```
void Catch::TestCase::invoke ( ) const
```

6.129.3.3 operator<()

```
bool Catch::TestCase::operator< (
    TestCase const & other ) const
```

6.129.3.4 operator==()

```
bool Catch::TestCase::operator== (
    TestCase const & other ) const
```

6.129.3.5 withName()

```
TestCase Catch::TestCase::withName (
    std::string const & _newName ) const
```

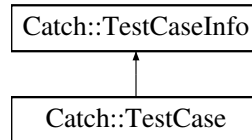
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.130 Catch::TestCaseInfo Struct Reference

```
#include <catch.h>
```

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](#))

6.130.1 Detailed Description

Definition at line [4772](#) of file [catch.h](#).

6.130.2 Member Enumeration Documentation

6.130.2.1 SpecialProperties

```
enum Catch::TestCaseInfo::SpecialProperties
```

Enumerator

None	
IsHidden	
ShouldFail	
MayFail	
Throws	
NonPortable	
Benchmark	

Definition at line 4773 of file [catch.h](#).

6.130.3 Constructor & Destructor Documentation

6.130.3.1 TestCaseInfo()

```
Catch::TestCaseInfo::TestCaseInfo (
    std::string const & _name,
    std::string const & _className,
    std::string const & _description,
    std::vector< std::string > const & _tags,
    SourceLineInfo const & _lineInfo )
```

6.130.4 Member Function Documentation

6.130.4.1 expectedToFail()

```
bool Catch::TestCaseInfo::expectedToFail ( ) const
```

6.130.4.2 isHidden()

```
bool Catch::TestCaseInfo::isHidden ( ) const
```

6.130.4.3 okToFail()

```
bool Catch::TestCaseInfo::okToFail ( ) const
```

6.130.4.4 tagsAsString()

```
std::string Catch::TestCaseInfo::tagsAsString ( ) const
```

6.130.4.5 throws()

```
bool Catch::TestCaseInfo::throws ( ) const
```

6.130.5 Friends And Related Symbol Documentation

6.130.5.1 setTags

```
void setTags (
    TestCaseInfo & testCaseInfo,
    std::vector< std::string > tags ) [friend]
```

6.130.6 Member Data Documentation

6.130.6.1 className

```
std::string Catch::TestCaseInfo::className
```

Definition at line 4799 of file [catch.h](#).

6.130.6.2 description

```
std::string Catch::TestCaseInfo::description
```

Definition at line 4800 of file [catch.h](#).

6.130.6.3 lcaseTags

```
std::vector<std::string> Catch::TestCaseInfo::lcaseTags
```

Definition at line 4802 of file [catch.h](#).

6.130.6.4 lineInfo

```
SourceLineInfo Catch::TestCaseInfo::lineInfo
```

Definition at line 4803 of file [catch.h](#).

6.130.6.5 name

```
std::string Catch::TestCaseInfo::name
```

Definition at line 4798 of file [catch.h](#).

6.130.6.6 properties

```
SpecialProperties Catch::TestCaseInfo::properties
```

Definition at line 4804 of file [catch.h](#).

6.130.6.7 tags

```
std::vector<std::string> Catch::TestCaseInfo::tags
```

Definition at line 4801 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.131 Catch::TestFailureException Struct Reference

```
#include <catch.h>
```

6.131.1 Detailed Description

Definition at line 2521 of file [catch.h](#).

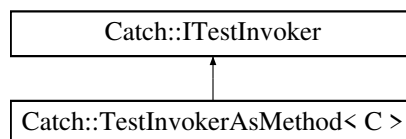
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.132 Catch::TestInvokerAsMethod< C > Class Template Reference

```
#include <catch.h>
```

Inheritance diagram for Catch::TestInvokerAsMethod< C >:



Public Member Functions

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

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

- [virtual ~ITestInvoker](#) ()

6.132.1 Detailed Description

```
template<typename C>
class Catch::TestInvokerAsMethod< C >
```

Definition at line 962 of file [catch.h](#).

6.132.2 Constructor & Destructor Documentation

6.132.2.1 TestInvokerAsMethod()

```
template<typename C >
Catch::TestInvokerAsMethod< C >::TestInvokerAsMethod (
    void(C::*)() testAsMethod ) [inline], [noexcept]
```

Definition at line 965 of file [catch.h](#).

6.132.3 Member Function Documentation

6.132.3.1 invoke()

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

Implements [Catch::ITestInvoker](#).

Definition at line 967 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.133 Catch::Timer Class Reference

```
#include <catch.h>
```

Public Member Functions

- [void start \(\)](#)
- [auto getElapsedNanoseconds \(\) const -> uint64_t](#)
- [auto getElapsedMicroseconds \(\) const -> uint64_t](#)
- [auto getElapsedMilliseconds \(\) const -> unsigned int](#)
- [auto getElapsedSeconds \(\) const -> double](#)

6.133.1 Detailed Description

Definition at line 2894 of file [catch.h](#).

6.133.2 Member Function Documentation

6.133.2.1 getElapsedMicroseconds()

```
auto Catch::Timer::getElapsedMicroseconds ( ) const -> uint64_t
```

6.133.2.2 getElapsedMilliseconds()

```
auto Catch::Timer::getElapsedMilliseconds ( ) const -> unsigned int
```

6.133.2.3 getElapsedNanoseconds()

```
auto Catch::Timer::getElapsedNanoseconds ( ) const -> uint64_t
```

6.133.2.4 getElapsedSeconds()

```
auto Catch::Timer::getElapsedSeconds ( ) const -> double
```

6.133.2.5 start()

```
void Catch::Timer::start ( )
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.134 Catch::Totals Struct Reference

```
#include <catch.h>
```

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](#)

6.134.1 Detailed Description

Definition at line 2842 of file [catch.h](#).

6.134.2 Member Function Documentation

6.134.2.1 `delta()`

```
Totals Catch::Totals::delta (
    Totals const & prevTotals ) const
```

6.134.2.2 `operator+=()`

```
Totals & Catch::Totals::operator+= (
    Totals const & other )
```

6.134.2.3 `operator-()`

```
Totals Catch::Totals::operator- (
    Totals const & other ) const
```

6.134.3 Member Data Documentation

6.134.3.1 `assertions`

```
Counts Catch::Totals::assertions
```

Definition at line 2850 of file [catch.h](#).

6.134.3.2 `error`

```
int Catch::Totals::error = 0
```

Definition at line 2849 of file [catch.h](#).

6.134.3.3 `testCases`

```
Counts Catch::Totals::testCases
```

Definition at line 2851 of file [catch.h](#).

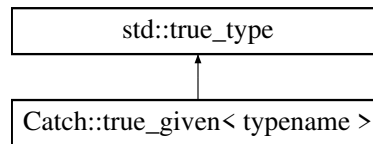
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.135 Catch::true_given< typename > Struct Template Reference

```
#include <catch.h>
```

Inheritance diagram for Catch::true_given< typename >:



6.135.1 Detailed Description

```
template<typename>
struct Catch::true_given< typename >
```

Definition at line 927 of file [catch.h](#).

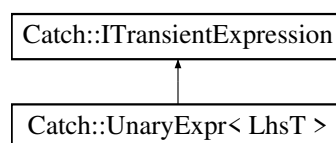
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.136 Catch::UnaryExpr< LhsT > Class Template Reference

```
#include <catch.h>
```

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)
- [virtual ~ITransientExpression](#) ()

Additional Inherited Members

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

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

6.136.1 Detailed Description

```
template<typename LhsT>
class Catch::UnaryExpr< LhsT >
```

Definition at line 2300 of file [catch.h](#).

6.136.2 Constructor & Destructor Documentation

6.136.2.1 UnaryExpr()

```
template<typename LhsT >
Catch::UnaryExpr< LhsT >::UnaryExpr (
    LhsT lhs ) [inline], [explicit]
```

Definition at line 2308 of file [catch.h](#).

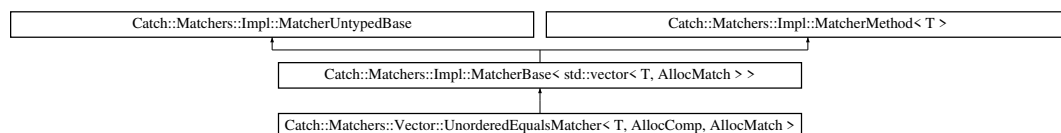
The documentation for this class was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

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

```
#include <catch.h>
```

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 \(\)=default](#)
- [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 ~MatcherUntypedBase \(\)](#)

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

- [std::string m_cachedToString](#)

6.137.1 Detailed Description

```
template<typename T, typename AllocComp, typename AllocMatch>
struct Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >
```

Definition at line 3717 of file [catch.h](#).

6.137.2 Constructor & Destructor Documentation**6.137.2.1 UnorderedEqualsMatcher()**

```
template<typename T , typename AllocComp , typename AllocMatch >
Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >::UnorderedEquals↵
Matcher (
    std::vector< T, AllocComp > const & target ) [inline]
```

Definition at line 3718 of file [catch.h](#).

6.137.3 Member Function Documentation

6.137.3.1 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](#).

Definition at line 3726 of file [catch.h](#).

6.137.3.2 match()

```
template<typename T , typename AllocComp , typename AllocMatch >
bool Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >::match (
    std::vector< T, AllocMatch > const & vec ) const [inline], [override]
```

Definition at line 3719 of file [catch.h](#).

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.138 Catch::UseColour Struct Reference

```
#include <catch.h>
```

Public Types

- enum [YesOrNo](#) { [Auto](#) , [Yes](#) , [No](#) }

6.138.1 Detailed Description

Definition at line 4498 of file [catch.h](#).

6.138.2 Member Enumeration Documentation

6.138.2.1 YesOrNo

```
enum Catch::UseColour::YesOrNo
```

Enumerator

Auto	
Yes	
No	

Definition at line 4498 of file [catch.h](#).

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.139 Var Class Reference

```
#include <Expr.h>
```

Inheritance diagram for Var:



Public Member Functions

- [Var](#) (const std::string &varPassed)
- bool [equals](#) ([Expr](#) *e) override
- int [interp](#) () override
- bool [hasVariable](#) () override
- [Expr](#) * [subst](#) (std::string stringInput, [Expr](#) *e) override
- void [print](#) (std::ostream &stream) override

Public Member Functions inherited from [Expr](#)

- virtual [~Expr](#) ()=default
- std::string [to_string](#) ()
- std::string [to_pp_string](#) ()
- void [pretty_print_at](#) (std::ostream &ot)

Public Attributes

- std::string [var](#)

Protected Member Functions

- void [pretty_print](#) (std::ostream &ot, [precedence_t](#) prec) override

6.139.1 Detailed Description

Definition at line 91 of file [Expr.h](#).

6.139.2 Constructor & Destructor Documentation

6.139.2.1 Var()

```
Var::Var (
    const std::string & varPassed )
```

Constructs a [Var](#) object with a given variable name.

Parameters

<i>varPassed</i>	The name of the variable.
------------------	---------------------------

Definition at line 36 of file [Expr.cpp](#).

6.139.3 Member Function Documentation

6.139.3.1 equals()

```
bool Var::equals (
    Expr * e ) [override], [virtual]
```

Checks if this [Var](#) object is equal to another expression.

Parameters

<i>e</i>	Pointer to the expression to compare with.
----------	--

Returns

true if the expressions are equivalent, false otherwise.

Implements [Expr](#).

Definition at line 76 of file [Expr.cpp](#).

6.139.3.2 hasVariable()

```
bool Var::hasVariable ( ) [override], [virtual]
```

Checks if the [Var](#) expression contains a variable.

Returns

true because [Var](#) represents a variable.

Implements [Expr](#).

Definition at line 141 of file [Expr.cpp](#).

6.139.3.3 interp()

```
int Var::interp ( ) [override], [virtual]
```

Evaluates the variable expression. Throws an error because variables cannot be directly evaluated.

Exceptions

<code>std::runtime_error</code>	when trying to evaluate a variable.
---------------------------------	-------------------------------------

Implements [Expr](#).

Definition at line 109 of file [Expr.cpp](#).

6.139.3.4 pretty_print()

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

Pretty prints the [Var](#) expression with appropriate precedence.

Parameters

<i>ot</i>	The output stream to print to.
<i>prec</i>	The precedence context in which this expression is being printed.

Implements [Expr](#).

Definition at line 275 of file [Expr.cpp](#).

6.139.3.5 print()

```
void Var::print (
    std::ostream & stream ) [override], [virtual]
```

Prints the [Var](#) expression to a given output stream.

Parameters

<i>stream</i>	The output stream to print to.
---------------	--------------------------------

Implements [Expr](#).

Definition at line 229 of file [Expr.cpp](#).

6.139.3.6 subst()

```
Expr * Var::subst (
    std::string stringInput,
    Expr * e ) [override], [virtual]
```

Substitutes a variable with another expression in a [Var](#) object.

Parameters

<i>stringInput</i>	The name of the variable to substitute.
<i>e</i>	The expression to substitute the variable with.

Returns

A new [Var](#) object with the variable substituted if the names match, otherwise returns a copy of itself.

Implements [Expr](#).

Definition at line 185 of file [Expr.cpp](#).

6.139.4 Member Data Documentation

6.139.4.1 var

```
std::string Var::var
```

Definition at line 93 of file [Expr.h](#).

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

- [/Users/samanthapope/msdscripRepo/msdScript/Expr.h](#)
- [/Users/samanthapope/msdscripRepo/msdScript/Expr.cpp](#)

6.140 Catch::detail::void_type<... > Struct Template Reference

```
#include <catch.h>
```

Public Types

- [using type = void](#)

6.140.1 Detailed Description

```
template<typename...>
struct Catch::detail::void_type<... >
```

Definition at line 1993 of file [catch.h](#).

6.140.2 Member Typedef Documentation

6.140.2.1 type

```
template<typename... >
using Catch::detail::void_type<... >::type = void
```

Definition at line 1994 of file [catch.h](#).

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.141 Catch::WaitForKeypress Struct Reference

```
#include <catch.h>
```

Public Types

- enum [When](#) { [Never](#) , [BeforeStart](#) = 1 , [BeforeExit](#) = 2 , [BeforeStartAndExit](#) = BeforeStart | BeforeExit }

6.141.1 Detailed Description

Definition at line 4503 of file [catch.h](#).

6.141.2 Member Enumeration Documentation

6.141.2.1 When

```
enum Catch::WaitForKeypress::When
```

Enumerator

Never	
BeforeStart	
BeforeExit	
BeforeStartAndExit	

Definition at line 4503 of file [catch.h](#).

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

- [/Users/samanthapope/msdscripRepo/msdScript/catch.h](#)

6.142 Catch::WarnAbout Struct Reference

```
#include <catch.h>
```

Public Types

- enum [What](#) { [Nothing](#) = 0x00 , [NoAssertions](#) = 0x01 , [NoTests](#) = 0x02 }

6.142.1 Detailed Description

Definition at line [4482](#) of file [catch.h](#).

6.142.2 Member Enumeration Documentation

6.142.2.1 What

```
enum Catch::WarnAbout::What
```

Enumerator

Nothing	
NoAssertions	
NoTests	

Definition at line [4482](#) of file [catch.h](#).

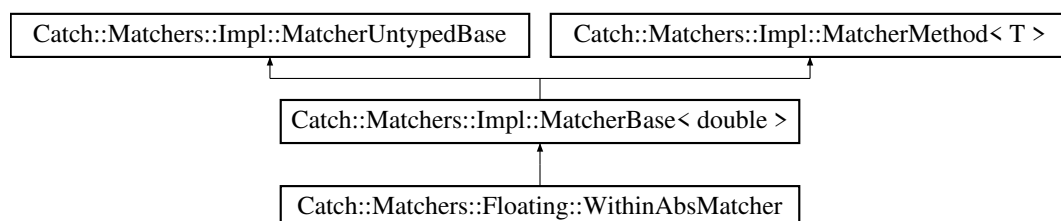
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.143 Catch::Matchers::Floating::WithinAbsMatcher Struct Reference

```
#include <catch.h>
```

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](#) ()=default
- [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 ~MatcherUntypedBase](#) ()

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

- [std::string m_cachedToString](#)

6.143.1 Detailed Description

Definition at line [3429](#) of file [catch.h](#).

6.143.2 Constructor & Destructor Documentation**6.143.2.1 WithinAbsMatcher()**

```
Catch::Matchers::Floating::WithinAbsMatcher::WithinAbsMatcher (
    double target,
    double margin )
```

6.143.3 Member Function Documentation

6.143.3.1 describe()

```
std::string Catch::Matchers::Floating::WithinAbsMatcher::describe ( ) const [override], [virtual]
```

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

6.143.3.2 match()

```
bool Catch::Matchers::Floating::WithinAbsMatcher::match (
    double const & matchee ) const [override]
```

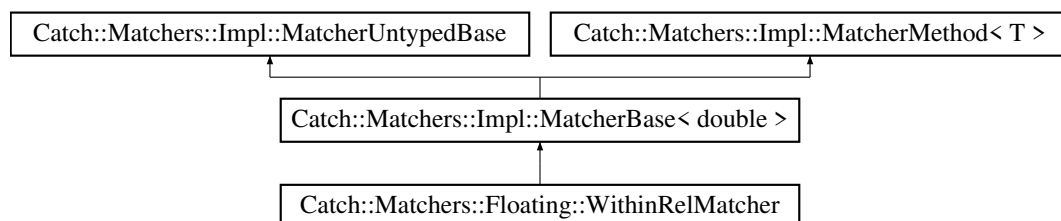
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.144 Catch::Matchers::Floating::WithinRelMatcher Struct Reference

```
#include <catch.h>
```

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](#) ()=default
- [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 ~MatcherUntypedBase](#) ()

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

- [std::string m_cachedToString](#)

6.144.1 Detailed Description

Definition at line 3454 of file [catch.h](#).

6.144.2 Constructor & Destructor Documentation**6.144.2.1 WithinRelMatcher()**

```
Catch::Matchers::Floating::WithinRelMatcher::WithinRelMatcher (
    double target,
    double epsilon )
```

6.144.3 Member Function Documentation**6.144.3.1 describe()**

```
std::string Catch::Matchers::Floating::WithinRelMatcher::describe ( ) const [override], [virtual]
```

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

6.144.3.2 match()

```
bool Catch::Matchers::Floating::WithinRelMatcher::match (
    double const & matchee ) const [override]
```

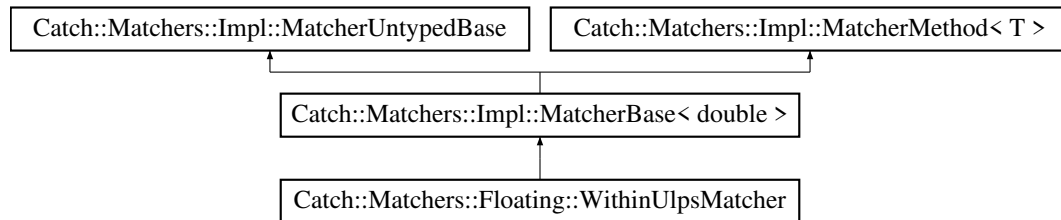
The documentation for this struct was generated from the following file:

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

6.145 Catch::Matchers::Floating::WithinUlpMatcher Struct Reference

```
#include <catch.h>
```

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](#) ()=default
- [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 ~MatcherUntypedBase](#) ()

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

- [std::string m_cachedToString](#)

6.145.1 Detailed Description

Definition at line 3438 of file [catch.h](#).

6.145.2 Constructor & Destructor Documentation

6.145.2.1 WithinUlpMatcher()

```
Catch::Matchers::Floating::WithinUlpMatcher::WithinUlpMatcher (
    double target,
    uint64_t ulps,
    FloatingPointKind baseType )
```

6.145.3 Member Function Documentation

6.145.3.1 describe()

```
std::string Catch::Matchers::Floating::WithinUlpMatcher::describe ( ) const [override],
[virtual]
```

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

6.145.3.2 match()

```
bool Catch::Matchers::Floating::WithinUlpMatcher::match (
    double const & matchee ) const [override]
```

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

- [/Users/samanthapope/msdscriptRepo/msdScript/catch.h](#)

Chapter 7

File Documentation

7.1 /Users/samanthapope/msdscriptRepo/msdScript/catch.h File Reference

```
#include <iosfwd>
#include <string>
#include <cstdint>
#include <vector>
#include <cstddef>
#include <cassert>
#include <type_traits>
#include <ostream>
#include <chrono>
#include <memory>
#include <exception>
#include <functional>
#include <algorithm>
#include <utility>
#include <random>
```

Classes

- struct [Catch_global_namespace_dummy](#)
- struct [Catch::CaseSensitive](#)
- class [Catch::NonCopyable](#)
- struct [Catch::SourceLineInfo](#)
- struct [Catch::StreamEndStop](#)
- struct [Catch::RegistrarForTagAliases](#)
- struct [Catch::ITestInvoker](#)
- struct [Catch::ITestCaseRegistry](#)
- class [Catch::StringRef](#)
- struct [Catch::always_false< T >](#)
- struct [Catch::true_given< typename >](#)
- struct [Catch::is_callable_tester](#)
- struct [Catch::is_callable< Fun\(Args...\)>](#)
- class [Catch::TestInvokerAsMethod< C >](#)
- struct [Catch::NameAndTags](#)

- struct [Catch::AutoReg](#)
- struct [Catch::ResultWas](#)
- struct [Catch::ResultDisposition](#)
- struct [Catch::AssertionInfo](#)
- struct [Catch::IStream](#)
- class [Catch::ReusableStringStream](#)
- struct [Catch::Detail::EnumInfo](#)
- struct [Catch::IMutableEnumValuesRegistry](#)
- class [Catch::Detail::IsStreamInsertable< T >](#)
- struct [Catch::StringMaker< T, typename >](#)
- struct [Catch::StringMaker< std::string >](#)
- struct [Catch::StringMaker< char const * >](#)
- struct [Catch::StringMaker< char * >](#)
- struct [Catch::StringMaker< std::wstring >](#)
- struct [Catch::StringMaker< wchar_t const * >](#)
- struct [Catch::StringMaker< wchar_t * >](#)
- struct [Catch::StringMaker< char\[SZ\]>](#)
- struct [Catch::StringMaker< signed char\[SZ\]>](#)
- struct [Catch::StringMaker< unsigned char\[SZ\]>](#)
- struct [Catch::StringMaker< int >](#)
- struct [Catch::StringMaker< long >](#)
- struct [Catch::StringMaker< long long >](#)
- struct [Catch::StringMaker< unsigned int >](#)
- struct [Catch::StringMaker< unsigned long >](#)
- struct [Catch::StringMaker< unsigned long long >](#)
- struct [Catch::StringMaker< bool >](#)
- struct [Catch::StringMaker< char >](#)
- struct [Catch::StringMaker< signed char >](#)
- struct [Catch::StringMaker< unsigned char >](#)
- struct [Catch::StringMaker< std::nullptr_t >](#)
- struct [Catch::StringMaker< float >](#)
- struct [Catch::StringMaker< double >](#)
- struct [Catch::StringMaker< T * >](#)
- struct [Catch::StringMaker< R C::* >](#)
- struct [Catch::detail::void_type<... >](#)
- struct [Catch::detail::is_range_impl< T, typename >](#)
- struct [Catch::detail::is_range_impl< T, typename void_type< decltype\(begin\(std::declval< T >\(\)\)\)>::type >](#)
- struct [Catch::is_range< T >](#)
- struct [Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStreamInsertable< R >::](#)
- struct [Catch::StringMaker< T\[SZ\]>](#)
- struct [Catch::ITransientExpression](#)
- class [Catch::BinaryExpr< LhsT, RhsT >](#)
- class [Catch::UnaryExpr< LhsT >](#)
- class [Catch::ExprLhs< LhsT >](#)
- struct [Catch::Decomposer](#)
- struct [Catch::IResultCapture](#)
- struct [Catch::TestFailureException](#)
- class [Catch::LazyExpression](#)
- struct [Catch::AssertionReaction](#)
- class [Catch::AssertionHandler](#)
- struct [Catch::MessageInfo](#)
- struct [Catch::MessageStream](#)
- struct [Catch::MessageBuilder](#)
- class [Catch::ScopedMessage](#)
- class [Catch::Capturer](#)

- struct [Catch::Counts](#)
- struct [Catch::Totals](#)
- struct [Catch::SectionInfo](#)
- struct [Catch::SectionEndInfo](#)
- class [Catch::Timer](#)
- class [Catch::Section](#)
- struct [Catch::IRegistryHub](#)
- struct [Catch::IMutableRegistryHub](#)
- struct [Catch::IExceptionTranslator](#)
- struct [Catch::IExceptionTranslatorRegistry](#)
- class [Catch::ExceptionTranslatorRegistrar](#)
- class [Catch::Detail::Approx](#)
- struct [Catch::StringMaker< Catch::Detail::Approx >](#)
- struct [Catch::pluralise](#)
- class [Catch::Matchers::Impl::MatcherUntypedBase](#)
- struct [Catch::Matchers::Impl::MatcherMethod< ObjectT >](#)
- struct [Catch::Matchers::Impl::MatcherBase< T >](#)
- struct [Catch::Matchers::Impl::MatchAllOf< ArgT >](#)
- struct [Catch::Matchers::Impl::MatchAnyOf< ArgT >](#)
- struct [Catch::Matchers::Impl::MatchNotOf< ArgT >](#)
- class [Catch::Matchers::Exception::ExceptionMessageMatcher](#)
- struct [Catch::Matchers::Floating::WithinAbsMatcher](#)
- struct [Catch::Matchers::Floating::WithinUlpMatcher](#)
- struct [Catch::Matchers::Floating::WithinRelMatcher](#)
- class [Catch::Matchers::Generic::PredicateMatcher< T >](#)
- struct [Catch::Matchers::StdString::CasedString](#)
- struct [Catch::Matchers::StdString::StringMatcherBase](#)
- struct [Catch::Matchers::StdString::EqualsMatcher](#)
- struct [Catch::Matchers::StdString::ContainsMatcher](#)
- struct [Catch::Matchers::StdString::StartsWithMatcher](#)
- struct [Catch::Matchers::StdString::EndsWithMatcher](#)
- struct [Catch::Matchers::StdString::RegexMatcher](#)
- struct [Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >](#)
- struct [Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >](#)
- struct [Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >](#)
- struct [Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >](#)
- struct [Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >](#)
- class [Catch::MatchExpr< ArgT, MatcherT >](#)
- class [Catch::Generators::GeneratorUntypedBase](#)
- struct [Catch::IGeneratorTracker](#)
- class [Catch::GeneratorException](#)
- struct [Catch::Generators::IGenerator< T >](#)
- class [Catch::Generators::SingleValueGenerator< T >](#)
- class [Catch::Generators::FixedValuesGenerator< T >](#)
- class [Catch::Generators::GeneratorWrapper< T >](#)
- class [Catch::Generators::Generators< T >](#)
- struct [Catch::Generators::as< T >](#)
- class [Catch::Generators::TakeGenerator< T >](#)
- class [Catch::Generators::FilterGenerator< T, Predicate >](#)
- class [Catch::Generators::RepeatGenerator< T >](#)
- class [Catch::Generators::MapGenerator< T, U, Func >](#)
- class [Catch::Generators::ChunkGenerator< T >](#)
- struct [Catch::IContext](#)
- struct [Catch::IMutableContext](#)
- class [Catch::Option< T >](#)

- struct [Catch::WarnAbout](#)
- struct [Catch::ShowDurations](#)
- struct [Catch::RunTests](#)
- struct [Catch::UseColour](#)
- struct [Catch::WaitForKeypress](#)
- struct [Catch::IConfig](#)
- class [Catch::SimplePcg32](#)
- class [Catch::Generators::RandomFloatingGenerator< Float >](#)
- class [Catch::Generators::RandomIntegerGenerator< Integer >](#)
- class [Catch::Generators::RangeGenerator< T >](#)
- class [Catch::Generators::IteratorGenerator< T >](#)
- struct [Catch::TestCaseInfo](#)
- class [Catch::TestCase](#)
- struct [Catch::IRunner](#)

Namespaces

- namespace [Catch](#)
- namespace [mpl_](#)
- namespace [Catch::Detail](#)
- namespace [Catch::detail](#)
- namespace [Catch::literals](#)
- namespace [Catch::Matchers](#)
- namespace [Catch::Matchers::Impl](#)
- namespace [Catch::Matchers::Exception](#)
- namespace [Catch::Matchers::Floating](#)
- namespace [Catch::Matchers::Generic](#)
- namespace [Catch::Matchers::Generic::Detail](#)
- namespace [Catch::Matchers::StdString](#)
- namespace [Catch::Matchers::Vector](#)
- namespace [Catch::Generators](#)
- namespace [Catch::Generators::pf](#)

Macros

- [#define CATCH_VERSION_MAJOR 2](#)
- [#define CATCH_VERSION_MINOR 13](#)
- [#define CATCH_VERSION_PATCH 10](#)
- [#define CATCH_INTERNAL_CONFIG_POSIX_SIGNALS](#)
- [#define CATCH_INTERNAL_CONFIG_COUNTER](#)
- [#define CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER](#)
- [#define CATCH_CONFIG_COUNTER](#)
- [#define CATCH_CONFIG_POSIX_SIGNALS](#)
- [#define CATCH_CONFIG_WCHAR](#)
- [#define CATCH_CONFIG_CPP11_TO_STRING](#)
- [#define CATCH_CONFIG_DISABLE_EXCEPTIONS](#)
- [#define CATCH_CONFIG_GLOBAL_NEXTAFTER](#)
- [#define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION](#)
- [#define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION](#)
- [#define CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS](#)
- [#define CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS](#)
- [#define CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS](#)
- [#define CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS](#)

Generated by Doxygen

- `#define INTERNAL_CATCH_STRINGIZE2(...) #__VA_ARGS__`
- `#define INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS(param) INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_STRINGIZE2(param))`
- `#define INTERNAL_CATCH_MAKE_NAMESPACE2(...) ns_##__VA_ARGS__`
- `#define INTERNAL_CATCH_MAKE_NAMESPACE(name) INTERNAL_CATCH_MAKE_NAMESPACE2(name)`
- `#define INTERNAL_CATCH_REMOVE_PARENS(...) INTERNAL_CATCH_EXPAND1(INTERNAL_CATCH_DEF __VA_ARGS__)`
- `#define INTERNAL_CATCH_MAKE_TYPE_LIST2(...) decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS_GEN(INTERNAL_CATCH_STRINGIZE2(__VA_ARGS__))>())`
- `#define INTERNAL_CATCH_MAKE_TYPE_LIST(...) INTERNAL_CATCH_MAKE_TYPE_LIST2(INTERNAL_CATCH_REMOVE_PARENS(INTERNAL_CATCH_STRINGIZE2(__VA_ARGS__)))`
- `#define INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(...) CATCH_REC_LIST(INTERNAL_CATCH_MAKE_TYPE_LIST2(INTERNAL_CATCH_REMOVE_PARENS(INTERNAL_CATCH_STRINGIZE2(__VA_ARGS__))))`
- `#define INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_0) INTERNAL_CATCH_REMOVE_PARENS(_0)`
- `#define INTERNAL_CATCH_REMOVE_PARENS_2_ARG(_0, _1) INTERNAL_CATCH_REMOVE_PARENS(INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_0), INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_1))`
- `#define INTERNAL_CATCH_REMOVE_PARENS_3_ARG(_0, _1, _2) INTERNAL_CATCH_REMOVE_PARENS(INTERNAL_CATCH_REMOVE_PARENS_2_ARG(_0, _1), INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_2))`
- `#define INTERNAL_CATCH_REMOVE_PARENS_4_ARG(_0, _1, _2, _3) INTERNAL_CATCH_REMOVE_PARENS(INTERNAL_CATCH_REMOVE_PARENS_3_ARG(_0, _1, _2), INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_3))`
- `#define INTERNAL_CATCH_REMOVE_PARENS_5_ARG(_0, _1, _2, _3, _4) INTERNAL_CATCH_REMOVE_PARENS(INTERNAL_CATCH_REMOVE_PARENS_4_ARG(_0, _1, _2, _3), INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_4))`
- `#define INTERNAL_CATCH_REMOVE_PARENS_6_ARG(_0, _1, _2, _3, _4, _5) INTERNAL_CATCH_REMOVE_PARENS(INTERNAL_CATCH_REMOVE_PARENS_5_ARG(_0, _1, _2, _3, _4), INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_5))`
- `#define INTERNAL_CATCH_REMOVE_PARENS_7_ARG(_0, _1, _2, _3, _4, _5, _6) INTERNAL_CATCH_REMOVE_PARENS(INTERNAL_CATCH_REMOVE_PARENS_6_ARG(_0, _1, _2, _3, _4, _5), INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_6))`
- `#define INTERNAL_CATCH_REMOVE_PARENS_8_ARG(_0, _1, _2, _3, _4, _5, _6, _7) INTERNAL_CATCH_REMOVE_PARENS(INTERNAL_CATCH_REMOVE_PARENS_7_ARG(_0, _1, _2, _3, _4, _5, _6), INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_7))`
- `#define INTERNAL_CATCH_REMOVE_PARENS_9_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8) INTERNAL_CATCH_REMOVE_PARENS(INTERNAL_CATCH_REMOVE_PARENS_8_ARG(_0, _1, _2, _3, _4, _5, _6, _7), INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_8))`
- `#define INTERNAL_CATCH_REMOVE_PARENS_10_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9) INTERNAL_CATCH_REMOVE_PARENS(INTERNAL_CATCH_REMOVE_PARENS_9_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8), INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_9))`
- `#define INTERNAL_CATCH_REMOVE_PARENS_11_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10) INTERNAL_CATCH_REMOVE_PARENS(INTERNAL_CATCH_REMOVE_PARENS_10_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9), INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_10))`
- `#define INTERNAL_CATCH_VA_NARGS_IMPL(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, N, ...) N`
- `#define INTERNAL_CATCH_TYPE_GEN`
- `#define INTERNAL_CATCH_NTTP_1(signature, ...)`
- `#define INTERNAL_CATCH_DECLARE_SIG_TEST0(TestName)`
- `#define INTERNAL_CATCH_DECLARE_SIG_TEST1(TestName, signature)`
- `#define INTERNAL_CATCH_DECLARE_SIG_TEST_X(TestName, signature, ...)`
- `#define INTERNAL_CATCH_DEFINE_SIG_TEST0(TestName)`
- `#define INTERNAL_CATCH_DEFINE_SIG_TEST1(TestName, signature)`
- `#define INTERNAL_CATCH_DEFINE_SIG_TEST_X(TestName, signature, ...)`
- `#define INTERNAL_CATCH_NTTP_REGISTER0(TestFunc, signature)`
- `#define INTERNAL_CATCH_NTTP_REGISTER(TestFunc, signature, ...)`
- `#define INTERNAL_CATCH_NTTP_REGISTER_METHOD0(TestName, signature, ...)`
- `#define INTERNAL_CATCH_NTTP_REGISTER_METHOD(TestName, signature, ...)`
- `#define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0(TestName, ClassName)`
- `#define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1(TestName, ClassName, signature)`
- `#define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X(TestName, ClassName, signature, ...)`
- `#define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0(TestName)`
- `#define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1(TestName, signature)`
- `#define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X(TestName, signature, ...)`
- `#define INTERNAL_CATCH_NTTP_0`

- #define INTERNAL_CATCH_NTP_GEN(...) INTERNAL_CATCH_VA_ARGS_IMPL(__VA_ARGS__ ↵
INTERNAL_CATCH_NTP_1(__VA_ARGS__), INTERNAL_CATCH_NTP_1(__VA_ARGS__),
INTERNAL_CATCH_NTP_1(__VA_ARGS__), INTERNAL_CATCH_NTP_1(__VA_ARGS__), INTERNAL_CATCH_NTP_1(↵
__VA_ARGS__), INTERNAL_CATCH_NTP_1(__VA_ARGS__), INTERNAL_CATCH_NTP_1(__VA_ ↵
ARGS__), INTERNAL_CATCH_NTP_1(__VA_ARGS__), INTERNAL_CATCH_NTP_1(__VA_ARGS__ ↵
), INTERNAL_CATCH_NTP_1(__VA_ARGS__), INTERNAL_CATCH_NTP_0)
- #define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, ...) INTERNAL_CATCH_VA_ARGS_IMPL(↵
"dummy", __VA_ARGS__, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_ ↵
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_C ↵
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0)(Test ↵
Name, __VA_ARGS__)
- #define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD(TestName, ClassName, ...) INTERNAL_CATCH_VA_ARGS_I ↵
"dummy", __VA_ARGS__, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TE ↵
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 ↵
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0)(Test ↵
Name, ClassName, __VA_ARGS__)
- #define INTERNAL_CATCH_NTP_REG_METHOD_GEN(TestName, ...) INTERNAL_CATCH_VA_ARGS_IMPL(↵
"dummy", __VA_ARGS__, INTERNAL_CATCH_NTP_REGISTER_METHOD, INTERNAL_CATCH_NTP_REGISTER_METH ↵
INTERNAL_CATCH_NTP_REGISTER_METHOD, INTERNAL_CATCH_NTP_REGISTER_METHOD,
INTERNAL_CATCH_NTP_REGISTER_METHOD, INTERNAL_CATCH_NTP_REGISTER_METHOD,
INTERNAL_CATCH_NTP_REGISTER_METHOD, INTERNAL_CATCH_NTP_REGISTER_METHOD,
INTERNAL_CATCH_NTP_REGISTER_METHOD, INTERNAL_CATCH_NTP_REGISTER_METHOD0,
INTERNAL_CATCH_NTP_REGISTER_METHOD0)(TestName, __VA_ARGS__)
- #define INTERNAL_CATCH_NTP_REG_GEN(TestFunc, ...) INTERNAL_CATCH_VA_ARGS_IMPL(↵
"dummy", __VA_ARGS__, INTERNAL_CATCH_NTP_REGISTER, INTERNAL_CATCH_NTP_REGISTER,
INTERNAL_CATCH_NTP_REGISTER, INTERNAL_CATCH_NTP_REGISTER, INTERNAL_CATCH_NTP_REGISTER,
INTERNAL_CATCH_NTP_REGISTER, INTERNAL_CATCH_NTP_REGISTER, INTERNAL_CATCH_NTP_REGISTER,
INTERNAL_CATCH_NTP_REGISTER, INTERNAL_CATCH_NTP_REGISTER0, INTERNAL_CATCH_NTP_REGISTER0)(↵
Func, __VA_ARGS__)
- #define INTERNAL_CATCH_DEFINE_SIG_TEST(TestName, ...) INTERNAL_CATCH_VA_ARGS_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 ↵
INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG ↵
INTERNAL_CATCH_DEFINE_SIG_TEST0)(TestName, __VA_ARGS__)
- #define INTERNAL_CATCH_DECLARE_SIG_TEST(TestName, ...) INTERNAL_CATCH_VA_ARGS_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_DECLAR ↵
INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLAR ↵
INTERNAL_CATCH_DECLARE_SIG_TEST1, INTERNAL_CATCH_DECLARE_SIG_TEST0)(TestName, ↵
__VA_ARGS__)
- #define INTERNAL_CATCH_REMOVE_PARENS_GEN(...) INTERNAL_CATCH_VA_ARGS_IMPL(__VA_ ↵
ARGS__, INTERNAL_CATCH_REMOVE_PARENS_11_ARG, INTERNAL_CATCH_REMOVE_PARENS_10_ARG, INTERNAL ↵
__VA_ARGS__)
- #define INTERNAL_CATCH_TESTCASE2(TestName, ...)
- #define INTERNAL_CATCH_TESTCASE(...) INTERNAL_CATCH_TESTCASE2(INTERNAL_CATCH_UNIQUE_NAME(↵
C_A_T_C_H_T_E_S_T_), __VA_ARGS__)
- #define INTERNAL_CATCH_METHOD_AS_TEST_CASE(QualifiedMethod, ...)
- #define INTERNAL_CATCH_TEST_CASE_METHOD2(TestName, ClassName, ...)
- #define INTERNAL_CATCH_TEST_CASE_METHOD(ClassName, ...) INTERNAL_CATCH_TEST_CASE_METHOD2(↵
INTERNAL_CATCH_UNIQUE_NAME(C_A_T_C_H_T_E_S_T_), ClassName, __VA_ARGS__)
- #define INTERNAL_CATCH_REGISTER_TESTCASE(Function, ...)
- #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_2(TestName, TestFunc, Name, Tags, Signature, ...)

- `#define INTERNAL_CATCH_TEMPLATE_TEST_CASE(Name, Tags, ...) INTERNAL_CATCH_TEMPLATE_TEST_CASE_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, typename TestType, __VA_ARGS__)`
- `#define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG(Name, Tags, Signature, ...) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_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__)`
- `#define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(testName, TestFuncName, Name, Tags, Signature, TmplTypes, TypesList)`
- `#define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE(Name, Tags, ...) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(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, typename T, __VA_ARGS__)`
- `#define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(Name, Tags, Signature, ...) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG_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__)`
- `#define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_2(testName, TestFunc, Name, Tags, TmplList)`
- `#define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE(Name, Tags, TmplList) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_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, TmplList)`
- `#define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2(testNameClass, testName, className, Name, Tags, Signature, ...)`
- `#define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD(className, Name, Tags, ...) 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_F_U_N_C_), className, Name, Tags, typename T, __VA_ARGS__)`
- `#define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG(className, Name, Tags, Signature, ...) 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_F_U_N_C_), className, Name, Tags, Signature, __VA_ARGS__)`
- `#define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2(testNameClass, TestFuncName, className, Name, Tags, Signature, TmplTypes, TypesList)`
- `#define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD(className, Name, Tags, ...) INTERNAL_CATCH_TEMPLATE_PRODUCT_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_), 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, Name, Tags, typename T, __VA_ARGS__)`
- `#define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG(className, Name, Tags, Signature, ...) INTERNAL_CATCH_TEMPLATE_PRODUCT_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_), 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, Name, Tags, Signature, __VA_ARGS__)`
- `#define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_2(testNameClass, testName, className, Name, Tags, TmplList)`
- `#define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD(className, Name, Tags, TmplList) INTERNAL_CATCH_TEMPLATE_LIST_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_), 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, Name, Tags, TmplList)`
- `#define INTERNAL_CATCH_REGISTER_ENUM(enumName, ...)`
- `#define CATCH_REGISTER_ENUM(enumName, ...) INTERNAL_CATCH_REGISTER_ENUM(enumName, __VA_ARGS__)`
- `#define CATCH_INTERNAL_STRINGIFY(...) # __VA_ARGS__`
- `#define INTERNAL_CATCH_TRY`
- `#define INTERNAL_CATCH_CATCH(capturer)`
- `#define INTERNAL_CATCH_REACT(handler) handler.complete();`
- `#define INTERNAL_CATCH_TEST(macroName, resultDisposition, ...)`
- `#define INTERNAL_CATCH_IF(macroName, resultDisposition, ...)`
- `#define INTERNAL_CATCH_ELSE(macroName, resultDisposition, ...)`
- `#define INTERNAL_CATCH_NO_THROW(macroName, resultDisposition, ...)`

- #define `INTERNAL_CATCH_THROWS`(macroName, resultDisposition, ...)
- #define `INTERNAL_CATCH_THROWS_AS`(macroName, exceptionType, resultDisposition, expr)
- #define `INTERNAL_CATCH_MSG`(macroName, messageType, resultDisposition, ...)
- #define `INTERNAL_CATCH_CAPTURE`(varName, macroName, ...)
- #define `INTERNAL_CATCH_INFO`(macroName, log) `Catch::ScopedMessage INTERNAL_CATCH_UNIQUE_NAME`(scopedMessage)(`Catch::MessageBuilder`(macroName##_catch_sr, `CATCH_INTERNAL_LINEINFO`, `Catch::ResultWas::Info`) << log);
- #define `INTERNAL_CATCH_UNSCOPED_INFO`(macroName, log) `Catch::getResultCapture`()<←
UnscopedMessage(`Catch::MessageBuilder`(macroName##_catch_sr, `CATCH_INTERNAL_LINEINFO`, `Catch::ResultWas::Info`) << log)
- #define `INTERNAL_CATCH_THROWS_STR_MATCHES`(macroName, resultDisposition, matcher, ...)
- #define `INTERNAL_CATCH_SECTION`(...)
- #define `INTERNAL_CATCH_DYNAMIC_SECTION`(...)
- #define `INTERNAL_CATCH_TRANSLATE_EXCEPTION2`(translatorName, signature)
- #define `INTERNAL_CATCH_TRANSLATE_EXCEPTION`(signature) `INTERNAL_CATCH_TRANSLATE_EXCEPTION2`(`INTERNAL_CATCH_UNIQUE_NAME`(catch_internal_ExceptionTranslator), signature)
- #define `INTERNAL_CHECK_THAT`(macroName, matcher, resultDisposition, arg)
- #define `INTERNAL_CATCH_THROWS_MATCHES`(macroName, exceptionType, resultDisposition, matcher, ...)
- #define `CATCH_MAKE_MSG`(...) (`Catch::ReusableStringStream`() << __VA_ARGS__).str()
- #define `CATCH_INTERNAL_ERROR`(...) `Catch::throw_logic_error`(`CATCH_MAKE_MSG`(`CATCH_INTERNAL_LINEINFO` << ": Internal Catch2 error: " << __VA_ARGS__))
- #define `CATCH_ERROR`(...) `Catch::throw_domain_error`(`CATCH_MAKE_MSG`(__VA_ARGS__))
- #define `CATCH_RUNTIME_ERROR`(...) `Catch::throw_runtime_error`(`CATCH_MAKE_MSG`(__VA_ARGS__ <←
__))
- #define `CATCH_ENFORCE`(condition, ...) do{ if(!(condition)) `CATCH_ERROR`(__VA_ARGS__); } while(false)
- #define `GENERATE`(...)
- #define `GENERATE_COPY`(...)
- #define `GENERATE_REF`(...)
- #define `REQUIRE`(...) `INTERNAL_CATCH_TEST`("REQUIRE", `Catch::ResultDisposition::Normal`, __VA_<←
ARGS__)
- #define `REQUIRE_FALSE`(...) `INTERNAL_CATCH_TEST`("REQUIRE_FALSE", `Catch::ResultDisposition::Normal` | `Catch::ResultDisposition::FalseTest`, __VA_ARGS__)
- #define `REQUIRE_THROWS`(...) `INTERNAL_CATCH_THROWS`("REQUIRE_THROWS", `Catch::ResultDisposition::Normal`, __VA_ARGS__)
- #define `REQUIRE_THROWS_AS`(expr, exceptionType) `INTERNAL_CATCH_THROWS_AS`("REQUIRE_<←
THROWS_AS", exceptionType, `Catch::ResultDisposition::Normal`, expr)
- #define `REQUIRE_THROWS_WITH`(expr, matcher) `INTERNAL_CATCH_THROWS_STR_MATCHES`("REQUIRE_THROWS_WITH", `Catch::ResultDisposition::Normal`, matcher, expr)
- #define `REQUIRE_THROWS_MATCHES`(expr, exceptionType, matcher) `INTERNAL_CATCH_THROWS_MATCHES`("REQUIRE_THROWS_MATCHES", exceptionType, `Catch::ResultDisposition::Normal`, matcher, expr)
- #define `REQUIRE_NOTHROW`(...) `INTERNAL_CATCH_NO_THROW`("REQUIRE_NOTHROW", `Catch::ResultDisposition::Nor`
__VA_ARGS__)
- #define `CHECK`(...) `INTERNAL_CATCH_TEST`("CHECK", `Catch::ResultDisposition::ContinueOnFailure`, <←
__VA_ARGS__)
- #define `CHECK_FALSE`(...) `INTERNAL_CATCH_TEST`("CHECK_FALSE", `Catch::ResultDisposition::ContinueOnFailure` | `Catch::ResultDisposition::FalseTest`, __VA_ARGS__)
- #define `CHECKED_IF`(...) `INTERNAL_CATCH_IF`("CHECKED_IF", `Catch::ResultDisposition::ContinueOnFailure`, __VA_ARGS__)
- #define `CHECKED_ELSE`(...) `INTERNAL_CATCH_ELSE`("CHECKED_ELSE", `Catch::ResultDisposition::ContinueOnFailure`, __VA_ARGS__)
- #define `CHECK_NOFAIL`(...) `INTERNAL_CATCH_TEST`("CHECK_NOFAIL", `Catch::ResultDisposition::ContinueOnFailure` | `Catch::ResultDisposition::SuppressFail`, __VA_ARGS__)
- #define `CHECK_THROWS`(...) `INTERNAL_CATCH_THROWS`("CHECK_THROWS", `Catch::ResultDisposition::ContinueOnFail`
__VA_ARGS__)

- `#define CHECK_THROWS_AS(expr, exceptionType) INTERNAL_CATCH_THROWS_AS("CHECK_↵`
`THROWS_AS", exceptionType, Catch::ResultDisposition::ContinueOnFailure, expr)`
- `#define CHECK_THROWS_WITH(expr, matcher) INTERNAL_CATCH_THROWS_STR_MATCHES(`
`"CHECK_THROWS_WITH", Catch::ResultDisposition::ContinueOnFailure, matcher, expr)`
- `#define CHECK_THROWS_MATCHES(expr, exceptionType, matcher) INTERNAL_CATCH_THROWS_MATCHES(`
`"CHECK_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::ContinueOnFailure, matcher,`
`expr)`
- `#define CHECK_NOTHROW(...) INTERNAL_CATCH_NO_THROW("CHECK_NOTHROW", Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__)`
- `#define CHECK_THAT(arg, matcher) INTERNAL_CHECK_THAT("CHECK_THAT", matcher, Catch::ResultDisposition::ContinueOnFailure, arg)`
- `#define REQUIRE_THAT(arg, matcher) INTERNAL_CHECK_THAT("REQUIRE_THAT", matcher, Catch::ResultDisposition::Normal, arg)`
- `#define INFO(msg) INTERNAL_CATCH_INFO("INFO", msg)`
- `#define UNSCOPED_INFO(msg) INTERNAL_CATCH_UNSCOPED_INFO("UNSCOPED_INFO", msg)`
- `#define WARN(msg) INTERNAL_CATCH_MSG("WARN", Catch::ResultWas::Warning, Catch::ResultDisposition::ContinueOnFailure, msg)`
- `#define CAPTURE(...) INTERNAL_CATCH_CAPTURE(INTERNAL_CATCH_UNIQUE_NAME(capturer), "CAPTURE", __VA_ARGS__)`
- `#define TEST_CASE(...) INTERNAL_CATCH_TESTCASE(__VA_ARGS__)`
- `#define TEST_CASE_METHOD(className, ...) INTERNAL_CATCH_TEST_CASE_METHOD(className, __VA_ARGS__)`
- `#define METHOD_AS_TEST_CASE(method, ...) INTERNAL_CATCH_METHOD_AS_TEST_CASE(method, __VA_ARGS__)`
- `#define REGISTER_TEST_CASE(Function, ...) INTERNAL_CATCH_REGISTER_TESTCASE(Function, __VA_ARGS__)`
- `#define SECTION(...) INTERNAL_CATCH_SECTION(__VA_ARGS__)`
- `#define DYNAMIC_SECTION(...) INTERNAL_CATCH_DYNAMIC_SECTION(__VA_ARGS__)`
- `#define FAIL(...) INTERNAL_CATCH_MSG("FAIL", Catch::ResultWas::ExplicitFailure, Catch::ResultDisposition::Normal, __VA_ARGS__)`
- `#define FAIL_CHECK(...) INTERNAL_CATCH_MSG("FAIL_CHECK", Catch::ResultWas::ExplicitFailure, Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__)`
- `#define SUCCEED(...) INTERNAL_CATCH_MSG("SUCCEED", Catch::ResultWas::Ok, Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__)`
- `#define ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE()`
- `#define TEMPLATE_TEST_CASE(...) INTERNAL_CATCH_TEMPLATE_TEST_CASE(__VA_ARGS__)`
- `#define TEMPLATE_TEST_CASE_SIG(...) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG(__VA_↵`
`ARGS__)`
- `#define TEMPLATE_TEST_CASE_METHOD(className, ...) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD(className, __VA_ARGS__)`
- `#define TEMPLATE_TEST_CASE_METHOD_SIG(className, ...) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG(className, __VA_ARGS__)`
- `#define TEMPLATE_PRODUCT_TEST_CASE(...) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE(__VA_ARGS__)`
- `#define TEMPLATE_PRODUCT_TEST_CASE_SIG(...) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(__VA_ARGS__)`
- `#define TEMPLATE_PRODUCT_TEST_CASE_METHOD(className, ...) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD(className, __VA_ARGS__)`
- `#define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG(className, ...) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG(className, __VA_ARGS__)`
- `#define TEMPLATE_LIST_TEST_CASE(...) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE(__VA_↵`
`ARGS__)`
- `#define TEMPLATE_LIST_TEST_CASE_METHOD(className, ...) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD(className, __VA_ARGS__)`
- `#define STATIC_REQUIRE(...) static_assert(__VA_ARGS__, #__VA_ARGS__); SUCCEED(#__VA_↵`
`ARGS__)`

- `#define STATIC_REQUIRE_FALSE(...) static_assert(!(__VA_ARGS__), "!(" #__VA_ARGS__ ")"); SUCCEED("!(" #__VA_ARGS__ ")")`
- `#define CATCH_TRANSLATE_EXCEPTION(signature) INTERNAL_CATCH_TRANSLATE_EXCEPTION(signature)`
- `#define SCENARIO(...) TEST_CASE("Scenario: " #__VA_ARGS__)`
- `#define SCENARIO_METHOD(className, ...) INTERNAL_CATCH_TEST_CASE_METHOD(className, "Scenario: " #__VA_ARGS__)`
- `#define GIVEN(desc) INTERNAL_CATCH_DYNAMIC_SECTION(" Given: " << desc)`
- `#define AND_GIVEN(desc) INTERNAL_CATCH_DYNAMIC_SECTION("And given: " << desc)`
- `#define WHEN(desc) INTERNAL_CATCH_DYNAMIC_SECTION(" When: " << desc)`
- `#define AND_WHEN(desc) INTERNAL_CATCH_DYNAMIC_SECTION(" And when: " << desc)`
- `#define THEN(desc) INTERNAL_CATCH_DYNAMIC_SECTION(" Then: " << desc)`
- `#define AND_THEN(desc) INTERNAL_CATCH_DYNAMIC_SECTION(" And: " << desc)`

Typedefs

- `template<typename Func , typename... U>
using Catch::FunctionReturnType = typename std::remove_reference<typename std::remove_cv<typename std::result_of<Func(U...)>::type>::type>::type`
- `using Catch::IReporterFactoryPtr = std::shared_ptr<IReporterFactory>`
- `using Catch::exceptionTranslateFunction = std::string(*)()`
- `using Catch::ExceptionTranslators = std::vector<std::unique_ptr<IExceptionTranslator const>>`
- `using Catch::StringMatcher = Matchers::Impl::MatcherBase<std::string>`
- `using Catch::Generators::GeneratorBasePtr = std::unique_ptr<GeneratorUntypedBase>`
- `using Catch::IConfigPtr = std::shared_ptr<IConfig const>`

Enumerations

- `enum class Catch::Verbosity { Catch::Quiet = 0 , Catch::Normal , Catch::High }`

Functions

- `unsigned int Catch::rngSeed ()`
- `std::ostream & operator<< (std::ostream &, Catch_global_namespace_dummy)`
- `std::ostream & Catch::operator<< (std::ostream &os, SourceLineInfo const &info)`
- `template<typename T >
T const & Catch::operator+ (T const &value, StreamEndStop)`
- `bool Catch::isThrowSafe (TestCase const &testCase, IConfig const &config)`
- `bool Catch::matchTest (TestCase const &testCase, TestSpec const &testSpec, IConfig const &config)`
- `std::vector< TestCase > Catch::filterTests (std::vector< TestCase > const &testCases, TestSpec const &testSpec, IConfig const &config)`
- `std::vector< TestCase > const & Catch::getAllTestCasesSorted (IConfig const &config)`
- `auto Catch::operator+= (std::string &lhs, StringRef const &sr) -> std::string &`
- `auto Catch::operator<< (std::ostream &os, StringRef const &sr) -> std::ostream &`
- `constexpr auto Catch::operator""_sr (char const *rawChars, std::size_t size) noexcept -> StringRef`
- `constexpr auto operator""_catch_sr (char const *rawChars, std::size_t size) noexcept -> Catch::StringRef`
- `auto Catch::makeTestInvoker (void(*testAsFunction)()) noexcept -> ITestInvoker *`
- `template<typename C >
auto Catch::makeTestInvoker (void(C::*testAsMethod)()) noexcept -> ITestInvoker *`
- `bool Catch::isOk (ResultWas::OfType resultType)`
- `bool Catch::isJustInfo (int flags)`
- `ResultDisposition::Flags Catch::operator| (ResultDisposition::Flags lhs, ResultDisposition::Flags rhs)`
- `bool Catch::shouldContinueOnFailure (int flags)`

- [bool Catch::isFalseTest \(int flags\)](#)
- [bool Catch::shouldSuppressFailure \(int flags\)](#)
- [std::ostream & Catch::cout \(\)](#)
- [std::ostream & Catch::cerr \(\)](#)
- [std::ostream & Catch::clog \(\)](#)
- [auto Catch::makeStream \(StringRef const &filename\) -> IStream const *](#)
- [std::string Catch::Detail::rawMemoryToString \(const void *object, std::size_t size\)](#)
- [template<typename T >
std::string Catch::Detail::rawMemoryToString \(const T &object\)](#)
- [template<typename E >
std::string Catch::Detail::convertUnknownEnumToString \(E e\)](#)
- [template<typename T >
std::enable_if<!std::is_enum< T >::value &&!std::is_base_of< std::exception, T >::value, std::string >::type
Catch::Detail::convertUnstreamable \(T const &\)](#)
- [template<typename T >
std::enable_if<!std::is_enum< T >::value &&std::is_base_of< std::exception, T >::value, std::string >::type
Catch::Detail::convertUnstreamable \(T const &ex\)](#)
- [template<typename T >
std::enable_if< std::is_enum< T >::value, std::string >::type Catch::Detail::convertUnstreamable \(T const
&value\)](#)
- [template<typename T >
std::string Catch::Detail::stringify \(const T &e\)](#)
- [template<typename InputIterator , typename Sentinel = InputIterator>
std::string Catch::Detail::rangeToString \(InputIterator first, Sentinel last\)](#)
- [template<typename Range >
std::string Catch::rangeToString \(Range const &range\)](#)
- [template<typename Allocator >
std::string Catch::rangeToString \(std::vector< bool, Allocator > const &v\)](#)
- [void Catch::formatReconstructedExpression \(std::ostream &os, std::string const &lhs, StringRef op, std::string const &rhs\)](#)
- [template<typename LhsT , typename RhsT >
auto Catch::compareEqual \(LhsT const &lhs, RhsT const &rhs\) -> bool](#)
- [template<typename T >
auto Catch::compareEqual \(T *const &lhs, int rhs\) -> bool](#)
- [template<typename T >
auto Catch::compareEqual \(T *const &lhs, long rhs\) -> bool](#)
- [template<typename T >
auto Catch::compareEqual \(int lhs, T *const &rhs\) -> bool](#)
- [template<typename T >
auto Catch::compareEqual \(long lhs, T *const &rhs\) -> bool](#)
- [template<typename LhsT , typename RhsT >
auto Catch::compareNotEqual \(LhsT const &lhs, RhsT &&rhs\) -> bool](#)
- [template<typename T >
auto Catch::compareNotEqual \(T *const &lhs, int rhs\) -> bool](#)
- [template<typename T >
auto Catch::compareNotEqual \(T *const &lhs, long rhs\) -> bool](#)
- [template<typename T >
auto Catch::compareNotEqual \(int lhs, T *const &rhs\) -> bool](#)
- [template<typename T >
auto Catch::compareNotEqual \(long lhs, T *const &rhs\) -> bool](#)
- [void Catch::handleExpression \(ITransientExpression const &expr\)](#)
- [template<typename T >
void Catch::handleExpression \(ExprLhs< T > const &expr\)](#)
- [IResultCapture & Catch::getResultCapture \(\)](#)
- [void Catch::handleExceptionMatchExpr \(AssertionHandler &handler, std::string const &str, StringRef const
&matcherString\)](#)

- `auto Catch::getCurrentNanosecondsSinceEpoch () -> uint64_t`
- `auto Catch::getEstimatedClockResolution () -> uint64_t`
- `IRegistryHub const & Catch::getRegistryHub ()`
- `IMutableRegistryHub & Catch::getMutableRegistryHub ()`
- `void Catch::cleanUp ()`
- `std::string Catch::translateActiveException ()`
- `Detail::Approx Catch::literals::operator""_a (long double val)`
- `Detail::Approx Catch::literals::operator""_a (unsigned long long val)`
- `bool Catch::startsWith (std::string const &s, std::string const &prefix)`
- `bool Catch::startsWith (std::string const &s, char prefix)`
- `bool Catch::endsWith (std::string const &s, std::string const &suffix)`
- `bool Catch::endsWith (std::string const &s, char suffix)`
- `bool Catch::contains (std::string const &s, std::string const &infix)`
- `void Catch::toLowerInPlace (std::string &s)`
- `std::string Catch::toLower (std::string const &s)`
- `std::string Catch::trim (std::string const &str)`
Returns a new string without whitespace at the start/end.
- `StringRef Catch::trim (StringRef ref)`
Returns a substring of the original ref without whitespace. Beware lifetimes!
- `std::vector< StringRef > Catch::splitStringRef (StringRef str, char delimiter)`
- `bool Catch::replaceInPlace (std::string &str, std::string const &replaceThis, std::string const &withThis)`
- `Exception::ExceptionMessageMatcher Catch::Matchers::Message (std::string const &message)`
- `Floating::WithinUlpMatcher Catch::Matchers::WithinULP (double target, uint64_t maxUlpDiff)`
- `Floating::WithinUlpMatcher Catch::Matchers::WithinULP (float target, uint64_t maxUlpDiff)`
- `Floating::WithinAbsMatcher Catch::Matchers::WithinAbs (double target, double margin)`
- `Floating::WithinRelMatcher Catch::Matchers::WithinRel (double target, double eps)`
- `Floating::WithinRelMatcher Catch::Matchers::WithinRel (double target)`
- `Floating::WithinRelMatcher Catch::Matchers::WithinRel (float target, float eps)`
- `Floating::WithinRelMatcher Catch::Matchers::WithinRel (float target)`
- `std::string Catch::Matchers::Generic::Detail::finalizeDescription (const std::string &desc)`
- `template<typename T >`
`Generic::PredicateMatcher< T > Catch::Matchers::Predicate (std::function< bool(T const &)> const`
`&predicate, std::string const &description="")`
- `StdString::EqualsMatcher Catch::Matchers::Equals (std::string const &str, CaseSensitive::Choice caseSensitivity=CaseSensitivity::CaseSensitive)`
- `StdString::ContainsMatcher Catch::Matchers::Contains (std::string const &str, CaseSensitive::Choice caseSensitivity=CaseSensitive::Yes)`
- `StdString::EndsWithMatcher Catch::Matchers::EndsWith (std::string const &str, CaseSensitive::Choice caseSensitivity=CaseSensitive::Yes)`
- `StdString::StartsWithMatcher Catch::Matchers::StartsWith (std::string const &str, CaseSensitive::Choice caseSensitivity=CaseSensitive::Yes)`
- `StdString::RegexMatcher Catch::Matchers::Matches (std::string const ®ex, CaseSensitive::Choice caseSensitivity=CaseSensitive::Yes)`
- `template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>`
`Vector::ContainsMatcher< T, AllocComp, AllocMatch > Catch::Matchers::Contains (std::vector< T,`
`AllocComp > const &comparator)`
- `template<typename T, typename Alloc = std::allocator<T>>`
`Vector::ContainsElementMatcher< T, Alloc > Catch::Matchers::VectorContains (T const &comparator)`
- `template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>`
`Vector::EqualsMatcher< T, AllocComp, AllocMatch > Catch::Matchers::Equals (std::vector< T, AllocComp`
`> const &comparator)`
- `template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>`
`Vector::ApproxMatcher< T, AllocComp, AllocMatch > Catch::Matchers::Approx (std::vector< T, AllocComp`
`> const &comparator)`
- `template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>`
`Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch > Catch::Matchers::UnorderedEquals (std::vector< T, AllocComp`
`> const &target)`

- `void Catch::handleExceptionMatchExpr (AssertionHandler &handler, StringMatcher const &matcher, StringRef const &matcherString)`
- `template<typename ArgT, typename MatcherT >
auto Catch::makeMatchExpr (ArgT const &arg, MatcherT const &matcher, StringRef const &matcherString)
-> MatchExpr< ArgT, MatcherT >`
- `void Catch::throw_exception (std::exception const &e)`
- `void Catch::throw_logic_error (std::string const &msg)`
- `void Catch::throw_domain_error (std::string const &msg)`
- `void Catch::throw_runtime_error (std::string const &msg)`
- `template<typename T, typename... Args>
std::unique_ptr< T > Catch::Generators::pf::make_unique (Args &&... args)`
- `template<typename T >
GeneratorWrapper< T > Catch::Generators::value (T &&value)`
- `template<typename T >
GeneratorWrapper< T > Catch::Generators::values (std::initializer_list< T > values)`
- `template<typename... Ts>
GeneratorWrapper< std::tuple< Ts... > > Catch::Generators::table (std::initializer_list< std::tuple< typename std::decay< Ts >::type... > > tuples)`
- `template<typename T, typename... Gs>
auto Catch::Generators::makeGenerators (GeneratorWrapper< T > &&generator, Gs &&... moreGenerators)
-> Generators< T >`
- `template<typename T >
auto Catch::Generators::makeGenerators (GeneratorWrapper< T > &&generator) -> Generators< T >`
- `template<typename T, typename... Gs>
auto Catch::Generators::makeGenerators (T &&val, Gs &&... moreGenerators) -> Generators< T >`
- `template<typename T, typename U, typename... Gs>
auto Catch::Generators::makeGenerators (as< T >, U &&val, Gs &&... moreGenerators) -> Generators< T >`
- `auto Catch::Generators::acquireGeneratorTracker (StringRef generatorName, SourceLineInfo const &lineInfo) -> IGeneratorTracker &`
- `template<typename L >
auto Catch::Generators::generate (StringRef generatorName, SourceLineInfo const &lineInfo, L const &generatorExpression) -> decltype(std::declval< decltype(generatorExpression())>().get())`
- `template<typename T >
GeneratorWrapper< T > Catch::Generators::take (size_t target, GeneratorWrapper< T > &&generator)`
- `template<typename T, typename Predicate >
GeneratorWrapper< T > Catch::Generators::filter (Predicate &&pred, GeneratorWrapper< T > &&generator)`
- `template<typename T >
GeneratorWrapper< T > Catch::Generators::repeat (size_t repeats, GeneratorWrapper< T > &&generator)`
- `template<typename Func, typename U, typename T = FunctionReturnType<Func, U>>
GeneratorWrapper< T > Catch::Generators::map (Func &&function, GeneratorWrapper< U > &&generator)`
- `template<typename T >
GeneratorWrapper< std::vector< T > > Catch::Generators::chunk (size_t size, GeneratorWrapper< T > &&generator)`
- `IMutableContext & Catch::getCurrentMutableContext ()`
- `IContext & Catch::getCurrentContext ()`
- `void Catch::cleanUpContext ()`
- `SimplePcg32 & Catch::rng ()`
- `template<typename T >
std::enable_if< std::is_integral< T >::value &&!std::is_same< T, bool >::value, GeneratorWrapper< T > >::type Catch::Generators::random (T a, T b)`
- `template<typename T >
std::enable_if< std::is_floating_point< T >::value, GeneratorWrapper< T > >::type Catch::Generators::random (T a, T b)`
- `template<typename T >
GeneratorWrapper< T > Catch::Generators::range (T const &start, T const &end, T const &step)`

- `template<typename T>`
`GeneratorWrapper< T > Catch::Generators::range (T const &start, T const &end)`
- `template<typename InputIterator, typename InputSentinel, typename ResultType = typename std::iterator_traits<InputIterator>::value_type>`
`GeneratorWrapper< ResultType > Catch::Generators::from_range (InputIterator from, InputSentinel to)`
- `template<typename Container, typename ResultType = typename Container::value_type>`
`GeneratorWrapper< ResultType > Catch::Generators::from_range (Container const &cnt)`
- `TestCase Catch::makeTestCase (ITestInvoker *testCase, std::string const &className, NameAndTags const &nameAndTags, SourceLineInfo const &lineInfo)`

Variables

- `const std::string Catch::Detail::unprintableString`

7.1.1 Macro Definition Documentation

7.1.1.1 AND_GIVEN

```
#define AND_GIVEN(
    desc ) INTERNAL_CATCH_DYNAMIC_SECTION( "And given:  " << desc )
```

Definition at line 17764 of file `catch.h`.

7.1.1.2 AND_THEN

```
#define AND_THEN(
    desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " And:  " << desc )
```

Definition at line 17768 of file `catch.h`.

7.1.1.3 AND_WHEN

```
#define AND_WHEN(
    desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " And when:  " << desc )
```

Definition at line 17766 of file `catch.h`.

7.1.1.4 ANON_TEST_CASE

```
#define ANON_TEST_CASE( ) INTERNAL_CATCH_TESTCASE()
```

Definition at line 17721 of file `catch.h`.

7.1.1.5 CAPTURE

```
#define CAPTURE(
    ... ) INTERNAL_CATCH_CAPTURE( INTERNAL_CATCH_UNIQUE_NAME(capturer), "CAPTURE",
__VA_ARGS__ )
```

Definition at line 17710 of file `catch.h`.

7.1.1.6 CATCH_CATCH_ALL

```
#define CATCH_CATCH_ALL if ((false))
```

Definition at line 455 of file [catch.h](#).

7.1.1.7 CATCH_CATCH_ANON

```
#define CATCH_CATCH_ANON(  
    type ) if ((false))
```

Definition at line 456 of file [catch.h](#).

7.1.1.8 CATCH_CONFIG_COUNTER

```
#define CATCH_CONFIG_COUNTER
```

Definition at line 354 of file [catch.h](#).

7.1.1.9 CATCH_CONFIG_CPP11_TO_STRING

```
#define CATCH_CONFIG_CPP11_TO_STRING
```

Definition at line 369 of file [catch.h](#).

7.1.1.10 CATCH_CONFIG_DISABLE_EXCEPTIONS

```
#define CATCH_CONFIG_DISABLE_EXCEPTIONS
```

Definition at line 397 of file [catch.h](#).

7.1.1.11 CATCH_CONFIG_GLOBAL_NEXTAFTER

```
#define CATCH_CONFIG_GLOBAL_NEXTAFTER
```

Definition at line 413 of file [catch.h](#).

7.1.1.12 CATCH_CONFIG_POSIX_SIGNALS

```
#define CATCH_CONFIG_POSIX_SIGNALS
```

Definition at line 361 of file [catch.h](#).

7.1.1.13 CATCH_CONFIG_WCHAR

```
#define CATCH_CONFIG_WCHAR
```

Definition at line 365 of file [catch.h](#).

7.1.1.14 CATCH_DEFER

```
#define CATCH_DEFER(  
    id ) id CATCH_EMPTY()
```

Definition at line 713 of file [catch.h](#).

7.1.1.15 CATCH_EMPTY

```
#define CATCH_EMPTY( )
```

Definition at line 712 of file [catch.h](#).

7.1.1.16 CATCH_ENFORCE

```
#define CATCH_ENFORCE(  
    condition,  
    ... )    do{ if( !(condition) ) CATCH_ERROR( __VA_ARGS__ ); } while(false)
```

Definition at line 3906 of file [catch.h](#).

7.1.1.17 CATCH_ERROR

```
#define CATCH_ERROR(  
    ... )    Catch::throw_domain_error(CATCH_MAKE_MSG( __VA_ARGS__ ))
```

Definition at line 3900 of file [catch.h](#).

7.1.1.18 CATCH_INTERNAL_CONFIG_COUNTER

```
#define CATCH_INTERNAL_CONFIG_COUNTER
```

Definition at line 298 of file [catch.h](#).

7.1.1.19 CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER

```
#define CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER
```

Definition at line 313 of file [catch.h](#).

7.1.1.20 CATCH_INTERNAL_CONFIG_POSIX_SIGNALS

```
#define CATCH_INTERNAL_CONFIG_POSIX_SIGNALS
```

Definition at line 189 of file [catch.h](#).

7.1.1.21 CATCH_INTERNAL_ERROR

```
#define CATCH_INTERNAL_ERROR(  
    ... )    Catch::throw_logic_error(CATCH_MAKE_MSG( CATCH_INTERNAL_LINEINFO << "  
Internal Catch2 error:  " << __VA_ARGS__))
```

Definition at line 3897 of file [catch.h](#).

7.1.1.22 CATCH_INTERNAL_IGNORE_BUT_WARN

```
#define CATCH_INTERNAL_IGNORE_BUT_WARN(  
    ... )
```

Definition at line 440 of file [catch.h](#).

7.1.1.23 CATCH_INTERNAL_LINEINFO

```
#define CATCH_INTERNAL_LINEINFO    ::Catch::SourceLineInfo( __FILE__, static_cast<std::size_t>(   
__LINE__ ) )
```

Definition at line 543 of file [catch.h](#).

7.1.1.24 CATCH_INTERNAL_START_WARNINGS_SUPPRESSION

```
#define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION
```

Definition at line 419 of file [catch.h](#).

7.1.1.25 CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION

```
#define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
```

Definition at line 422 of file [catch.h](#).

7.1.1.26 CATCH_INTERNAL_STRINGIFY

```
#define CATCH_INTERNAL_STRINGIFY(  
    ... ) #__VA_ARGS__
```

Definition at line 2680 of file [catch.h](#).

7.1.1.27 CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS

```
#define CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS
```

Definition at line 428 of file [catch.h](#).

7.1.1.28 CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS

```
#define CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS
```

Definition at line 425 of file [catch.h](#).

7.1.1.29 CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS

```
#define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
```

Definition at line 450 of file [catch.h](#).

7.1.1.30 CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS

```
#define CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS
```

Definition at line 431 of file [catch.h](#).

7.1.1.31 CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS

```
#define CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS
```

Definition at line 434 of file [catch.h](#).

7.1.1.32 CATCH_MAKE_MSG

```
#define CATCH_MAKE_MSG(  
    ... )    (Catch::ReusableStringStream() << __VA_ARGS__).str()
```

Definition at line 3894 of file [catch.h](#).

7.1.1.33 CATCH_REC_END

```
#define CATCH_REC_END(  
    ... )
```

Definition at line 709 of file [catch.h](#).

7.1.1.34 CATCH_REC_GET_END

```
#define CATCH_REC_GET_END(
    ... ) CATCH_REC_GET_END1
```

Definition at line 717 of file [catch.h](#).

7.1.1.35 CATCH_REC_GET_END1

```
#define CATCH_REC_GET_END1(
    ... ) CATCH_REC_GET_END2
```

Definition at line 716 of file [catch.h](#).

7.1.1.36 CATCH_REC_GET_END2

```
#define CATCH_REC_GET_END2( ) 0, CATCH_REC_END
```

Definition at line 715 of file [catch.h](#).

7.1.1.37 CATCH_REC_LIST

```
#define CATCH_REC_LIST(
    f,
    ... ) CATCH_RECURSE(CATCH_REC_LIST2(f, __VA_ARGS__,
    ()()(), ()()(), ()()(), 0))
```

Definition at line 735 of file [catch.h](#).

7.1.1.38 CATCH_REC_LIST0

```
#define CATCH_REC_LIST0(
    f,
    x,
    peek,
    ... ) , f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1) ) ( f, peek, ↵
__VA_ARGS__ )
```

Definition at line 722 of file [catch.h](#).

7.1.1.39 CATCH_REC_LIST0_UD

```
#define CATCH_REC_LIST0_UD(
    f,
    userdata,
    x,
    peek,
    ... ) , f(userdata, x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1_UD) )
( f, userdata, peek, __VA_ARGS__ )
```

Definition at line 726 of file [catch.h](#).

7.1.1.40 CATCH_REC_LIST1

```
#define CATCH_REC_LIST1(
    f,
    x,
    peek,
    ... ) , f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST0) ) ( f, peek, ↵
__VA_ARGS__ )
```

Definition at line 723 of file [catch.h](#).

7.1.1.41 CATCH_REC_LIST1_UD

```
#define CATCH_REC_LIST1_UD(
    f,
    userdata,
    x,
    peek,
    ... ) , f(userdata, x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST0_UD) )
( f, userdata, peek, __VA_ARGS__ )
```

Definition at line 727 of file [catch.h](#).

7.1.1.42 CATCH_REC_LIST2

```
#define CATCH_REC_LIST2(
    f,
    x,
    peek,
    ... ) f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1) ) ( f, peek, ↵
VA_ARGS__ )
```

Definition at line 724 of file [catch.h](#).

7.1.1.43 CATCH_REC_LIST2_UD

```
#define CATCH_REC_LIST2_UD(
    f,
    userdata,
    x,
    peek,
    ... ) f(userdata, x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1_UD) ) (
f, userdata, peek, __VA_ARGS__ )
```

Definition at line 728 of file [catch.h](#).

7.1.1.44 CATCH_REC_LIST_UD

```
#define CATCH_REC_LIST_UD(
    f,
    userdata,
    ... ) CATCH_RECURSE(CATCH_REC_LIST2_UD(f, userdata, __VA_ARGS__, ()()(), ()()(),
()()(), 0))
```

Definition at line 733 of file [catch.h](#).

7.1.1.45 CATCH_REC_NEXT

```
#define CATCH_REC_NEXT(  
    test,  
    next ) CATCH_REC_NEXT1 (CATCH_REC_GET_END test, next)
```

Definition at line 720 of file [catch.h](#).

7.1.1.46 CATCH_REC_NEXT0

```
#define CATCH_REC_NEXT0(  
    test,  
    next,  
    ... ) next CATCH_REC_OUT
```

Definition at line 718 of file [catch.h](#).

7.1.1.47 CATCH_REC_NEXT1

```
#define CATCH_REC_NEXT1(  
    test,  
    next ) CATCH_DEFER ( CATCH_REC_NEXT0 ) ( test, next, 0)
```

Definition at line 719 of file [catch.h](#).

7.1.1.48 CATCH_REC_OUT

```
#define CATCH_REC_OUT
```

Definition at line 710 of file [catch.h](#).

7.1.1.49 CATCH_RECURSE

```
#define CATCH_RECURSE(  
    ... ) CATCH_RECURSION_LEVEL5 (__VA_ARGS__)
```

Definition at line 706 of file [catch.h](#).

7.1.1.50 CATCH_RECURSION_LEVEL0

```
#define CATCH_RECURSION_LEVEL0(  
    ... ) __VA_ARGS__
```

Definition at line 693 of file [catch.h](#).

7.1.1.51 CATCH_RECURSION_LEVEL1

```
#define CATCH_RECURSION_LEVEL1(  
    ... ) CATCH_RECURSION_LEVEL0 (CATCH_RECURSION_LEVEL0 (CATCH_RECURSION_LEVEL0 (___↵  
VA_ARGS__)))
```

Definition at line 694 of file [catch.h](#).

7.1.1.52 CATCH_RECURSION_LEVEL2

```
#define CATCH_RECURSION_LEVEL2(  
    ... ) CATCH_RECURSION_LEVEL1 (CATCH_RECURSION_LEVEL1 (CATCH_RECURSION_LEVEL1 (___↵  
VA_ARGS__)))
```

Definition at line 695 of file [catch.h](#).

7.1.1.53 CATCH_RECURSION_LEVEL3

```
#define CATCH_RECURSION_LEVEL3(  
    ... ) CATCH_RECURSION_LEVEL2 (CATCH_RECURSION_LEVEL2 (CATCH_RECURSION_LEVEL2 (___↵  
VA_ARGS__)))
```

Definition at line 696 of file [catch.h](#).

7.1.1.54 CATCH_RECURSION_LEVEL4

```
#define CATCH_RECURSION_LEVEL4(  
    ... ) CATCH_RECURSION_LEVEL3 (CATCH_RECURSION_LEVEL3 (CATCH_RECURSION_LEVEL3 (___↵  
VA_ARGS__)))
```

Definition at line 697 of file [catch.h](#).

7.1.1.55 CATCH_RECURSION_LEVEL5

```
#define CATCH_RECURSION_LEVEL5(  
    ... ) CATCH_RECURSION_LEVEL4 (CATCH_RECURSION_LEVEL4 (CATCH_RECURSION_LEVEL4 (___↵  
VA_ARGS__)))
```

Definition at line 698 of file [catch.h](#).

7.1.1.56 CATCH_REGISTER_ENUM

```
#define CATCH_REGISTER_ENUM(  
    enumName,  
    ... ) INTERNAL_CATCH_REGISTER_ENUM( enumName, __VA_ARGS__ )
```

Definition at line 2182 of file [catch.h](#).

7.1.1.57 CATCH_REGISTER_TAG_ALIAS

```
#define CATCH_REGISTER_TAG_ALIAS(
    alias,
    spec )
```

Value:

```
CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
namespace{ Catch::RegistrarForTagAliases INTERNAL_CATCH_UNIQUE_NAME( AutoRegisterTagAlias )( alias,
    spec, CATCH_INTERNAL_LINEINFO ); } \
CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
```

Definition at line 555 of file [catch.h](#).

7.1.1.58 CATCH_RUNTIME_ERROR

```
#define CATCH_RUNTIME_ERROR(
    ... ) Catch::throw_runtime_error(CATCH_MAKE_MSG( __VA_ARGS__ ))
```

Definition at line 3903 of file [catch.h](#).

7.1.1.59 CATCH_TRANSLATE_EXCEPTION

```
#define CATCH_TRANSLATE_EXCEPTION(
    signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION( signature )
```

Definition at line 17757 of file [catch.h](#).

7.1.1.60 CATCH_TRY

```
#define CATCH_TRY if ((true))
```

Definition at line 454 of file [catch.h](#).

7.1.1.61 CATCH_VERSION_MAJOR

```
#define CATCH_VERSION_MAJOR 2
```

Definition at line 16 of file [catch.h](#).

7.1.1.62 CATCH_VERSION_MINOR

```
#define CATCH_VERSION_MINOR 13
```

Definition at line 17 of file [catch.h](#).

7.1.1.63 CATCH_VERSION_PATCH

```
#define CATCH_VERSION_PATCH 10
```

Definition at line 18 of file [catch.h](#).

7.1.1.64 CHECK

```
#define CHECK(  
    ... ) INTERNAL_CATCH_TEST( "CHECK", Catch::ResultDisposition::ContinueOnFailure,  
    __VA_ARGS__ )
```

Definition at line 17687 of file [catch.h](#).

7.1.1.65 CHECK_FALSE

```
#define CHECK_FALSE(  
    ... ) INTERNAL_CATCH_TEST( "CHECK_FALSE", Catch::ResultDisposition::ContinueOnFailure  
    | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
```

Definition at line 17688 of file [catch.h](#).

7.1.1.66 CHECK_NOFAIL

```
#define CHECK_NOFAIL(  
    ... ) INTERNAL_CATCH_TEST( "CHECK_NOFAIL", Catch::ResultDisposition::ContinueOnFailure  
    | Catch::ResultDisposition::SuppressFail, __VA_ARGS__ )
```

Definition at line 17691 of file [catch.h](#).

7.1.1.67 CHECK_NOTHROW

```
#define CHECK_NOTHROW(  
    ... ) INTERNAL_CATCH_NO_THROW( "CHECK_NOTHROW", Catch::ResultDisposition::ContinueOnFailure,  
    __VA_ARGS__ )
```

Definition at line 17699 of file [catch.h](#).

7.1.1.68 CHECK_THAT

```
#define CHECK_THAT(  
    arg,  
    matcher ) INTERNAL_CHECK_THAT( "CHECK_THAT", matcher, Catch::ResultDisposition::ContinueOnFailure,  
    arg )
```

Definition at line 17702 of file [catch.h](#).

7.1.1.69 CHECK_THROWS

```
#define CHECK_THROWS(  
    ... ) INTERNAL_CATCH_THROWS( "CHECK_THROWS", Catch::ResultDisposition::ContinueOnFailure,  
    __VA_ARGS__ )
```

Definition at line 17693 of file [catch.h](#).

7.1.1.70 CHECK_THROWS_AS

```
#define CHECK_THROWS_AS(  
    expr,  
    exceptionType ) INTERNAL_CATCH_THROWS_AS( "CHECK_THROWS_AS", exceptionType,  
    Catch::ResultDisposition::ContinueOnFailure, expr )
```

Definition at line 17694 of file [catch.h](#).

7.1.1.71 CHECK_THROWS_MATCHES

```
#define CHECK_THROWS_MATCHES(  
    expr,  
    exceptionType,  
    matcher ) INTERNAL_CATCH_THROWS_MATCHES( "CHECK_THROWS_MATCHES", exceptionType,  
    Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
```

Definition at line 17697 of file [catch.h](#).

7.1.1.72 CHECK_THROWS_WITH

```
#define CHECK_THROWS_WITH(  
    expr,  
    matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "CHECK_THROWS_WITH", Catch::ResultDisposition::ContinueOnFailure,  
    matcher, expr )
```

Definition at line 17695 of file [catch.h](#).

7.1.1.73 CHECKED_ELSE

```
#define CHECKED_ELSE(  
    ... ) INTERNAL_CATCH_ELSE( "CHECKED_ELSE", Catch::ResultDisposition::ContinueOnFailure,  
    __VA_ARGS__ )
```

Definition at line 17690 of file [catch.h](#).

7.1.1.74 CHECKED_IF

```
#define CHECKED_IF(  
    ... ) INTERNAL_CATCH_IF( "CHECKED_IF", Catch::ResultDisposition::ContinueOnFailure,  
    __VA_ARGS__ )
```

Definition at line 17689 of file [catch.h](#).

7.1.1.75 DYNAMIC_SECTION

```
#define DYNAMIC_SECTION(
    ... ) INTERNAL_CATCH_DYNAMIC_SECTION( __VA_ARGS__ )
```

Definition at line 17717 of file [catch.h](#).

7.1.1.76 FAIL

```
#define FAIL(
    ... ) INTERNAL_CATCH_MSG( "FAIL", Catch::ResultWas::ExplicitFailure, Catch::ResultDisposition::
__VA_ARGS__ )
```

Definition at line 17718 of file [catch.h](#).

7.1.1.77 FAIL_CHECK

```
#define FAIL_CHECK(
    ... ) INTERNAL_CATCH_MSG( "FAIL_CHECK", Catch::ResultWas::ExplicitFailure,
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
```

Definition at line 17719 of file [catch.h](#).

7.1.1.78 GENERATE

```
#define GENERATE(
    ... )
```

Value:

```
Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
    CATCH_INTERNAL_LINEINFO, \
    [=]{ using namespace Catch::Generators; return makeGenerators( __VA_ARGS__
); } )
```

Definition at line 4100 of file [catch.h](#).

7.1.1.79 GENERATE_COPY

```
#define GENERATE_COPY(
    ... )
```

Value:

```
Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
    CATCH_INTERNAL_LINEINFO, \
    [=]{ using namespace Catch::Generators; return makeGenerators( __VA_ARGS__
); } )
```

Definition at line 4104 of file [catch.h](#).

7.1.1.80 GENERATE_REF

```
#define GENERATE_REF(
    ... )
```

Value:

```
Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
    CATCH_INTERNAL_LINEINFO, \
    [&]{ using namespace Catch::Generators; return makeGenerators( __VA_ARGS__ ); } )
```

Definition at line 4108 of file [catch.h](#).

7.1.1.81 GIVEN

```
#define GIVEN(
    desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " Given: " << desc )
```

Definition at line 17763 of file [catch.h](#).

7.1.1.82 INFO

```
#define INFO(
    msg ) INTERNAL_CATCH_INFO( "INFO", msg )
```

Definition at line 17707 of file [catch.h](#).

7.1.1.83 INTERNAL_CATCH_CAPTURE

```
#define INTERNAL_CATCH_CAPTURE(
    varName,
    macroName,
    ... )
```

Value:

```
auto varName = Catch::Capturer( macroName, CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info, #__VA_ARGS__ ); \
varName.captureValues( 0, __VA_ARGS__ )
```

Definition at line 2786 of file [catch.h](#).

7.1.1.84 INTERNAL_CATCH_CATCH

```
#define INTERNAL_CATCH_CATCH(
    capturer )
```

Definition at line 2691 of file [catch.h](#).

7.1.1.85 INTERNAL_CATCH_DECLARE_SIG_TEST

```
#define INTERNAL_CATCH_DECLARE_SIG_TEST(
    TestName,
    ... ) INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__, INTERNAL_CATCH_DECLARE_SIG_TEST_X, INT
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_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL
INTERNAL_CATCH_DECLARE_SIG_TEST1, INTERNAL_CATCH_DECLARE_SIG_TEST0) (TestName, __VA_ARGS__)
```

Definition at line 903 of file [catch.h](#).

7.1.1.86 INTERNAL_CATCH_DECLARE_SIG_TEST0

```
#define INTERNAL_CATCH_DECLARE_SIG_TEST0(
    TestName )
```

Definition at line 830 of file [catch.h](#).

7.1.1.87 INTERNAL_CATCH_DECLARE_SIG_TEST1

```
#define INTERNAL_CATCH_DECLARE_SIG_TEST1(
    TestName,
    signature )
```

Value:

```
template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
static void TestName()
```

Definition at line 831 of file [catch.h](#).

7.1.1.88 INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD

```
#define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD(
    TestName,
    ClassName,
    ... ) INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__, INTERNAL_CATCH_DECLARE_SIG_TEST_METHO
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0) (TestName,
ClassName, __VA_ARGS__)
```

Definition at line 899 of file [catch.h](#).

7.1.1.89 INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0

```
#define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0(
    TestName,
    ClassName )
```

Definition at line 874 of file [catch.h](#).

7.1.1.90 INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1

```
#define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1(
    TestName,
    ClassName,
    signature )
```

Value:

```
template<typename TestType> \
struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName)<TestType> { \
    void test();\
}
```

Definition at line 875 of file [catch.h](#).

7.1.1.91 INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X

```
#define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X(
    TestName,
    ClassName,
    signature,
    ... )
```

Value:

```
template<INTERNAL_CATCH_REMOVE_PARENS(signature)> \
struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName)<__VA_ARGS__> { \
    void test();\
}
```

Definition at line 881 of file [catch.h](#).

7.1.1.92 INTERNAL_CATCH_DECLARE_SIG_TEST_X

```
#define INTERNAL_CATCH_DECLARE_SIG_TEST_X(
    TestName,
    signature,
    ... )
```

Value:

```
template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
static void TestName()
```

Definition at line 834 of file [catch.h](#).

7.1.1.93 INTERNAL_CATCH_DEF

```
#define INTERNAL_CATCH_DEF(
    ... ) INTERNAL_CATCH_DEF __VA_ARGS__
```

Definition at line 739 of file [catch.h](#).

7.1.1.94 INTERNAL_CATCH_DEFINE_SIG_TEST

```
#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_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST1,
INTERNAL_CATCH_DEFINE_SIG_TEST0) (TestName, __VA_ARGS__)
```

Definition at line 902 of file [catch.h](#).

7.1.1.95 INTERNAL_CATCH_DEFINE_SIG_TEST0

```
#define INTERNAL_CATCH_DEFINE_SIG_TEST0(
    TestName )
```

Definition at line 838 of file [catch.h](#).

7.1.1.96 INTERNAL_CATCH_DEFINE_SIG_TEST1

```
#define INTERNAL_CATCH_DEFINE_SIG_TEST1(
    TestName,
    signature )
```

Value:

```
template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
static void TestName()
```

Definition at line 839 of file [catch.h](#).

7.1.1.97 INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD

```
#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_METHOD1,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0) (TestName, __VA_ARGS__)
```

Definition at line 898 of file [catch.h](#).

7.1.1.98 INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0

```
#define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0(
    TestName )
```

Definition at line 887 of file [catch.h](#).

7.1.1.99 INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1

```
#define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1(
    TestName,
    signature )
```

Value:

```
template<typename TestType> \
void INTERNAL_CATCH_MAKE_NAMESPACE( TestName )::TestName<TestType>::test()
```

Definition at line 888 of file [catch.h](#).

7.1.1.100 INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X

```
#define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X(
    TestName,
    signature,
    ... )
```

Value:

```
template<INTERNAL_CATCH_REMOVE_PARENS( signature )> \
void INTERNAL_CATCH_MAKE_NAMESPACE( TestName )::TestName<__VA_ARGS__>::test()
```

Definition at line 891 of file [catch.h](#).

7.1.1.101 INTERNAL_CATCH_DEFINE_SIG_TEST_X

```
#define INTERNAL_CATCH_DEFINE_SIG_TEST_X(
    TestName,
    signature,
    ... )
```

Value:

```
template<INTERNAL_CATCH_REMOVE_PARENS( signature )>\
static void TestName()
```

Definition at line 842 of file [catch.h](#).

7.1.1.102 INTERNAL_CATCH_DYNAMIC_SECTION

```
#define INTERNAL_CATCH_DYNAMIC_SECTION(
    ... )
```

Value:

```
CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
if( Catch::Section const& INTERNAL_CATCH_UNIQUE_NAME( catch_internal_section ) = Catch::SectionInfo(
    CATCH_INTERNAL_LINEINFO, (Catch::ReusableStringStream() << __VA_ARGS__).str() ) ) \
CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
```

Definition at line 2936 of file [catch.h](#).

7.1.1.103 INTERNAL_CATCH_ELSE

```
#define INTERNAL_CATCH_ELSE(
    macroName,
    resultDisposition,
    ... )
```

Value:

```
INTERNAL_CATCH_TEST( macroName, resultDisposition, __VA_ARGS__ ); \
if( !Catch::getResultCapture().lastAssertionPassed() )
```

Definition at line 2722 of file [catch.h](#).

7.1.1.104 INTERNAL_CATCH_EXPAND1

```
#define INTERNAL_CATCH_EXPAND1(
    param ) INTERNAL_CATCH_EXPAND2( param)
```

Definition at line 737 of file [catch.h](#).

7.1.1.105 INTERNAL_CATCH_EXPAND2

```
#define INTERNAL_CATCH_EXPAND2(
    ... ) INTERNAL_CATCH_NO## __VA_ARGS__
```

Definition at line 738 of file [catch.h](#).

7.1.1.106 INTERNAL_CATCH_IF

```
#define INTERNAL_CATCH_IF(
    macroName,
    resultDisposition,
    ... )
```

Value:

```
INTERNAL_CATCH_TEST( macroName, resultDisposition, __VA_ARGS__ ); \
if( Catch::getResultCapture().lastAssertionPassed() )
```

Definition at line 2717 of file [catch.h](#).

7.1.1.107 INTERNAL_CATCH_INFO

```
#define INTERNAL_CATCH_INFO(
    macroName,
    log ) Catch::ScopedMessage INTERNAL_CATCH_UNIQUE_NAME( scopedMessage )( Catch::MessageBuilder(
macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info ) << log );
```

Definition at line 2791 of file [catch.h](#).

7.1.1.108 INTERNAL_CATCH_MAKE_NAMESPACE

```
#define INTERNAL_CATCH_MAKE_NAMESPACE(
    name ) INTERNAL_CATCH_MAKE_NAMESPACE2(name)
```

Definition at line 753 of file [catch.h](#).

7.1.1.109 INTERNAL_CATCH_MAKE_NAMESPACE2

```
#define INTERNAL_CATCH_MAKE_NAMESPACE2(
    ... ) ns_##__VA_ARGS__
```

Definition at line 752 of file [catch.h](#).

7.1.1.110 INTERNAL_CATCH_MAKE_TYPE_LIST

```
#define INTERNAL_CATCH_MAKE_TYPE_LIST(
    ... ) INTERNAL_CATCH_MAKE_TYPE_LIST2(INTERNAL_CATCH_REMOVE_PARENS(__VA_ARGS__))
```

Definition at line 759 of file [catch.h](#).

7.1.1.111 INTERNAL_CATCH_MAKE_TYPE_LIST2

```
#define INTERNAL_CATCH_MAKE_TYPE_LIST2(
    ... ) decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS_GEN(__VA_ARGS__)>())
```

Definition at line 758 of file [catch.h](#).

7.1.1.112 INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES

```
#define INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(
    ... ) CATCH_REC_LIST(INTERNAL_CATCH_MAKE_TYPE_LIST, __VA_ARGS__)
```

Definition at line 765 of file [catch.h](#).

7.1.1.113 INTERNAL_CATCH_METHOD_AS_TEST_CASE

```
#define INTERNAL_CATCH_METHOD_AS_TEST_CASE(
    QualifiedMethod,
    ... )
```

Value:

```
CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
namespace{ Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker(
&QualifiedMethod ), CATCH_INTERNAL_LINEINFO, "&" #QualifiedMethod, Catch::NameAndTags{ __VA_ARGS__ }
); } /* NOLINT */ \
CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
```

Definition at line 1058 of file [catch.h](#).

7.1.1.114 INTERNAL_CATCH_MSG

```
#define INTERNAL_CATCH_MSG(
    macroName,
    messageType,
    resultDisposition,
    ... )
```

Value:

```
do { \
    Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
    Catch::StringRef(), resultDisposition ); \
    catchAssertionHandler.handleMessage( messageType, ( Catch::MessageStream() « __VA_ARGS__ +
    ::Catch::StreamEndStop() ).m_stream.str() ); \
    INTERNAL_CATCH_REACT( catchAssertionHandler ) \
} while( false )
```

Definition at line 2778 of file [catch.h](#).

7.1.1.115 INTERNAL_CATCH_NO_THROW

```
#define INTERNAL_CATCH_NO_THROW(
    macroName,
    resultDisposition,
    ... )
```

Value:

```
do { \
    Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
    CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition ); \
    try { \
        static_cast<void>(__VA_ARGS__); \
        catchAssertionHandler.handleExceptionNotThrownAsExpected(); \
    } \
    catch( ... ) { \
        catchAssertionHandler.handleUnexpectedInflightException(); \
    } \
    INTERNAL_CATCH_REACT( catchAssertionHandler ) \
} while( false )
```

Definition at line 2727 of file [catch.h](#).

7.1.1.116 INTERNAL_CATCH_NOINTERNAL_CATCH_DEF

```
#define INTERNAL_CATCH_NOINTERNAL_CATCH_DEF
```

Definition at line 740 of file [catch.h](#).

7.1.1.117 INTERNAL_CATCH_NTTP_0

```
#define INTERNAL_CATCH_NTTP_0
```

Definition at line 896 of file [catch.h](#).

7.1.1.118 INTERNAL_CATCH_NTTP_1

```
#define INTERNAL_CATCH_NTTP_1(
    signature,
    ... )
```

Value:

```
template<INTERNAL_CATCH_REMOVE_PARENS(signature)> struct Nttp{};\
template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
constexpr auto get_wrapper() noexcept -> Nttp<__VA_ARGS__> { return {}; } \
template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class...> struct NttpTemplateTypeList{};\
template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class...Cs>\
constexpr auto get_wrapper() noexcept -> NttpTemplateTypeList<Cs...> { return {}; } \
\
template< template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class Container,
    template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class List,
    INTERNAL_CATCH_REMOVE_PARENS(signature)>\
struct rewrap<NttpTemplateTypeList<Container>, List<__VA_ARGS__> { using type =
    TypeList<Container<__VA_ARGS__>; };}\
template< template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class Container,
    template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class List, INTERNAL_CATCH_REMOVE_PARENS(signature),
    typename...Elements>\
struct rewrap<NttpTemplateTypeList<Container>, List<__VA_ARGS__>, Elements...> { using type = typename
    append<TypeList<Container<__VA_ARGS__>, typename rewrap<NttpTemplateTypeList<Container>,
    Elements...>::type>::type; };}\
template<template <typename...> class Final, template<INTERNAL_CATCH_REMOVE_PARENS(signature)>
    class...Containers, typename...Types>\
struct create<Final, NttpTemplateTypeList<Containers...>, TypeList<Types...> { using type = typename
    append<Final<>, typename rewrap<NttpTemplateTypeList<Containers>, Types...>::type...>::type; };
```

Definition at line 815 of file [catch.h](#).

7.1.1.119 INTERNAL_CATCH_NTTP_GEN

```
#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)
```

Definition at line 897 of file [catch.h](#).

7.1.1.120 INTERNAL_CATCH_NTTP_REG_GEN

```
#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_REGISTER0,
INTERNAL_CATCH_NTTP_REGISTER0)(TestFunc, __VA_ARGS__)
```

Definition at line 901 of file [catch.h](#).

7.1.1.121 INTERNAL_CATCH_NTTP_REG_METHOD_GEN

```
#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_METHOD0,
INTERNAL_CATCH_NTTP_REGISTER_METHOD0)(TestName, __VA_ARGS__)
```

Definition at line 900 of file [catch.h](#).

7.1.1.122 INTERNAL_CATCH_NTTP_REGISTER

```
#define INTERNAL_CATCH_NTTP_REGISTER(
    TestFunc,
    signature,
    ... )
```

Value:

```
template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
void reg_test(Nttp<__VA_ARGS__>, Catch::NameAndTags nameAndTags)\
{\
    Catch::AutoReg( Catch::makeTestInvoker(&TestFunc<__VA_ARGS__>), CATCH_INTERNAL_LINEINFO,
    Catch::StringRef(), nameAndTags);\
}
```

Definition at line 853 of file [catch.h](#).

7.1.1.123 INTERNAL_CATCH_NTTP_REGISTER0

```
#define INTERNAL_CATCH_NTTP_REGISTER0(
    TestFunc,
    signature )
```

Value:

```
template<typename Type>\
void reg_test(TypeList<Type>, Catch::NameAndTags nameAndTags)\
{\
    Catch::AutoReg( Catch::makeTestInvoker(&TestFunc<Type>), CATCH_INTERNAL_LINEINFO,
    Catch::StringRef(), nameAndTags);\
}
```

Definition at line 846 of file [catch.h](#).

7.1.1.124 INTERNAL_CATCH_NTTP_REGISTER_METHOD

```
#define INTERNAL_CATCH_NTTP_REGISTER_METHOD(
    TestName,
    signature,
    ... )
```

Value:

```
template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
void reg_test(Nttp<__VA_ARGS__>, Catch::StringRef className, Catch::NameAndTags nameAndTags)\
{\
    Catch::AutoReg( Catch::makeTestInvoker(&TestName<__VA_ARGS__>::test), CATCH_INTERNAL_LINEINFO,
    className, nameAndTags);\
}
```

Definition at line 867 of file [catch.h](#).

7.1.1.125 INTERNAL_CATCH_NTTP_REGISTER_METHOD0

```
#define INTERNAL_CATCH_NTTP_REGISTER_METHOD0(
    TestName,
    signature,
    ... )
```

Value:

```
template<typename Type>\
void reg_test(TypeList<Type>, Catch::StringRef className, Catch::NameAndTags nameAndTags)\
{\
    Catch::AutoReg( Catch::makeTestInvoker(&TestName<Type>::test), CATCH_INTERNAL_LINEINFO, className,
    nameAndTags);\
}
```

Definition at line 860 of file [catch.h](#).

7.1.1.126 INTERNAL_CATCH_REACT

```
#define INTERNAL_CATCH_REACT(  
    handler ) handler.complete();
```

Definition at line 2700 of file [catch.h](#).

7.1.1.127 INTERNAL_CATCH_REGISTER_ENUM

```
#define INTERNAL_CATCH_REGISTER_ENUM(  
    enumName,  
    ... )
```

Value:

```
namespace Catch { \  
    template<> struct StringMaker<enumName> { \  
        static std::string convert( enumName value ) { \  
            static const auto& enumInfo =  
                ::Catch::getMutableRegistryHub().getMutableEnumValuesRegistry().registerEnum( #enumName, #__VA_ARGS__,  
                { __VA_ARGS__ } ); \  
            return static_cast<std::string>(enumInfo.lookup( static_cast<int>( value ) )); \  
        } \  
    }; \  
}
```

Definition at line 2172 of file [catch.h](#).

7.1.1.128 INTERNAL_CATCH_REGISTER_TESTCASE

```
#define INTERNAL_CATCH_REGISTER_TESTCASE(  
    Function,  
    ... )
```

Value:

```
CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \  
CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \  
Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker( Function ),  
CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ __VA_ARGS__ } ); /* NOLINT */ \  
CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
```

Definition at line 1080 of file [catch.h](#).

7.1.1.129 INTERNAL_CATCH_REMOVE_PARENS

```
#define INTERNAL_CATCH_REMOVE_PARENS(  
    ... ) INTERNAL_CATCH_EXPAND1( INTERNAL_CATCH_DEF __VA_ARGS__ )
```

Definition at line 755 of file [catch.h](#).

7.1.1.130 INTERNAL_CATCH_REMOVE_PARENS_10_ARG

```
#define INTERNAL_CATCH_REMOVE_PARENS_10_ARG(  
    _0,  
    _1,  
    _2,  
    _3,  
    _4,  
    _5,  
    _6,  
    _7,  
    _8,  
    _9 ) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_9_ARG(_1, _↵  
2, _3, _4, _5, _6, _7, _8, _9)
```

Definition at line 777 of file [catch.h](#).

7.1.1.131 INTERNAL_CATCH_REMOVE_PARENS_11_ARG

```
#define INTERNAL_CATCH_REMOVE_PARENS_11_ARG(  
    _0,  
    _1,  
    _2,  
    _3,  
    _4,  
    _5,  
    _6,  
    _7,  
    _8,  
    _9,  
    _10 ) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_10_ARG(_1,  
_2, _3, _4, _5, _6, _7, _8, _9, _10)
```

Definition at line 778 of file [catch.h](#).

7.1.1.132 INTERNAL_CATCH_REMOVE_PARENS_1_ARG

```
#define INTERNAL_CATCH_REMOVE_PARENS_1_ARG(  
    _0 ) INTERNAL_CATCH_REMOVE_PARENS(_0)
```

Definition at line 768 of file [catch.h](#).

7.1.1.133 INTERNAL_CATCH_REMOVE_PARENS_2_ARG

```
#define INTERNAL_CATCH_REMOVE_PARENS_2_ARG(  
    _0,  
    _1 ) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_1)
```

Definition at line 769 of file [catch.h](#).

7.1.1.134 INTERNAL_CATCH_REMOVE_PARENS_3_ARG

```
#define INTERNAL_CATCH_REMOVE_PARENS_3_ARG(  
    _0,  
    _1,  
    _2 ) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_2_ARG(_1, _↵  
2)
```

Definition at line 770 of file [catch.h](#).

7.1.1.135 INTERNAL_CATCH_REMOVE_PARENS_4_ARG

```
#define INTERNAL_CATCH_REMOVE_PARENS_4_ARG(  
    _0,  
    _1,  
    _2,  
    _3 ) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_3_ARG(_1, _↵  
2, _3)
```

Definition at line 771 of file [catch.h](#).

7.1.1.136 INTERNAL_CATCH_REMOVE_PARENS_5_ARG

```
#define INTERNAL_CATCH_REMOVE_PARENS_5_ARG(  
    _0,  
    _1,  
    _2,  
    _3,  
    _4 ) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_4_ARG(_1, _↵  
2, _3, _4)
```

Definition at line 772 of file [catch.h](#).

7.1.1.137 INTERNAL_CATCH_REMOVE_PARENS_6_ARG

```
#define INTERNAL_CATCH_REMOVE_PARENS_6_ARG(  
    _0,  
    _1,  
    _2,  
    _3,  
    _4,  
    _5 ) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_5_ARG(_1, _↵  
2, _3, _4, _5)
```

Definition at line 773 of file [catch.h](#).

7.1.1.138 INTERNAL_CATCH_REMOVE_PARENS_7_ARG

```
#define INTERNAL_CATCH_REMOVE_PARENS_7_ARG (
    _0,
    _1,
    _2,
    _3,
    _4,
    _5,
    _6 ) INTERNAL_CATCH_REMOVE_PARENS (_0), INTERNAL_CATCH_REMOVE_PARENS_6_ARG (_1, _↵
2, _3, _4, _5, _6)
```

Definition at line 774 of file [catch.h](#).

7.1.1.139 INTERNAL_CATCH_REMOVE_PARENS_8_ARG

```
#define INTERNAL_CATCH_REMOVE_PARENS_8_ARG (
    _0,
    _1,
    _2,
    _3,
    _4,
    _5,
    _6,
    _7 ) INTERNAL_CATCH_REMOVE_PARENS (_0), INTERNAL_CATCH_REMOVE_PARENS_7_ARG (_1, _↵
2, _3, _4, _5, _6, _7)
```

Definition at line 775 of file [catch.h](#).

7.1.1.140 INTERNAL_CATCH_REMOVE_PARENS_9_ARG

```
#define INTERNAL_CATCH_REMOVE_PARENS_9_ARG (
    _0,
    _1,
    _2,
    _3,
    _4,
    _5,
    _6,
    _7,
    _8 ) INTERNAL_CATCH_REMOVE_PARENS (_0), INTERNAL_CATCH_REMOVE_PARENS_8_ARG (_1, _↵
2, _3, _4, _5, _6, _7, _8)
```

Definition at line 776 of file [catch.h](#).

7.1.1.141 INTERNAL_CATCH_REMOVE_PARENS_GEN

```
#define INTERNAL_CATCH_REMOVE_PARENS_GEN (
    ... ) INTERNAL_CATCH_VA_NARGS_IMPL (__VA_ARGS__, INTERNAL_CATCH_REMOVE_PARENS_11_ARG, INTERNAL_CA
__VA_ARGS__)
```

Definition at line 904 of file [catch.h](#).

7.1.1.142 INTERNAL_CATCH_SECTION

```
#define INTERNAL_CATCH_SECTION(
    ... )
```

Value:

```
CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
if( Catch::Section const& INTERNAL_CATCH_UNIQUE_NAME( catch_internal_Section ) = Catch::SectionInfo(
    CATCH_INTERNAL_LINEINFO, __VA_ARGS__ ) ) \
CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
```

Definition at line 2930 of file [catch.h](#).

7.1.1.143 INTERNAL_CATCH_STRINGIZE

```
#define INTERNAL_CATCH_STRINGIZE(
    ... ) INTERNAL_CATCH_STRINGIZE2( __VA_ARGS__ )
```

Definition at line 741 of file [catch.h](#).

7.1.1.144 INTERNAL_CATCH_STRINGIZE2

```
#define INTERNAL_CATCH_STRINGIZE2(
    ... ) #__VA_ARGS__
```

Definition at line 743 of file [catch.h](#).

7.1.1.145 INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS

```
#define INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS(
    param ) INTERNAL_CATCH_STRINGIZE( INTERNAL_CATCH_REMOVE_PARENS( param ) )
```

Definition at line 744 of file [catch.h](#).

7.1.1.146 INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE

```
#define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE(
    Name,
    Tags,
    TmplList ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_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, TmplList )
```

Definition at line 1208 of file [catch.h](#).

7.1.1.147 INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_2

```
#define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_2(
    TestName,
    TestFunc,
    Name,
    Tags,
    TmplList )
```

Value:

```
CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
template<typename TestType> static void TestFunc(); \
namespace {\ \
    namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName) { \
        INTERNAL_CATCH_TYPE_GEN \
        template<typename... Types> \
        struct TestName { \
            void reg_tests() { \
                int index = 0; \
                using expander = int[]; \
                (void)expander{ (Catch::AutoReg( Catch::makeTestInvoker( &TestFunc<Types> ), \
CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ Name " - " + \
std::string(INTERNAL_CATCH_STRINGIZE(TmplList)) + " - " + std::to_string(index), Tags } ), index++)... \
}; /* NOLINT */ \
            } \
        }; \
        static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = []() { \
            using TestInit = typename convert<TestName, TmplList>::type; \
            TestInit t; \
            t.reg_tests(); \
            return 0; \
        }(); \
    } \
CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
template<typename TestType> \
static void TestFunc()
```

Definition at line 1181 of file [catch.h](#).

7.1.1.148 INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD

```
#define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD(
    ClassName,
    Name,
    Tags,
    TmplList ) INTERNAL_CATCH_TEMPLATE_LIST_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_ ), 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, Name, Tags, TmplList )
```

Definition at line 1338 of file [catch.h](#).

7.1.1.149 INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_2

```
#define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_2(
    TestNameClass,
    TestName,
    ClassName,
    Name,
    Tags,
    TmplList )
```

Value:

```

CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
template<typename TestType> \
struct TestName : INTERNAL_CATCH_REMOVE_PARENS (ClassName <TestType>) { \
    void test(); \
}; \
namespace { \
namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName) { \
    INTERNAL_CATCH_TYPE_GEN \
    template<typename...Types> \
    struct TestNameClass { \
        void reg_tests() { \
            int index = 0; \
            using expander = int[]; \
            (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestName<Types>::test ), \
CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ Name " - " + \
std::string(INTERNAL_CATCH_STRINGIZE(TmplList)) + " - " + std::to_string(index), Tags } ), index++)... \
}; /* NOLINT */ \
        } \
    }; \
    static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = []() { \
        using TestInit = typename convert<TestNameClass, TmplList>::type; \
        TestInit t; \
        t.reg_tests(); \
        return 0; \
    }(); \
} \
CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
template<typename TestType> \
void TestName<TestType>::test()

```

Definition at line 1308 of file [catch.h](#).

7.1.1.150 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE

```

#define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( \
    Name, \
    Tags, \
    ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2( 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, typename T, __VA_ARGS__ )

```

Definition at line 1166 of file [catch.h](#).

7.1.1.151 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2

```

#define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2( \
    TestName, \
    TestFuncName, \
    Name, \
    Tags, \
    Signature, \
    TmplTypes, \
    TypesList )

```

Definition at line 1132 of file [catch.h](#).

7.1.1.152 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD

```

#define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( \
    ClassName, \
    Name, \
    Tags, \
    ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_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_ ), 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, Name, Tags, typename T, __VA_ARGS__ )

```

Definition at line 1293 of file [catch.h](#).

7.1.1.153 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2

```
#define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2 (
    TestNameClass,
    TestName,
    ClassName,
    Name,
    Tags,
    Signature,
    ImplTypes,
    TypesList )
```

Definition at line 1256 of file [catch.h](#).

7.1.1.154 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG

```
#define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG (
    ClassName,
    Name,
    Tags,
    Signature,
    ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_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_ ), 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, Name, Tags, Signature, __VA_ARGS__ )
```

Definition at line 1301 of file [catch.h](#).

7.1.1.155 INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG

```
#define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG (
    Name,
    Tags,
    Signature,
    ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2 (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__)
```

Definition at line 1174 of file [catch.h](#).

7.1.1.156 INTERNAL_CATCH_TEMPLATE_TEST_CASE

```
#define INTERNAL_CATCH_TEMPLATE_TEST_CASE (
    Name,
    Tags,
    ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_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, typename TestType, __VA_ARGS__ )
```

Definition at line 1117 of file [catch.h](#).

7.1.1.157 INTERNAL_CATCH_TEMPLATE_TEST_CASE_2

```
#define INTERNAL_CATCH_TEMPLATE_TEST_CASE_2(
    TestName,
    TestFunc,
    Name,
    Tags,
    Signature,
    ... )
```

Value:

```
CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
INTERNAL_CATCH_DECLARE_SIG_TEST(TestFunc, INTERNAL_CATCH_REMOVE_PARENS(Signature));\
namespace {\
namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName){\
    INTERNAL_CATCH_TYPE_GEN\
    INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS(Signature))\
    INTERNAL_CATCH_NTTP_REG_GEN(TestFunc, INTERNAL_CATCH_REMOVE_PARENS(Signature))\
    template<typename...Types> \
    struct TestName{\
        TestName(){\
            int index = 0; \
            constexpr char const* tpl_types[] = \
{CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, __VA_ARGS__)};\
            using expander = int[];\
            (void)expander{(reg_test(Types{}, Catch::NameAndTags{ Name " - " +\
std::string(tpl_types[index]), Tags } ), index++)... };/* NOLINT */ \
        }\
    };\
    static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
        TestName<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(__VA_ARGS__)>();\
        return 0;\
    }();\
}\
}\
CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
INTERNAL_CATCH_DEFINE_SIG_TEST(TestFunc, INTERNAL_CATCH_REMOVE_PARENS(Signature))
```

Definition at line 1087 of file [catch.h](#).

7.1.1.158 INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD

```
#define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD(
    ClassName,
    Name,
    Tags,
    ... ) 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__ )
```

Definition at line 1241 of file [catch.h](#).

7.1.1.159 INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2

```
#define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2(
    TestNameClass,
    TestName,
    ClassName,
    Name,
    Tags,
```

```
Signature,
... )
```

Value:

```
CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
namespace {\
namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName) { \
INTERNAL_CATCH_TYPE_GEN\
INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS(Signature))\
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD(TestName, ClassName,
INTERNAL_CATCH_REMOVE_PARENS(Signature));\
INTERNAL_CATCH_NTTP_REG_METHOD_GEN(TestName, INTERNAL_CATCH_REMOVE_PARENS(Signature))\
template<typename...Types> \
struct TestNameClass{\
TestNameClass(){\
int index = 0;\
constexpr char const* tpl_types[] = \
{CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, __VA_ARGS__)};\
using expander = int[];\
(void)expander{(reg_test(Types{}, #ClassName, Catch::NameAndTags{ Name " - " +
std::string(tmpl_types[index]), Tags } ), index++)... };/* NOLINT */ \
}\
};\
static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
TestNameClass<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(__VA_ARGS__)>();\
return 0;\
}();\
}\
}\
CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, INTERNAL_CATCH_REMOVE_PARENS(Signature))
```

Definition at line 1211 of file [catch.h](#).

7.1.1.160 INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG

```
#define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG(
    ClassName,
    Name,
    Tags,
    Signature,
    ... ) 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__ )
```

Definition at line 1249 of file [catch.h](#).

7.1.1.161 INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG

```
#define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG(
    Name,
    Tags,
    Signature,
    ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_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__ )
```

Definition at line 1125 of file [catch.h](#).

7.1.1.162 INTERNAL_CATCH_TEST

```
#define INTERNAL_CATCH_TEST(
    macroName,
    resultDisposition,
    ... )
```

Value:

```
do { \
    CATCH_INTERNAL_IGNORE_BUT_WARN(__VA_ARGS__); \
    Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
    CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition ); \
    INTERNAL_CATCH_TRY { \
        CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
        CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS \
        catchAssertionHandler.handleExpr( Catch::Decomposer() <= __VA_ARGS__ ); \
        CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
    } INTERNAL_CATCH_CATCH( catchAssertionHandler ) \
    INTERNAL_CATCH_REACT( catchAssertionHandler ) \
} while( (void)0, (false) && static_cast<bool>( !!( __VA_ARGS__ ) ) )
```

Definition at line 2703 of file [catch.h](#).

7.1.1.163 INTERNAL_CATCH_TEST_CASE_METHOD

```
#define INTERNAL_CATCH_TEST_CASE_METHOD(
    ClassName,
    ... ) INTERNAL_CATCH_TEST_CASE_METHOD2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_↵
H_T_E_S_T_ ), ClassName, __VA_ARGS__ )
```

Definition at line 1076 of file [catch.h](#).

7.1.1.164 INTERNAL_CATCH_TEST_CASE_METHOD2

```
#define INTERNAL_CATCH_TEST_CASE_METHOD2(
    TestName,
    ClassName,
    ... )
```

Value:

```
CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
namespace{ \
    struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName) { \
        void test(); \
    }; \
    Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar ) ( Catch::makeTestInvoker(
    &TestName::test ), CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ __VA_ARGS__ } ); /* NOLINT
*/ \
} \
CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
void TestName::test()
```

Definition at line 1065 of file [catch.h](#).

7.1.1.165 INTERNAL_CATCH_TESTCASE

```
#define INTERNAL_CATCH_TESTCASE(
    ... ) INTERNAL_CATCH_TESTCASE2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_↵
T_ ), __VA_ARGS__ )
```

Definition at line 1054 of file [catch.h](#).

7.1.1.166 INTERNAL_CATCH_TESTCASE2

```
#define INTERNAL_CATCH_TESTCASE2(
    TestName,
    ... )
```

Value:

```
static void TestName(); \
CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
namespace{ Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker(
&TestName ), CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ __VA_ARGS__ } ); } /*
NOLINT */ \
CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
static void TestName()
```

Definition at line 1047 of file [catch.h](#).

7.1.1.167 INTERNAL_CATCH_THROWS

```
#define INTERNAL_CATCH_THROWS(
    macroName,
    resultDisposition,
    ... )
```

Value:

```
do { \
    Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition); \
    if( catchAssertionHandler.allowThrows() ) \
        try { \
            static_cast<void>(__VA_ARGS__); \
            catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
        } \
        catch( ... ) { \
            catchAssertionHandler.handleExceptionThrownAsExpected(); \
        } \
    else \
        catchAssertionHandler.handleThrowingCallSkipped(); \
    INTERNAL_CATCH_REACT( catchAssertionHandler ) \
} while( false )
```

Definition at line 2741 of file [catch.h](#).

7.1.1.168 INTERNAL_CATCH_THROWS_AS

```
#define INTERNAL_CATCH_THROWS_AS(
    macroName,
    exceptionType,
    resultDisposition,
    expr )
```

Value:

```
do { \
    Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
CATCH_INTERNAL_STRINGIFY(expr) ", " CATCH_INTERNAL_STRINGIFY(exceptionType), resultDisposition ); \
    if( catchAssertionHandler.allowThrows() ) \
        try { \
            static_cast<void>(expr); \
            catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
        } \
        catch( exceptionType const& ) { \
            catchAssertionHandler.handleExceptionThrownAsExpected(); \
        } \
        catch( ... ) { \
            catchAssertionHandler.handleUnexpectedInflightException(); \
        } \
    else \
        catchAssertionHandler.handleThrowingCallSkipped(); \
    INTERNAL_CATCH_REACT( catchAssertionHandler ) \
} while( false )
```

Definition at line 2758 of file [catch.h](#).

7.1.1.169 INTERNAL_CATCH_THROWS_MATCHES

```
#define INTERNAL_CATCH_THROWS_MATCHES(
    macroName,
    exceptionType,
    resultDisposition,
    matcher,
    ... )
```

Value:

```
do { \
    Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
    CATCH_INTERNAL_STRINGIFY(__VA_ARGS__) ", " CATCH_INTERNAL_STRINGIFY(exceptionType) ", "
    CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
    if( catchAssertionHandler.allowThrows() ) \
        try { \
            static_cast<void>(__VA_ARGS__ ); \
            catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
        } \
        catch( exceptionType const& ex ) { \
            catchAssertionHandler.handleExpr( Catch::makeMatchExpr( ex, matcher, #matcher##_catch_sr )
); \
    } \
    catch( ... ) { \
        catchAssertionHandler.handleUnexpectedInflightException(); \
    } \
    else \
        catchAssertionHandler.handleThrowingCallSkipped(); \
    INTERNAL_CATCH_REACT( catchAssertionHandler ) \
} while( false )
```

Definition at line 3814 of file [catch.h](#).

7.1.1.170 INTERNAL_CATCH_THROWS_STR_MATCHES

```
#define INTERNAL_CATCH_THROWS_STR_MATCHES(
    macroName,
    resultDisposition,
    matcher,
    ... )
```

Value:

```
do { \
    Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
    CATCH_INTERNAL_STRINGIFY(__VA_ARGS__) ", " CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
    if( catchAssertionHandler.allowThrows() ) \
        try { \
            static_cast<void>(__VA_ARGS__); \
            catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
        } \
        catch( ... ) { \
            Catch::handleExceptionMatchExpr( catchAssertionHandler, matcher, #matcher##_catch_sr ); \
        } \
    else \
        catchAssertionHandler.handleThrowingCallSkipped(); \
    INTERNAL_CATCH_REACT( catchAssertionHandler ) \
} while( false )
```

Definition at line 2800 of file [catch.h](#).

7.1.1.171 INTERNAL_CATCH_TRANSLATE_EXCEPTION

```
#define INTERNAL_CATCH_TRANSLATE_EXCEPTION(
    signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION2( INTERNAL_CATCH_UNIQUE_NAME(
catch_internal_ExceptionTranslator ), signature )
```

Definition at line 3068 of file [catch.h](#).

7.1.1.172 INTERNAL_CATCH_TRANSLATE_EXCEPTION2

```
#define INTERNAL_CATCH_TRANSLATE_EXCEPTION2(
    translatorName,
    signature )
```

Value:

```
static std::string translatorName( signature ); \
CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
namespace{ Catch::ExceptionTranslatorRegistrar INTERNAL_CATCH_UNIQUE_NAME(
    catch_internal_ExceptionRegistrar )( &translatorName ); } \
CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
static std::string translatorName( signature )
```

Definition at line 3060 of file [catch.h](#).

7.1.1.173 INTERNAL_CATCH_TRY

```
#define INTERNAL_CATCH_TRY
```

Definition at line 2690 of file [catch.h](#).

7.1.1.174 INTERNAL_CATCH_TYPE_GEN

```
#define INTERNAL_CATCH_TYPE_GEN
```

Definition at line 782 of file [catch.h](#).

7.1.1.175 INTERNAL_CATCH_UNIQUE_NAME

```
#define INTERNAL_CATCH_UNIQUE_NAME(
    name ) INTERNAL_CATCH_UNIQUE_NAME_LINE( name, __COUNTER__ )
```

Definition at line 471 of file [catch.h](#).

7.1.1.176 INTERNAL_CATCH_UNIQUE_NAME_LINE

```
#define INTERNAL_CATCH_UNIQUE_NAME_LINE(
    name,
    line ) INTERNAL_CATCH_UNIQUE_NAME_LINE2( name, line )
```

Definition at line 469 of file [catch.h](#).

7.1.1.177 INTERNAL_CATCH_UNIQUE_NAME_LINE2

```
#define INTERNAL_CATCH_UNIQUE_NAME_LINE2(
    name,
    line ) name##line
```

Definition at line 468 of file [catch.h](#).

7.1.1.178 INTERNAL_CATCH_UNSCOPED_INFO

```
#define INTERNAL_CATCH_UNSCOPED_INFO(
    macroName,
    log )    Catch::getResultCapture().emplaceUnscopedMessage( Catch::MessageBuilder(
macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info ) << log )
```

Definition at line 2795 of file [catch.h](#).

7.1.1.179 INTERNAL_CATCH_VA_NARGS_IMPL

```
#define INTERNAL_CATCH_VA_NARGS_IMPL(
    _0,
    _1,
    _2,
    _3,
    _4,
    _5,
    _6,
    _7,
    _8,
    _9,
    _10,
    N,
    ... ) N
```

Definition at line 780 of file [catch.h](#).

7.1.1.180 INTERNAL_CHECK_THAT

```
#define INTERNAL_CHECK_THAT(
    macroName,
    matcher,
    resultDisposition,
    arg )
```

Value:

```
do { \
    Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
CATCH_INTERNAL_STRINGIFY(arg) ", " CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
    INTERNAL_CATCH_TRY { \
        catchAssertionHandler.handleExpr( Catch::makeMatchExpr( arg, matcher, #matcher##_catch_sr ) ); \
    } INTERNAL_CATCH_CATCH( catchAssertionHandler ) \
    INTERNAL_CATCH_REACT( catchAssertionHandler ) \
} while( false )
```

Definition at line 3804 of file [catch.h](#).

7.1.1.181 METHOD_AS_TEST_CASE

```
#define METHOD_AS_TEST_CASE(
    method,
    ... )    INTERNAL_CATCH_METHOD_AS_TEST_CASE( method, __VA_ARGS__ )
```

Definition at line 17714 of file [catch.h](#).

7.1.1.182 REGISTER_TEST_CASE

```
#define REGISTER_TEST_CASE(  
    Function,  
    ... ) INTERNAL_CATCH_REGISTER_TESTCASE( Function, __VA_ARGS__ )
```

Definition at line 17715 of file [catch.h](#).

7.1.1.183 REQUIRE

```
#define REQUIRE(  
    ... ) INTERNAL_CATCH_TEST( "REQUIRE", Catch::ResultDisposition::Normal, __VA_↵  
    ARGS__ )
```

Definition at line 17676 of file [catch.h](#).

7.1.1.184 REQUIRE_FALSE

```
#define REQUIRE_FALSE(  
    ... ) INTERNAL_CATCH_TEST( "REQUIRE_FALSE", Catch::ResultDisposition::Normal |  
    Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
```

Definition at line 17677 of file [catch.h](#).

7.1.1.185 REQUIRE_NOTHROW

```
#define REQUIRE_NOTHROW(  
    ... ) INTERNAL_CATCH_NO_THROW( "REQUIRE_NOTHROW", Catch::ResultDisposition::Normal,  
    __VA_ARGS__ )
```

Definition at line 17685 of file [catch.h](#).

7.1.1.186 REQUIRE_THAT

```
#define REQUIRE_THAT(  
    arg,  
    matcher ) INTERNAL_CHECK_THAT( "REQUIRE_THAT", matcher, Catch::ResultDisposition::Normal,  
    arg )
```

Definition at line 17704 of file [catch.h](#).

7.1.1.187 REQUIRE_THROWS

```
#define REQUIRE_THROWS(  
    ... ) INTERNAL_CATCH_THROWS( "REQUIRE_THROWS", Catch::ResultDisposition::Normal,  
    __VA_ARGS__ )
```

Definition at line 17679 of file [catch.h](#).

7.1.1.188 REQUIRE_THROWS_AS

```
#define REQUIRE_THROWS_AS(
    expr,
    exceptionType ) INTERNAL_CATCH_THROWS_AS( "REQUIRE_THROWS_AS", exceptionType,
Catch::ResultDisposition::Normal, expr )
```

Definition at line 17680 of file [catch.h](#).

7.1.1.189 REQUIRE_THROWS_MATCHES

```
#define REQUIRE_THROWS_MATCHES(
    expr,
    exceptionType,
    matcher ) INTERNAL_CATCH_THROWS_MATCHES( "REQUIRE_THROWS_MATCHES", exceptionType,
Catch::ResultDisposition::Normal, matcher, expr )
```

Definition at line 17683 of file [catch.h](#).

7.1.1.190 REQUIRE_THROWS_WITH

```
#define REQUIRE_THROWS_WITH(
    expr,
    matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "REQUIRE_THROWS_WITH", Catch::ResultDisposition::Normal,
matcher, expr )
```

Definition at line 17681 of file [catch.h](#).

7.1.1.191 SCENARIO

```
#define SCENARIO(
    ... ) TEST_CASE( "Scenario: " __VA_ARGS__ )
```

Definition at line 17760 of file [catch.h](#).

7.1.1.192 SCENARIO_METHOD

```
#define SCENARIO_METHOD(
    className,
    ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, "Scenario: " __VA_ARGS__ )
```

Definition at line 17761 of file [catch.h](#).

7.1.1.193 SECTION

```
#define SECTION(
    ... ) INTERNAL_CATCH_SECTION( __VA_ARGS__ )
```

Definition at line 17716 of file [catch.h](#).

7.1.1.194 STATIC_REQUIRE

```
#define STATIC_REQUIRE(  
    ... ) static_assert( __VA_ARGS__, #__VA_ARGS__ ); SUCCEED( #__VA_ARGS__ )
```

Definition at line 17748 of file [catch.h](#).

7.1.1.195 STATIC_REQUIRE_FALSE

```
#define STATIC_REQUIRE_FALSE(  
    ... ) static_assert( !(__VA_ARGS__), "!( " #__VA_ARGS__ " )" ); SUCCEED( "!( " #↵  
    __VA_ARGS__ " )" )
```

Definition at line 17749 of file [catch.h](#).

7.1.1.196 SUCCEED

```
#define SUCCEED(  
    ... ) INTERNAL_CATCH_MSG( "SUCCEED", Catch::ResultWas::Ok, Catch::ResultDisposition::ContinueOn  
    __VA_ARGS__ )
```

Definition at line 17720 of file [catch.h](#).

7.1.1.197 TEMPLATE_LIST_TEST_CASE

```
#define TEMPLATE_LIST_TEST_CASE(  
    ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE( __VA_ARGS__ )
```

Definition at line 17732 of file [catch.h](#).

7.1.1.198 TEMPLATE_LIST_TEST_CASE_METHOD

```
#define TEMPLATE_LIST_TEST_CASE_METHOD(  
    className,  
    ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD( className, __VA_ARGS__ )
```

Definition at line 17733 of file [catch.h](#).

7.1.1.199 TEMPLATE_PRODUCT_TEST_CASE

```
#define TEMPLATE_PRODUCT_TEST_CASE(  
    ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ )
```

Definition at line 17728 of file [catch.h](#).

7.1.1.200 TEMPLATE_PRODUCT_TEST_CASE_METHOD

```
#define TEMPLATE_PRODUCT_TEST_CASE_METHOD(  
    className,  
    ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__  
)
```

Definition at line 17730 of file [catch.h](#).

7.1.1.201 TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG

```
#define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG(  
    className,  
    ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__  
ARGS__ )
```

Definition at line 17731 of file [catch.h](#).

7.1.1.202 TEMPLATE_PRODUCT_TEST_CASE_SIG

```
#define TEMPLATE_PRODUCT_TEST_CASE_SIG(  
    ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( __VA_ARGS__ )
```

Definition at line 17729 of file [catch.h](#).

7.1.1.203 TEMPLATE_TEST_CASE

```
#define TEMPLATE_TEST_CASE(  
    ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
```

Definition at line 17724 of file [catch.h](#).

7.1.1.204 TEMPLATE_TEST_CASE_METHOD

```
#define TEMPLATE_TEST_CASE_METHOD(  
    className,  
    ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( className, __VA_ARGS__ )
```

Definition at line 17726 of file [catch.h](#).

7.1.1.205 TEMPLATE_TEST_CASE_METHOD_SIG

```
#define TEMPLATE_TEST_CASE_METHOD_SIG(  
    className,  
    ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
```

Definition at line 17727 of file [catch.h](#).

7.1.1.206 TEMPLATE_TEST_CASE_SIG

```
#define TEMPLATE_TEST_CASE_SIG(
    ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ )
```

Definition at line 17725 of file [catch.h](#).

7.1.1.207 TEST_CASE

```
#define TEST_CASE(
    ... ) INTERNAL_CATCH_TESTCASE( __VA_ARGS__ )
```

Definition at line 17712 of file [catch.h](#).

7.1.1.208 TEST_CASE_METHOD

```
#define TEST_CASE_METHOD(
    className,
    ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, __VA_ARGS__ )
```

Definition at line 17713 of file [catch.h](#).

7.1.1.209 THEN

```
#define THEN(
    desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " Then: " << desc )
```

Definition at line 17767 of file [catch.h](#).

7.1.1.210 UNSCOPED_INFO

```
#define UNSCOPED_INFO(
    msg ) INTERNAL_CATCH_UNSCOPED_INFO( "UNSCOPED_INFO", msg )
```

Definition at line 17708 of file [catch.h](#).

7.1.1.211 WARN

```
#define WARN(
    msg ) INTERNAL_CATCH_MSG( "WARN", Catch::ResultWas::Warning, Catch::ResultDisposition::ContinueOnFailure,
    msg )
```

Definition at line 17709 of file [catch.h](#).

7.1.1.212 WHEN

```
#define WHEN(
    desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " When: " << desc )
```

Definition at line 17765 of file [catch.h](#).

7.1.2 Function Documentation

7.1.2.1 operator""_catch_sr()

```
constexpr auto operator""_catch_sr (
    char const * rawChars,
    std::size_t size ) -> Catch::StringRef [constexpr], [noexcept]
```

Definition at line 685 of file [catch.h](#).

7.1.2.2 operator<<()

```
std::ostream & operator<< (
    std::ostream & ,
    Catch_global_namespace_dummy )
```

7.2 catch.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * Catch v2.13.10
00003  * Generated: 2022-10-16 11:01:23.452308
00004  * -----
00005  * This file has been merged from multiple headers. Please don't edit it directly
00006  * Copyright (c) 2022 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 10
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 #   ifdef __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 #   if (__cplusplus >= 201402L) || (defined(_MSVC_LANG) && _MSVC_LANG >= 201402L)
00128 #       define CATCH_CPP14_OR_GREATER
00129 #   endif
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__) /*
NOLINT(cppcoreguidelines-pro-type-vararg, hicpp-vararg) */
00166 #   endif
00167
00168 #   define CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
    _Pragma( "clang diagnostic ignored \\"-Wexit-time-destructors\\" ) \
    _Pragma( "clang diagnostic ignored \\"-Wglobal-constructors\\"")
00169
00170 #   define CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS \
    _Pragma( "clang diagnostic ignored \\"-Wparentheses\\"")
00171
00172 #   define CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
    _Pragma( "clang diagnostic ignored \\"-Wunused-variable\\"")
00173
00174 #   define CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
    _Pragma( "clang diagnostic ignored \\"-Wgnu-zero-variadic-macro-arguments\\"")
00175
00176 #   define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
    _Pragma( "clang diagnostic ignored \\"-Wunused-template\\"")
00177
00178 #   define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
    _Pragma( "clang diagnostic ignored \\"-Wunused-template\\"")
00179
00180 #   define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
    _Pragma( "clang diagnostic ignored \\"-Wunused-template\\"")
00181
00182 #   define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
    _Pragma( "clang diagnostic ignored \\"-Wunused-template\\"")
00183
00184 #endif // __clang__
00185
00186 // Assume that non-Windows platforms support posix signals by default
00187 #if !defined(CATCH_PLATFORM_WINDOWS)
00188 #define CATCH_INTERNAL_CONFIG_POSIX_SIGNALS
00189 #endif
00190
00191 // We know some environments not to support full POSIX signals
00192 #if defined(__CYGWIN__) || defined(__QNX__) || defined(__EMSCRIPTEN__) || defined(__DJGPP__)
00193 #define CATCH_INTERNAL_CONFIG_NO_POSIX_SIGNALS
00194 #endif
00195
00196 #if defined(__OS400__)
00197 #define CATCH_INTERNAL_CONFIG_NO_POSIX_SIGNALS
00198 #define CATCH_CONFIG_COLOUR_NONE
00199 #endif
00200
00201 // Android somehow still does not support std::to_string
00202 #if defined(__ANDROID__)
00203 #define CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING
00204 #define CATCH_INTERNAL_CONFIG_ANDROID_LOGWRITE
00205 #endif
00206
00207 // Not all Windows environments support SEH properly
00208 #if defined(__MINGW32__)
00209 #define CATCH_INTERNAL_CONFIG_NO_WINDOWS_SEH
00210 #endif
00211
00212 // PS4
00213 #if defined(__ORBIS__)
00214 #define CATCH_INTERNAL_CONFIG_NO_NEW_CAPTURE
00215 #endif
00216
00217 // Cygwin
00218 #if defined(__CYGWIN__)
00219 // Required for some versions of Cygwin to declare gettimeofday
00220 // see: http://stackoverflow.com/questions/36901803/gettimeofday-not-declared-in-this-scope-cygwin
00221 #define CATCH_INTERNAL_CONFIG_BSD_SOURCE
00222 #endif

```



```

00229 // some versions of cygwin (most) do not support std::to_string. Use the libstd check.
00230 // https://gcc.gnu.org/onlinedocs/gcc-4.8.2/libstdc++/api/a01053_source.html line 2812-2813
00231 # if !((__cplusplus >= 201103L) && defined(_GLIBCXX_USE_C99) \
00232      && !defined(_GLIBCXX_HAVE_BROKEN_VSWPRINTF))
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 // Universal Windows platform does not support SEH
00244 // Or console colours (or console at all...)
00245 # if defined(WINAPI_FAMILY) && (WINAPI_FAMILY == WINAPI_FAMILY_APP)
00246 #     define CATCH_CONFIG_COLOUR_NONE
00247 # else
00248 #     define CATCH_INTERNAL_CONFIG_WINDOWS_SEH
00249 # endif
00250
00251 # if !defined(__clang__) // Handle Clang masquerading for msvc
00252
00253 // MSVC traditional preprocessor needs some workaround for __VA_ARGS__
00254 // _MSVC_TRADITIONAL == 0 means new conformant preprocessor
00255 // _MSVC_TRADITIONAL == 1 means old traditional non-conformant preprocessor
00256 #     if !defined(_MSVC_TRADITIONAL) || (defined(_MSVC_TRADITIONAL) && _MSVC_TRADITIONAL)
00257 #         define CATCH_INTERNAL_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00258 #     endif // _MSVC_TRADITIONAL
00259
00260 // Only do this if we're not using clang on Windows, which uses `diagnostic push` & `diagnostic pop`
00261 #     define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION __pragma( warning(push) )
00262 #     define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION __pragma( warning(pop) )
00263 # endif // __clang__
00264
00265 #endif // _MSC_VER
00266
00267 #if defined(_REENTRANT) || defined(_MSC_VER)
00268 // Enable async processing, as -pthread is specified or no additional linking is required
00269 # define CATCH_INTERNAL_CONFIG_USE_ASYNC
00270 #endif // _MSC_VER
00271
00273 // Check if we are compiled with -fno-exceptions or equivalent
00274 #if defined(__EXCEPTIONS) || defined(__cpp_exceptions) || defined(_CPPUNWIND)
00275 #     define CATCH_INTERNAL_CONFIG_EXCEPTIONS_ENABLED
00276 #endif
00277
00279 // DJGPP
00280 #ifdef __DJGPP__
00281 #     define CATCH_INTERNAL_CONFIG_NO_WCHAR
00282 #endif // __DJGPP__
00283
00285 // Embarcadero C++Build
00286 #if defined(__BORLANDC__)
00287 #define CATCH_INTERNAL_CONFIG_POLYFILL_ISNAN
00288 #endif
00289
00291
00292 // Use of __COUNTER__ is suppressed during code analysis in
00293 // CLion/AppCode 2017.2.x and former, because __COUNTER__ is not properly
00294 // handled by it.
00295 // Otherwise all supported compilers support COUNTER macro,
00296 // but user still might want to turn it off
00297 #if ( !defined(__JETBRAINS_IDE__) || __JETBRAINS_IDE__ >= 20170300L )
00298 #define CATCH_INTERNAL_CONFIG_COUNTER
00299 #endif
00300
00302
00303 // RTX is a special version of Windows that is real time.
00304 // This means that it is detected as Windows, but does not provide
00305 // the same set of capabilities as real Windows does.
00306 #if defined(UNDER_RTSS) || defined(RTX64_BUILD)
00307 #define CATCH_INTERNAL_CONFIG_NO_WINDOWS_SEH
00308 #define CATCH_INTERNAL_CONFIG_NO_ASYNC
00309 #define CATCH_CONFIG_COLOUR_NONE
00310 #endif
00311
00312 #if !defined(_GLIBCXX_USE_C99_MATH_TR1)
00313 #define CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER
00314 #endif
00315
00316 // Various stdlib support checks that require __has_include
00317 #if defined(__has_include)
00318 // Check if string_view is available and usable
00319 #if __has_include(<string_view>) && defined(CATCH_CPP17_OR_GREATER)
00320 #     define CATCH_INTERNAL_CONFIG_CPP17_STRING_VIEW
00321 #endif
00322 #endif

```

```

00322
00323 // Check if optional is available and usable
00324 # if __has_include(<optional>) && defined(CATCH_CPP17_OR_GREATER)
00325 #     define CATCH_INTERNAL_CONFIG_CPP17_OPTIONAL
00326 # endif // __has_include(<optional>) && defined(CATCH_CPP17_OR_GREATER)
00327
00328 // Check if byte is available and usable
00329 # if __has_include(<cstdint>) && defined(CATCH_CPP17_OR_GREATER)
00330 #     include <cstdint>
00331 #     if defined(__cpp_lib_byte) && (__cpp_lib_byte > 0)
00332 #         define CATCH_INTERNAL_CONFIG_CPP17_BYTE
00333 #     endif
00334 # endif // __has_include(<cstdint>) && defined(CATCH_CPP17_OR_GREATER)
00335
00336 // Check if variant is available and usable
00337 # if __has_include(<variant>) && defined(CATCH_CPP17_OR_GREATER)
00338 #     if defined(__clang__) && (__clang_major__ < 8)
00339 // work around clang bug with libstdc++ https://bugs.llvm.org/show_bug.cgi?id=31852
00340 // fix should be in clang 8, workaround in libstdc++ 8.2
00341 #         include <ciso646>
00342 #         if defined(__GLIBCXX__) && defined(_GLIBCXX_RELEASE) && (_GLIBCXX_RELEASE < 9)
00343 #             define CATCH_CONFIG_NO_CPP17_VARIANT
00344 #         else
00345 #             define CATCH_INTERNAL_CONFIG_CPP17_VARIANT
00346 #         endif // defined(__GLIBCXX__) && defined(_GLIBCXX_RELEASE) && (_GLIBCXX_RELEASE < 9)
00347 #     else
00348 #         define CATCH_INTERNAL_CONFIG_CPP17_VARIANT
00349 #     endif // defined(__clang__) && (__clang_major__ < 8)
00350 # endif // __has_include(<variant>) && defined(CATCH_CPP17_OR_GREATER)
00351 #endif // defined(__has_include)
00352
00353 #if defined(CATCH_INTERNAL_CONFIG_COUNTER) && !defined(CATCH_CONFIG_NO_COUNTER) &&
!defined(CATCH_CONFIG_COUNTER)
00354 #     define CATCH_CONFIG_COUNTER
00355 #endif
00356 #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)
00357 #     define CATCH_CONFIG_WINDOWS_SEH
00358 #endif
00359 // This is set by default, because we assume that unix compilers are posix-signal-compatible by
default.
00360 #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)
00361 #     define CATCH_CONFIG_POSIX_SIGNALS
00362 #endif
00363 // This is set by default, because we assume that compilers with no wchar_t support are just rare
exceptions.
00364 #if !defined(CATCH_INTERNAL_CONFIG_NO_WCHAR) && !defined(CATCH_CONFIG_NO_WCHAR) &&
!defined(CATCH_CONFIG_WCHAR)
00365 #     define CATCH_CONFIG_WCHAR
00366 #endif
00367
00368 #if !defined(CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING) && !defined(CATCH_CONFIG_NO_CPP11_TO_STRING) &&
!defined(CATCH_CONFIG_CPP11_TO_STRING)
00369 #     define CATCH_CONFIG_CPP11_TO_STRING
00370 #endif
00371
00372 #if defined(CATCH_INTERNAL_CONFIG_CPP17_OPTIONAL) && !defined(CATCH_CONFIG_NO_CPP17_OPTIONAL) &&
!defined(CATCH_CONFIG_CPP17_OPTIONAL)
00373 #     define CATCH_CONFIG_CPP17_OPTIONAL
00374 #endif
00375
00376 #if defined(CATCH_INTERNAL_CONFIG_CPP17_STRING_VIEW) && !defined(CATCH_CONFIG_NO_CPP17_STRING_VIEW) &&
!defined(CATCH_CONFIG_CPP17_STRING_VIEW)
00377 #     define CATCH_CONFIG_CPP17_STRING_VIEW
00378 #endif
00379
00380 #if defined(CATCH_INTERNAL_CONFIG_CPP17_VARIANT) && !defined(CATCH_CONFIG_NO_CPP17_VARIANT) &&
!defined(CATCH_CONFIG_CPP17_VARIANT)
00381 #     define CATCH_CONFIG_CPP17_VARIANT
00382 #endif
00383
00384 #if defined(CATCH_INTERNAL_CONFIG_CPP17_BYTE) && !defined(CATCH_CONFIG_NO_CPP17_BYTE) &&
!defined(CATCH_CONFIG_CPP17_BYTE)
00385 #     define CATCH_CONFIG_CPP17_BYTE
00386 #endif
00387
00388 #if defined(CATCH_CONFIG_EXPERIMENTAL_REDIRECT)
00389 #     define CATCH_INTERNAL_CONFIG_NEW_CAPTURE
00390 #endif
00391
00392 #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)
00393 #     define CATCH_CONFIG_NEW_CAPTURE
00394 #endif
00395
00396 #if !defined(CATCH_INTERNAL_CONFIG_EXCEPTIONS_ENABLED) && !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)

```

```

00397 # define CATCH_CONFIG_DISABLE_EXCEPTIONS
00398 #endif
00399
00400 #if defined(CATCH_INTERNAL_CONFIG_POLYFILL_ISNAN) && !defined(CATCH_CONFIG_NO_POLYFILL_ISNAN) &&
    !defined(CATCH_CONFIG_POLYFILL_ISNAN)
00401 # define CATCH_CONFIG_POLYFILL_ISNAN
00402 #endif
00403
00404 #if defined(CATCH_INTERNAL_CONFIG_USE_ASYNC) && !defined(CATCH_INTERNAL_CONFIG_NO_ASYNC) &&
    !defined(CATCH_CONFIG_NO_USE_ASYNC) && !defined(CATCH_CONFIG_USE_ASYNC)
00405 # define CATCH_CONFIG_USE_ASYNC
00406 #endif
00407
00408 #if defined(CATCH_INTERNAL_CONFIG_ANDROID_LOGWRITE) && !defined(CATCH_CONFIG_NO_ANDROID_LOGWRITE) &&
    !defined(CATCH_CONFIG_ANDROID_LOGWRITE)
00409 # define CATCH_CONFIG_ANDROID_LOGWRITE
00410 #endif
00411
00412 #if defined(CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER) && !defined(CATCH_CONFIG_NO_GLOBAL_NEXTAFTER) &&
    !defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
00413 # define CATCH_CONFIG_GLOBAL_NEXTAFTER
00414 #endif
00415
00416 // Even if we do not think the compiler has that warning, we still have
00417 // to provide a macro that can be used by the code.
00418 #if !defined(CATCH_INTERNAL_START_WARNINGS_SUPPRESSION)
00419 # define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION
00420 #endif
00421 #if !defined(CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION)
00422 # define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
00423 #endif
00424 #if !defined(CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS)
00425 # define CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS
00426 #endif
00427 #if !defined(CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS)
00428 # define CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS
00429 #endif
00430 #if !defined(CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS)
00431 # define CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS
00432 #endif
00433 #if !defined(CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS)
00434 # define CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS
00435 #endif
00436
00437 // The goal of this macro is to avoid evaluation of the arguments, but
00438 // still have the compiler warn on problems inside...
00439 #if !defined(CATCH_INTERNAL_IGNORE_BUT_WARN)
00440 # define CATCH_INTERNAL_IGNORE_BUT_WARN(...)
00441 #endif
00442
00443 #if defined(__APPLE__) && defined(__apple_build_version__) && (__clang_major__ < 10)
00444 # undef CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
00445 #elif defined(__clang__) && (__clang_major__ < 5)
00446 # undef CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
00447 #endif
00448
00449 #if !defined(CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS)
00450 # define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
00451 #endif
00452
00453 #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
00454 #define CATCH_TRY if ((true))
00455 #define CATCH_CATCH_ALL if ((false))
00456 #define CATCH_CATCH_ANON(type) if ((false))
00457 #else
00458 #define CATCH_TRY try
00459 #define CATCH_CATCH_ALL catch (...)
00460 #define CATCH_CATCH_ANON(type) catch (type)
00461 #endif
00462
00463 #if defined(CATCH_INTERNAL_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR) &&
    !defined(CATCH_CONFIG_NO_TRADITIONAL_MSVC_PREPROCESSOR) &&
    !defined(CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR)
00464 #define CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00465 #endif
00466
00467 // end catch_compiler_capabilities.h
00468 #define INTERNAL_CATCH_UNIQUE_NAME_LINE2( name, line ) name##line
00469 #define INTERNAL_CATCH_UNIQUE_NAME_LINE( name, line ) INTERNAL_CATCH_UNIQUE_NAME_LINE2( name, line )
00470 #ifdef CATCH_CONFIG_COUNTER
00471 # define INTERNAL_CATCH_UNIQUE_NAME( name ) INTERNAL_CATCH_UNIQUE_NAME_LINE( name, __COUNTER__ )
00472 #else
00473 # define INTERNAL_CATCH_UNIQUE_NAME( name ) INTERNAL_CATCH_UNIQUE_NAME_LINE( name, __LINE__ )
00474 #endif
00475
00476 #include <iosfwd>
00477 #include <string>

```

```

00478 #include <cstdint>
00479
00480 // We need a dummy global operator« so we can bring it into Catch namespace later
00481 struct Catch_global_namespace_dummy {};
00482 std::ostream& operator«(std::ostream&, Catch_global_namespace_dummy);
00483
00484 namespace Catch {
00485
00486     struct CaseSensitive { enum Choice {
00487         Yes,
00488         No
00489     }; };
00490
00491     class NonCopyable {
00492     public:
00493         NonCopyable( NonCopyable const& )           = delete;
00494         NonCopyable( NonCopyable && )               = delete;
00495         NonCopyable& operator = ( NonCopyable const& ) = delete;
00496         NonCopyable& operator = ( NonCopyable && )     = delete;
00497
00498     protected:
00499         NonCopyable();
00500         virtual ~NonCopyable();
00501     };
00502
00503     struct SourceLineInfo {
00504     public:
00505         SourceLineInfo() = delete;
00506         SourceLineInfo( char const* _file, std::size_t _line ) noexcept
00507             : file( _file ),
00508               line( _line )
00509         {}
00510
00511         SourceLineInfo( SourceLineInfo const& other )           = default;
00512         SourceLineInfo& operator = ( SourceLineInfo const& )   = default;
00513         SourceLineInfo( SourceLineInfo&& )                     noexcept = default;
00514         SourceLineInfo& operator = ( SourceLineInfo&& )         noexcept = default;
00515
00516         bool empty() const noexcept { return file[0] == '\0'; }
00517         bool operator == ( SourceLineInfo const& other ) const noexcept;
00518         bool operator < ( SourceLineInfo const& other ) const noexcept;
00519
00520     public:
00521         char const* file;
00522         std::size_t line;
00523     };
00524
00525     std::ostream& operator « ( std::ostream& os, SourceLineInfo const& info );
00526
00527     // Bring in operator« from global namespace into Catch namespace
00528     // This is necessary because the overload of operator« above makes
00529     // lookup stop at namespace Catch
00530     using ::operator«;
00531
00532     // Use this in variadic streaming macros to allow
00533     // » +StreamEndStop
00534     // as well as
00535     // » stuff +StreamEndStop
00536     struct StreamEndStop {
00537     public:
00538         std::string operator+() const;
00539     };
00540
00541     template<typename T>
00542     T const& operator + ( T const& value, StreamEndStop ) {
00543         return value;
00544     }
00545
00546 #define CATCH_INTERNAL_LINEINFO \
00547     ::Catch::SourceLineInfo( __FILE__, static_cast<std::size_t>( __LINE__ ) )
00548
00549 // end catch_common.h
00550 namespace Catch {
00551
00552     struct RegistrarForTagAliases {
00553     public:
00554         RegistrarForTagAliases( char const* alias, char const* tag, SourceLineInfo const& lineInfo );
00555     };
00556
00557 } // end namespace Catch
00558
00559 #define CATCH_REGISTER_TAG_ALIAS( alias, spec ) \
00560     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
00561     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
00562     namespace{ Catch::RegistrarForTagAliases INTERNAL_CATCH_UNIQUE_NAME( AutoRegisterTagAlias )( \
00563         alias, spec, CATCH_INTERNAL_LINEINFO ); } \
00564     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
00565 // end catch_tag_alias_autoregistrar.h
00566 // start catch_test_registry.h
00567

```

```

00564 // start catch_interfaces_testcase.h
00565
00566 #include <vector>
00567
00568 namespace Catch {
00569
00570     class TestSpec;
00571
00572     struct ITestInvoker {
00573         virtual void invoke () const = 0;
00574         virtual ~ITestInvoker();
00575     };
00576
00577     class TestCase;
00578     struct IConfig;
00579
00580     struct ITestCaseRegistry {
00581         virtual ~ITestCaseRegistry();
00582         virtual std::vector<TestCase> const& getAllTests() const = 0;
00583         virtual std::vector<TestCase> const& getAllTestsSorted( IConfig const& config ) const = 0;
00584     };
00585
00586     bool isThrowSafe( TestCase const& testCase, IConfig const& config );
00587     bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config );
00588     std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
testSpec, IConfig const& config );
00589     std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config );
00590
00591 }
00592
00593 // end catch_interfaces_testcase.h
00594 // start catch_stringref.h
00595
00596 #include <cstddef>
00597 #include <string>
00598 #include <iosfwd>
00599 #include <cassert>
00600
00601 namespace Catch {
00602
00603     class StringRef {
00604     public:
00605         using size_type = std::size_t;
00606         using const_iterator = const char*;
00607
00608     private:
00609         static constexpr char const* s_empty = "";
00610
00611         char const* m_start = s_empty;
00612         size_type m_size = 0;
00613
00614     public: // construction
00615         constexpr StringRef() noexcept = default;
00616
00617         StringRef( char const* rawChars ) noexcept;
00618
00619         constexpr StringRef( char const* rawChars, size_type size ) noexcept
00620             : m_start( rawChars ),
00621               m_size( size )
00622         {}
00623
00624         StringRef( std::string const& stdString ) noexcept
00625             : m_start( stdString.c_str() ),
00626               m_size( stdString.size() )
00627         {}
00628
00629         explicit operator std::string() const {
00630             return std::string(m_start, m_size);
00631         }
00632
00633     public: // operators
00634         auto operator == ( StringRef const& other ) const noexcept -> bool;
00635         auto operator != ( StringRef const& other ) const noexcept -> bool {
00636             return !(*this == other);
00637         }
00638
00639         auto operator[] ( size_type index ) const noexcept -> char {
00640             assert(index < m_size);
00641             return m_start[index];
00642         }
00643
00644     public: // named queries
00645         constexpr auto empty() const noexcept -> bool {
00646             return m_size == 0;
00647         }
00648         constexpr auto size() const noexcept -> size_type {
00649             return m_size;
00650         }
00651
00652     };

```

```

00653     }
00654
00655     // Returns the current start pointer. If the StringRef is not
00656     // null-terminated, throws std::domain_exception
00657     auto c_str() const -> char const*;
00658
00659     public: // substrings and searches
00660         // Returns a substring of [start, start + length).
00661         // If start + length > size(), then the substring is [start, size()).
00662         // If start > size(), then the substring is empty.
00663         auto substr( size_type start, size_type length ) const noexcept -> StringRef;
00664
00665         // Returns the current start pointer. May not be null-terminated.
00666         auto data() const noexcept -> char const*;
00667
00668         constexpr auto isNullTerminated() const noexcept -> bool {
00669             return m_start[m_size] == '\0';
00670         }
00671
00672     public: // iterators
00673         constexpr const_iterator begin() const { return m_start; }
00674         constexpr const_iterator end() const { return m_start + m_size; }
00675     };
00676
00677     auto operator += ( std::string& lhs, StringRef const& sr ) -> std::string&;
00678     auto operator << ( std::ostream& os, StringRef const& sr ) -> std::ostream&;
00679
00680     constexpr auto operator "" _sr( char const* rawChars, std::size_t size ) noexcept -> StringRef {
00681         return StringRef( rawChars, size );
00682     }
00683 } // namespace Catch
00684
00685 constexpr auto operator "" _catch_sr( char const* rawChars, std::size_t size ) noexcept ->
    Catch::StringRef {
00686     return Catch::StringRef( rawChars, size );
00687 }
00688
00689 // end catch_stringref.h
00690 // start catch_preprocessor.hpp
00691
00692
00693 #define CATCH_RECURSION_LEVEL0(...) __VA_ARGS__
00694 #define CATCH_RECURSION_LEVEL1(...)
00695     CATCH_RECURSION_LEVEL0(CATCH_RECURSION_LEVEL0(CATCH_RECURSION_LEVEL0(__VA_ARGS__)))
00696 #define CATCH_RECURSION_LEVEL2(...)
00697     CATCH_RECURSION_LEVEL1(CATCH_RECURSION_LEVEL1(CATCH_RECURSION_LEVEL1(__VA_ARGS__)))
00698 #define CATCH_RECURSION_LEVEL3(...)
00699     CATCH_RECURSION_LEVEL2(CATCH_RECURSION_LEVEL2(CATCH_RECURSION_LEVEL2(__VA_ARGS__)))
00700 #define CATCH_RECURSION_LEVEL4(...)
00701     CATCH_RECURSION_LEVEL3(CATCH_RECURSION_LEVEL3(CATCH_RECURSION_LEVEL3(__VA_ARGS__)))
00702 #define CATCH_RECURSION_LEVEL5(...)
00703     CATCH_RECURSION_LEVEL4(CATCH_RECURSION_LEVEL4(CATCH_RECURSION_LEVEL4(__VA_ARGS__)))
00704
00705 #ifdef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00706 #define INTERNAL_CATCH_EXPAND_VARGS(...) __VA_ARGS__
00707 // MSVC needs more evaluations
00708 #define CATCH_RECURSION_LEVEL6(...)
00709     CATCH_RECURSION_LEVEL5(CATCH_RECURSION_LEVEL5(CATCH_RECURSION_LEVEL5(__VA_ARGS__)))
00710 #else
00711 #define CATCH_RECURSION_LEVEL6(...)
00712     CATCH_RECURSION_LEVEL4(CATCH_RECURSION_LEVEL4(CATCH_RECURSION_LEVEL4(__VA_ARGS__)))
00713 #endif
00714
00715 #define CATCH_REC_END(...)
00716 #define CATCH_REC_OUT
00717
00718 #define CATCH_EMPTY()
00719 #define CATCH_DEFER(id) id CATCH_EMPTY()
00720
00721 #define CATCH_REC_GET_END2() 0, CATCH_REC_END
00722 #define CATCH_REC_GET_END1(...) CATCH_REC_GET_END2
00723 #define CATCH_REC_GET_END(...) CATCH_REC_GET_END1
00724 #define CATCH_REC_NEXT0(test, next, ...) next CATCH_REC_OUT
00725 #define CATCH_REC_NEXT1(test, next) CATCH_DEFER ( CATCH_REC_NEXT0 ) ( test, next, 0)
00726 #define CATCH_REC_NEXT(test, next) CATCH_REC_NEXT1(CATCH_REC_GET_END test, next)
00727
00728 #define CATCH_REC_LIST0(f, x, peek, ...) , f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1) )
00729     ( f, peek, __VA_ARGS__ )
00730 #define CATCH_REC_LIST1(f, x, peek, ...) , f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST0) )
00731     ( f, peek, __VA_ARGS__ )
00732 #define CATCH_REC_LIST2(f, x, peek, ...) f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1) )
00733     ( f, peek, __VA_ARGS__ )
00734
00735 #define CATCH_REC_LIST0_UD(f, userdata, x, peek, ...) , f(userdata, x) CATCH_DEFER (
00736     CATCH_REC_NEXT(peek, CATCH_REC_LIST1_UD) ) ( f, userdata, peek, __VA_ARGS__ )
00737 #define CATCH_REC_LIST1_UD(f, userdata, x, peek, ...) , f(userdata, x) CATCH_DEFER (
00738     CATCH_REC_NEXT(peek, CATCH_REC_LIST0_UD) ) ( f, userdata, peek, __VA_ARGS__ )

```

```

00728 #define CATCH_REC_LIST2_UD(f, userdata, x, peek, ...) f(userdata, x) CATCH_DEFER (
    CATCH_REC_NEXT(peek, CATCH_REC_LIST1_UD) ) ( f, userdata, peek, __VA_ARGS__ )
00729
00730 // Applies the function macro `f` to each of the remaining parameters, inserts commas between the
    results,
00731 // and passes userdata as the first parameter to each invocation,
00732 // e.g. CATCH_REC_LIST_UD(f, x, a, b, c) evaluates to f(x, a), f(x, b), f(x, c)
00733 #define CATCH_REC_LIST_UD(f, userdata, ...) CATCH_RECURSE(CATCH_REC_LIST2_UD(f, userdata, __VA_ARGS__,
    )()(), ()()(), ()()(), 0))
00734
00735 #define CATCH_REC_LIST(f, ...) CATCH_RECURSE(CATCH_REC_LIST2(f, __VA_ARGS__, ()()(), ()()(), ()()(),
    0))
00736
00737 #define INTERNAL_CATCH_EXPAND1(param) INTERNAL_CATCH_EXPAND2(param)
00738 #define INTERNAL_CATCH_EXPAND2(...) INTERNAL_CATCH_NO## __VA_ARGS__
00739 #define INTERNAL_CATCH_DEF(...) INTERNAL_CATCH_DEF __VA_ARGS__
00740 #define INTERNAL_CATCH_NOINTERNAL_CATCH_DEF
00741 #define INTERNAL_CATCH_STRINGIZE(...) INTERNAL_CATCH_STRINGIZE2(__VA_ARGS__)
00742 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00743 #define INTERNAL_CATCH_STRINGIZE2(...) #__VA_ARGS__
00744 #define INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS(param)
    INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_REMOVE_PARENS(param))
00745 #else
00746 // MSVC is adding extra space and needs another indirection to expand
    INTERNAL_CATCH_NOINTERNAL_CATCH_DEF
00747 #define INTERNAL_CATCH_STRINGIZE2(...) INTERNAL_CATCH_STRINGIZE3(__VA_ARGS__)
00748 #define INTERNAL_CATCH_STRINGIZE3(...) #__VA_ARGS__
00749 #define INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS(param)
    (INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_REMOVE_PARENS(param)) + 1)
00750 #endif
00751
00752 #define INTERNAL_CATCH_MAKE_NAMESPACE2(...) ns_##__VA_ARGS__
00753 #define INTERNAL_CATCH_MAKE_NAMESPACE(name) INTERNAL_CATCH_MAKE_NAMESPACE2(name)
00754
00755 #define INTERNAL_CATCH_REMOVE_PARENS(...) INTERNAL_CATCH_EXPAND1(INTERNAL_CATCH_DEF __VA_ARGS__)
00756
00757 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
00758 #define INTERNAL_CATCH_MAKE_TYPE_LIST2(...)
    decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS_GEN(__VA_ARGS__)>())
00759 #define INTERNAL_CATCH_MAKE_TYPE_LIST(...)
    INTERNAL_CATCH_MAKE_TYPE_LIST2(INTERNAL_CATCH_REMOVE_PARENS(__VA_ARGS__))
00760 #else
00761 #define INTERNAL_CATCH_MAKE_TYPE_LIST2(...)
    INTERNAL_CATCH_EXPAND_VARGS(decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS_GEN(__VA_ARGS__)>()))
00762 #define INTERNAL_CATCH_MAKE_TYPE_LIST(...)
    INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_MAKE_TYPE_LIST2(INTERNAL_CATCH_REMOVE_PARENS(__VA_ARGS__)))
00763 #endif
00764
00765 #define INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(...) \
00766     CATCH_REC_LIST(INTERNAL_CATCH_MAKE_TYPE_LIST, __VA_ARGS__)
00767
00768 #define INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_0) INTERNAL_CATCH_REMOVE_PARENS(_0)
00769 #define INTERNAL_CATCH_REMOVE_PARENS_2_ARG(_0, _1) INTERNAL_CATCH_REMOVE_PARENS(_0),
    INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_1)
00770 #define INTERNAL_CATCH_REMOVE_PARENS_3_ARG(_0, _1, _2) INTERNAL_CATCH_REMOVE_PARENS(_0),
    INTERNAL_CATCH_REMOVE_PARENS_2_ARG(_1, _2)
00771 #define INTERNAL_CATCH_REMOVE_PARENS_4_ARG(_0, _1, _2, _3) INTERNAL_CATCH_REMOVE_PARENS(_0),
    INTERNAL_CATCH_REMOVE_PARENS_3_ARG(_1, _2, _3)
00772 #define INTERNAL_CATCH_REMOVE_PARENS_5_ARG(_0, _1, _2, _3, _4) INTERNAL_CATCH_REMOVE_PARENS(_0),
    INTERNAL_CATCH_REMOVE_PARENS_4_ARG(_1, _2, _3, _4)
00773 #define INTERNAL_CATCH_REMOVE_PARENS_6_ARG(_0, _1, _2, _3, _4, _5) INTERNAL_CATCH_REMOVE_PARENS(_0),
    INTERNAL_CATCH_REMOVE_PARENS_5_ARG(_1, _2, _3, _4, _5)
00774 #define INTERNAL_CATCH_REMOVE_PARENS_7_ARG(_0, _1, _2, _3, _4, _5, _6)
    INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_6_ARG(_1, _2, _3, _4, _5, _6)
00775 #define INTERNAL_CATCH_REMOVE_PARENS_8_ARG(_0, _1, _2, _3, _4, _5, _6, _7)
    INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_7_ARG(_1, _2, _3, _4, _5, _6, _7)
00776 #define INTERNAL_CATCH_REMOVE_PARENS_9_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8)
    INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_8_ARG(_1, _2, _3, _4, _5, _6, _7, _8)
00777 #define INTERNAL_CATCH_REMOVE_PARENS_10_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9)
    INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_9_ARG(_1, _2, _3, _4, _5, _6, _7, _8,
    _9)
00778 #define INTERNAL_CATCH_REMOVE_PARENS_11_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10)
    INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_10_ARG(_1, _2, _3, _4, _5, _6, _7, _8,
    _9, _10)
00779
00780 #define INTERNAL_CATCH_VA_NARGS_IMPL(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, N, ...) N
00781
00782 #define INTERNAL_CATCH_TYPE_GEN \
00783     template<typename...> struct TypeList {}; \
00784     template<typename...Ts> \
00785         constexpr auto get_wrapper() noexcept -> TypeList<Ts...> { return {}; } \
00786     template<template<typename...> class...> struct TemplateTypeList {}; \
00787     template<template<typename...> class...Cs> \
00788         constexpr auto get_wrapper() noexcept -> TemplateTypeList<Cs...> { return {}; } \
00789     template<typename...> \
00790     struct append; \
00791     template<typename...> \

```



```

00792     struct rewrap;\
00793     template<template<typename...> class, typename...>\
00794     struct create;\
00795     template<template<typename...> class, typename>\
00796     struct convert;\
00797     \
00798     template<typename T> \
00799     struct append<T> { using type = T; };\
00800     template<template<typename...> class L1, typename...E1, template<typename...> class L2,
typename...E2, typename...Rest>\
00801     struct append<L1<E1...>, L2<E2...>, Rest...> { using type = typename append<L1<E1...,E2...>,
Rest...>::type; };\
00802     template<template<typename...> class L1, typename...E1, typename...Rest>\
00803     struct append<L1<E1...>, TypeList<mpl_::na>, Rest...> { using type = L1<E1...>; };\
00804     \
00805     template<template<typename...> class Container, template<typename...> class List,
typename...elems>\
00806     struct rewrap<TemplateTypeList<Container>, List<elems...> { using type =
TypeList<Container<elems...>; };\
00807     template<template<typename...> class Container, template<typename...> class List, class...Elems,
typename...Elements>\
00808     struct rewrap<TemplateTypeList<Container>, List<Elems...>, Elements...> { using type = typename
append<TypeList<Container<Elems...>, typename rewrap<TemplateTypeList<Container>,
Elements...>::type>::type; };\
00809     \
00810     template<template <typename...> class Final, template<typename...> class...Containers,
typename...Types>\
00811     struct create<Final, TemplateTypeList<Containers...>, TypeList<Types...> { using type = typename
append<Final<>, typename rewrap<TemplateTypeList<Containers>, Types...>::type...>::type; };\
00812     template<template <typename...> class Final, template <typename...> class List, typename...Ts>\
00813     struct convert<Final, List<Ts...> { using type = typename append<Final<>,TypeList<Ts>...>::type;
};\
00814
00815 #define INTERNAL_CATCH_NTTP_1(signature, ...)\
00816     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> struct Nttp{};\
00817     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00818     constexpr auto get_wrapper() noexcept -> Nttp<__VA_ARGS__> { return {}; } \
00819     template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class...> struct
NttpTemplateTypeList{};\
00820     template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class...Cs>\
00821     constexpr auto get_wrapper() noexcept -> NttpTemplateTypeList<Cs...> { return {}; } \
00822     \
00823     template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class Container,
template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class List,
INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00824     struct rewrap<NttpTemplateTypeList<Container>, List<__VA_ARGS__> { using type =
TypeList<Container<__VA_ARGS__>; };\
00825     template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class Container,
template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class List, INTERNAL_CATCH_REMOVE_PARENS(signature),
typename...Elements>\
00826     struct rewrap<NttpTemplateTypeList<Container>, List<__VA_ARGS__>, Elements...> { using type =
typename append<TypeList<Container<__VA_ARGS__>, typename rewrap<NttpTemplateTypeList<Container>,
Elements...>::type>::type; };\
00827     template<template <typename...> class Final, template<INTERNAL_CATCH_REMOVE_PARENS(signature)>
class...Containers, typename...Types>\
00828     struct create<Final, NttpTemplateTypeList<Containers...>, TypeList<Types...> { using type =
typename append<Final<>, typename rewrap<NttpTemplateTypeList<Containers>, Types...>::type...>::type;
};\
00829
00830 #define INTERNAL_CATCH_DECLARE_SIG_TEST0(TestName)
00831 #define INTERNAL_CATCH_DECLARE_SIG_TEST1(TestName, signature)\
00832     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00833     static void TestName()
00834 #define INTERNAL_CATCH_DECLARE_SIG_TEST_X(TestName, signature, ...)\
00835     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00836     static void TestName()
00837
00838 #define INTERNAL_CATCH_DEFINE_SIG_TEST0(TestName)
00839 #define INTERNAL_CATCH_DEFINE_SIG_TEST1(TestName, signature)\
00840     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00841     static void TestName()
00842 #define INTERNAL_CATCH_DEFINE_SIG_TEST_X(TestName, signature,...)\
00843     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00844     static void TestName()
00845
00846 #define INTERNAL_CATCH_NTTP_REGISTER0(TestFunc, signature)\
00847     template<typename Type>\
00848     void reg_test(TypeList<Type>, Catch::NameAndTags nameAndTags)\
00849     {\
00850         Catch::AutoReg( Catch::makeTestInvoker(&TestFunc<Type>), CATCH_INTERNAL_LINEINFO,
Catch::StringRef(), nameAndTags);\
00851     }
00852
00853 #define INTERNAL_CATCH_NTTP_REGISTER(TestFunc, signature, ...)\
00854     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
00855     void reg_test(Nttp<__VA_ARGS__>, Catch::NameAndTags nameAndTags)\
00856     {\

```



```
00857 Catch::AutoReg( Catch::makeTestInvoker(&TestFunc<__VA_ARGS__>()), CATCH_INTERNAL_LINEINFO,  
Catch::StringRef(), nameAndTags);\n  
00858 }\n00859 \n  
00860 #define INTERNAL_CATCH_NTTP_REGISTER_METHOD0(TestName, signature, ...)\n00861     template<typename Type>\n00862     void reg_test(TypeList<Type>, Catch::StringRef className, Catch::NameAndTags nameAndTags)\n00863     {\n00864         Catch::AutoReg( Catch::makeTestInvoker(&TestName<Type>::test), CATCH_INTERNAL_LINEINFO,\nclassName, nameAndTags);\n00865     }\n00866 \n  
00867 #define INTERNAL_CATCH_NTTP_REGISTER_METHOD(TestName, signature, ...)\n00868     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\n00869     void reg_test(Nttp<__VA_ARGS__>, Catch::StringRef className, Catch::NameAndTags nameAndTags)\n00870     {\n00871         Catch::AutoReg( Catch::makeTestInvoker(&TestName<__VA_ARGS__>::test), CATCH_INTERNAL_LINEINFO,\nclassName, nameAndTags);\n00872     }\n00873 \n  
00874 #define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0(TestName, ClassName)\n00875 #define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1(TestName, ClassName, signature)\n00876     template<typename TestType> \n00877     struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName)<TestType> { \n00878         void test();\n00879     }\n00880 \n  
00881 #define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X(TestName, ClassName, signature, ...)\n00882     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> \n00883     struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName)<__VA_ARGS__> { \n00884         void test();\n00885     }\n00886 \n  
00887 #define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0(TestName)\n00888 #define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1(TestName, signature)\n00889     template<typename TestType> \n00890     void INTERNAL_CATCH_MAKE_NAMESPACE(TestName)::TestName<TestType>::test()\n00891 #define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X(TestName, signature, ...)\n00892     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> \n00893     void INTERNAL_CATCH_MAKE_NAMESPACE(TestName)::TestName<__VA_ARGS__>::test()\n00894 \n  
00895 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR\n00896 #define INTERNAL_CATCH_NTTP_0\n00897 #define INTERNAL_CATCH_NTTP_GEN(...) INTERNAL_CATCH_VA_NARGS_IMPL(__VA_ARGS__,\nINTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_1(__VA_ARGS__),\nINTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_1(__VA_ARGS__),\nINTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_1(\n__VA_ARGS__), INTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_1(\n__VA_ARGS__),INTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_0)\n00898 #define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, ...) INTERNAL_CATCH_VA_NARGS_IMPL("dummy",\n__VA_ARGS__, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,\nINTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,\nINTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,\nINTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,\nINTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0)(TestName, __VA_ARGS__)\n00899 #define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD(TestName, ClassName, ...) INTERNAL_CATCH_VA_NARGS_IMPL(\n"dummy", __VA_ARGS__,\nINTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,\nINTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,\nINTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,\nINTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD,\nINTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0)(TestName, ClassName,\n__VA_ARGS__)\n00900 #define INTERNAL_CATCH_NTTP_REG_METHOD_GEN(TestName, ...) INTERNAL_CATCH_VA_NARGS_IMPL("dummy",\n__VA_ARGS__, INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,\nINTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,\nINTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,\nINTERNAL_CATCH_NTTP_REGISTER_METHOD,INTERNAL_CATCH_NTTP_REGISTER_METHOD0,\nINTERNAL_CATCH_NTTP_REGISTER_METHOD0)(TestName, __VA_ARGS__)\n00901 #define INTERNAL_CATCH_NTTP_REG_GEN(TestFunc, ...) INTERNAL_CATCH_VA_NARGS_IMPL("dummy", __VA_ARGS__,\nINTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,\nINTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,\nINTERNAL_CATCH_NTTP_REGISTER0, INTERNAL_CATCH_NTTP_REGISTER0)(TestFunc, __VA_ARGS__)\n00902 #define INTERNAL_CATCH_DEFINE_SIG_TEST(TestName, ...) INTERNAL_CATCH_VA_NARGS_IMPL("dummy",\n__VA_ARGS__, INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X,\nINTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X,\nINTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST1,\nINTERNAL_CATCH_DEFINE_SIG_TEST0)(TestName, __VA_ARGS__)\n00903 #define INTERNAL_CATCH_DECLARE_SIG_TEST(TestName, ...) INTERNAL_CATCH_VA_NARGS_IMPL("dummy",\n__VA_ARGS__, INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,\nINTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,\nINTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,\nINTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,\nINTERNAL_CATCH_DECLARE_SIG_TEST1, INTERNAL_CATCH_DECLARE_SIG_TEST0)(TestName, __VA_ARGS__)
```

Generated by Doxygen

```

    U...»>;
00946 #else
00947     // Keep ::type here because we still support C++11
00948     template <typename Func, typename... U>
00949     using FunctionReturnType = typename std::remove_reference<typename std::remove_cv<typename
std::result_of<Func(U...)>::type>::type>::type;
00950 #endif
00951
00952 } // namespace Catch
00953
00954 namespace mpl_{
00955     struct na;
00956 }
00957
00958 // end catch_meta.hpp
00959 namespace Catch {
00960
00961     template<typename C>
00962     class TestInvokerAsMethod : public ITestInvoker {
00963     public:
00964         TestInvokerAsMethod( void (C::*testAsMethod)() ) noexcept : m_testAsMethod( testAsMethod ) {}
00965
00966         void invoke() const override {
00967             C obj;
00968             (obj.*m_testAsMethod)();
00969         }
00970     };
00971
00972     auto makeTestInvoker( void(*testAsFunction)() ) noexcept -> ITestInvoker*;
00973
00974     template<typename C>
00975     auto makeTestInvoker( void (C::*testAsMethod)() ) noexcept -> ITestInvoker* {
00976         return new(std::nothrow) TestInvokerAsMethod<C>( testAsMethod );
00977     }
00978
00979     struct NameAndTags {
00980     public:
00981         NameAndTags( StringRef const& name_ = StringRef(), StringRef const& tags_ = StringRef() )
00982         noexcept;
00983         StringRef name;
00984         StringRef tags;
00985     };
00986
00987     struct AutoReg : NonCopyable {
00988     public:
00989         AutoReg( ITestInvoker* invoker, SourceLineInfo const& lineInfo, StringRef const&
classOrMethod, NameAndTags const& nameAndTags ) noexcept;
00990         ~AutoReg();
00991     };
00992
00993     #if defined(CATCH_CONFIG_DISABLE)
00994     #define INTERNAL_CATCH_TESTCASE_NO_REGISTRATION( TestName, ... ) \
00995         static void TestName()
00996     #define INTERNAL_CATCH_TESTCASE_METHOD_NO_REGISTRATION( TestName, ClassName, ... ) \
00997         namespace{ \
00998             struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName) { \
00999                 void test(); \
01000             }; \
01001         } \
01002         void TestName::test()
01003     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION_2( TestName, TestFunc, Name, Tags, \
Signature, ... ) \
01004         INTERNAL_CATCH_DEFINE_SIG_TEST(TestFunc, INTERNAL_CATCH_REMOVE_PARENS(Signature))
01005     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2( TestNameClass, TestName, \
ClassName, Name, Tags, Signature, ... ) \
01006         namespace{ \
01007             namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName) { \
01008                 INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD(TestName, ClassName, \
INTERNAL_CATCH_REMOVE_PARENS(Signature)); \
01009             } \
01010         } \
01011         INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, INTERNAL_CATCH_REMOVE_PARENS(Signature))
01012
01013     #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01014     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(Name, Tags, ...) \
01015         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, typename TestType, __VA_ARGS__ )
01016     #else
01017     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(Name, Tags, ...) \
01018         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, typename TestType, __VA_ARGS__ ) )
01019     #endif
01020
01021     #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR

```

```

01022     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(Name, Tags, Signature, ...) \
01023     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__ )
01024     #else
01025     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(Name, Tags, Signature, ...) \
01026     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__ ) )
01027     #endif
01028
01029     #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01030     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION( ClassName, Name, Tags,... )
\
01031     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__ )
01032     #else
01033     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION( ClassName, Name, Tags,... )
\
01034     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__ ) )
01035     #endif
01036
01037     #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01038     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION( ClassName, Name, Tags,
Signature, ... ) \
01039     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__ )
01040     #else
01041     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION( ClassName, Name, Tags,
Signature, ... ) \
01042     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__ ) )
01043     #endif
01044 #endif
01045
01047 #define INTERNAL_CATCH_TESTCASE2( TestName, ... ) \
01048     static void TestName(); \
01049     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01050     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01051     namespace{ Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker(
&TestName ), CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ __VA_ARGS__ } ); } /*
NOLINT */ \
01052     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01053     static void TestName()
01054 #define INTERNAL_CATCH_TESTCASE( ... ) \
01055     INTERNAL_CATCH_TESTCASE2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ), __VA_ARGS__ )
01056
01058 #define INTERNAL_CATCH_METHOD_AS_TEST_CASE( QualifiedMethod, ... ) \
01059     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01060     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01061     namespace{ Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker(
&QualifiedMethod ), CATCH_INTERNAL_LINEINFO, "&" #QualifiedMethod, Catch::NameAndTags{ __VA_ARGS__ }
); } /* NOLINT */ \
01062     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
01063
01065 #define INTERNAL_CATCH_TEST_CASE_METHOD2( TestName, ClassName, ... )\
01066     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01067     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01068     namespace{ \
01069         struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName) { \
01070             void test(); \
01071         }; \
01072         Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker(
&TestName::test ), CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ __VA_ARGS__ } ); } /* NOLINT
*/ \
01073     } \
01074     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01075     void TestName::test()
01076 #define INTERNAL_CATCH_TEST_CASE_METHOD( ClassName, ... ) \
01077     INTERNAL_CATCH_TEST_CASE_METHOD2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ), ClassName,
__VA_ARGS__ )
01078
01080 #define INTERNAL_CATCH_REGISTER_TESTCASE( Function, ... ) \
01081     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01082     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01083     Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker( Function
), CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ __VA_ARGS__ } ); } /* NOLINT */ \
01084     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
01085
01087 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( TestName, TestFunc, Name, Tags, Signature, ... )\

```

```

01088     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01089     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01090     CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
01091     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01092     INTERNAL_CATCH_DECLARE_SIG_TEST(TestFunc, INTERNAL_CATCH_REMOVE_PARENS(Signature));\
01093     namespace {\
01094     namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName){\
01095         INTERNAL_CATCH_TYPE_GEN\
01096         INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS(Signature))\
01097         INTERNAL_CATCH_NTTP_REG_GEN(TestFunc, INTERNAL_CATCH_REMOVE_PARENS(Signature))\
01098         template<typename...Types> \
01099         struct TestName{\
01100             TestName(){\
01101                 int index = 0; \
01102                 constexpr char const* tmpl_types[] = \
01103 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, __VA_ARGS__)};\
01104                 using expander = int[];\
01105                 (void)expander{(reg_test(Types{}, Catch::NameAndTags{ Name " - " + \
01106 std::string(tmpl_types[index]), Tags } ), index++)... };/* NOLINT */ \
01107             }\
01108             static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
01109                 TestName<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(__VA_ARGS__)>();\
01110                 return 0;\
01111             }\
01112             }\
01113     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01114     INTERNAL_CATCH_DEFINE_SIG_TEST(TestFunc, INTERNAL_CATCH_REMOVE_PARENS(Signature))
01115
01116 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01117 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE(Name, Tags, ...) \
01118     INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME( \
01119 C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( \
01120 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, __VA_ARGS__ )
01121 #else
01122 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE(Name, Tags, ...) \
01123     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME( \
01124 C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( \
01125 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, __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 std::string(tmpl_types[index / num_types]) + "<" + std::string(types_list[index % num_types]) + ">", \
01163 Tags } ), index++)... };/* NOLINT */ \
01164             } \
01165             static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){ \
01166                 using TestInit = typename create<TestName, \
01167 decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS(TmplTypes)>())>, \
01168 TypeList<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(INTERNAL_CATCH_REMOVE_PARENS(TypesList))>::type; \
01169                 TestInit t; \
01170                 t.reg_tests(); \
01171             } \
01172         } \
01173     } \
01174     static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){ \
01175         using TestInit = typename create<TestName, \
01176 decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS(TmplTypes)>())>, \
01177 TypeList<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(INTERNAL_CATCH_REMOVE_PARENS(TypesList))>::type; \
01178         TestInit t; \
01179         t.reg_tests(); \
01180     } \
01181 }

```

```

01157         return 0;
01158     }();
01159 }
01160 }
01161 CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
01162 template<typename TestType>
01163 static void TestFuncName()
01164
01165 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01166 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE(Name, Tags, ...) \
01167     INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(INTERNAL_CATCH_UNIQUE_NAME(
01168         C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
01169         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__)
01170 #else
01171 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE(Name, Tags, ...) \
01172     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(
01173         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(
01174         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__ ) )
01175 #endif
01176
01177 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01178 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(Name, Tags, Signature, ...) \
01179     INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(INTERNAL_CATCH_UNIQUE_NAME(
01180         C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
01181         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__)
01182 #else
01183 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(Name, Tags, Signature, ...) \
01184     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(
01185         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(
01186         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__ ) )
01187 #endif
01188
01189 #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_2( testName, testFunc, name, tags, tplList ) \
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(tplList)) + " - " + std::to_string(index), Tags } ), index++)... \
01205                     }; /* NOLINT */ \
01206                 } \
01207             }; \
01208         } \
01209         static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = []() { \
01210             using TestInit = typename convert<TestName, tplList>::type; \
01211             TestInit t; \
01212             t.reg_tests(); \
01213             return 0; \
01214         }(); \
01215     } \
01216     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
01217 template<typename TestType>
01218 static void TestFunc()
01219
01220 #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE(Name, Tags, tplList) \
01221     INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME(
01222         C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
01223         C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, tplList )
01224
01225 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2( TestNameClass, testName, className, name, tags, \
01226     Signature, ... ) \
01227     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01228     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01229     CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
01230     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01231     namespace { \
01232         namespace INTERNAL_CATCH_MAKE_NAMESPACE( testName ) { \
01233             INTERNAL_CATCH_TYPE_GEN \
01234             INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS( Signature )) \
01235             INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD( TestName, className, \
01236                 INTERNAL_CATCH_REMOVE_PARENS( Signature )) \
01237             INTERNAL_CATCH_NTTP_REG_METHOD_GEN( TestName, INTERNAL_CATCH_REMOVE_PARENS( Signature )) \
01238             template<typename... Types> \
01239             struct TestNameClass { \
01240                 TestNameClass() { \
01241                     int index = 0; \
01242                     constexpr char const* tpl_types[] = \
01243 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, __VA_ARGS__)}; \
01244                     using expander = int[]; \

```

```

01228         (void)expander{(reg_test(Types{}, #ClassName, Catch::NameAndTags{ Name " - " +
std::string(tmpl_types[index]), Tags } ), index++)... };/* NOLINT */ \
01229     }\
01230     };\
01231     static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
01232         TestNameClass<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(__VA_ARGS__)>();\
01233         return 0;\
01234     }();\
01235     }\
01236     };\
01237     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01238     INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, INTERNAL_CATCH_REMOVE_PARENS(Signature))
01239
01240 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01241 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( ClassName, Name, Tags,... ) \
01242     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_ ), 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__ )
01243 #else
01244 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( ClassName, Name, Tags,... ) \
01245     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_ ),
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__ ) )
01246 #endif
01247
01248 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01249 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature, ... ) \
01250     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_ ), 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__ )
01251 #else
01252 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature, ... ) \
01253     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_ ),
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__ ) )
01254 #endif
01255
01256 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2( TestNameClass, TestName, ClassName, Name,
Tags, Signature, TmplTypes, TypesList ) \
01257     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01258     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01259     CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
01260     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01261     template<typename TestType> \
01262     struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName <TestType>) { \
01263         void test(); \
01264     }; \
01265     namespace { \
01266         namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestNameClass ) { \
01267             INTERNAL_CATCH_TYPE_GEN \
01268             INTERNAL_CATCH_NTTP_GEN( INTERNAL_CATCH_REMOVE_PARENS(Signature) ) \
01269             template<typename... Types> \
01270             struct TestNameClass { \
01271                 void reg_tests() { \
01272                     int index = 0; \
01273                     using expander = int[]; \
01274                     constexpr char const* tmpl_types[] = \
01275 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TmplTypes))}; \
01276                     constexpr char const* types_list[] = \
01277 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TypesList))}; \
01278                     constexpr auto num_types = sizeof(types_list) / sizeof(types_list[0]); \
01279                     (void)expander{ Catch::AutoReg( Catch::makeTestInvoker( &TestName<Types>::test ), \
CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ Name " - " + std::string(tmpl_types[index / \
num_types]) + "<" + std::string(types_list[index % num_types]) + ">", Tags } ), index++)... };/* \
NOLINT */ \
01280                 } \
01281             }; \
01282             static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){ \
01283                 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; \
01284                 TestInit t; \
01285                 t.reg_tests(); \
01286                 return 0; \
01287             }(); \
01288         } \
01289     } \
01290     template<typename TestType> \
01291     void TestName<TestType>::test()
01292
01293 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01294 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( ClassName, Name, Tags, ... ) \
01295     INTERNAL_CATCH_TEMPLATE_PRODUCT_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_ ), 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, Name, Tags, typename T, __VA_ARGS__ )
01295 #else
01296 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( ClassName, Name, Tags, ... )\
01297     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_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_ ), 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, Name, Tags, typename T, __VA_ARGS__ ) )
01298 #endif
01299
01300 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
01301 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature, ... )\
01302     INTERNAL_CATCH_TEMPLATE_PRODUCT_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_ ), 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, Name, Tags, Signature, __VA_ARGS__ )
01303 #else
01304 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature, ... )\
01305     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_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_ ), 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, Name, Tags, Signature, __VA_ARGS__ ) )
01306 #endif
01307
01308 #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_2( TestNameClass, TestName, ClassName, Name,
    Tags, TmplList) \
01309     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
01310     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
01311     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
01312     template<typename TestType> \
01313     struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName <TestType>) { \
01314         void test();\
01315     };\
01316     namespace {\
01317         namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName){ \
01318             INTERNAL_CATCH_TYPE_GEN\
01319             template<typename...Types>\
01320             struct TestNameClass{\
01321                 void reg_tests(){\
01322                     int index = 0;\
01323                     using expander = int[];\
01324                     (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestName<Types>::test ),
        CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ Name " - " +
        std::string(INTERNAL_CATCH_STRINGIZE(TmplList)) + " - " + std::to_string(index), Tags } ), index++)...
        };/* NOLINT */ \
01325             }}\
01326         };\
01327         static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
01328             using TestInit = typename convert<TestNameClass, TmplList>::type;\
01329             TestInit t;\
01330             t.reg_tests();\
01331             return 0;\
01332         }(); \
01333     };\
01334     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
01335     template<typename TestType> \
01336     void TestName<TestType>::test()
01337
01338 #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD(ClassName, Name, Tags, TmplList) \
01339     INTERNAL_CATCH_TEMPLATE_LIST_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_ ), 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, Name, Tags, TmplList )
01340
01341 // end catch_test_registry.h
01342 // start catch_capture.hpp
01343
01344 // start catch_assertionhandler.h
01345
01346 // start catch_assertioninfo.h
01347
01348 // start catch_result_type.h
01349
01350 namespace Catch {
01351
01352     // ResultWas::OfType enum
01353     struct ResultWas { enum OfType {
01354         Unknown = -1,
01355         Ok = 0,
01356         Info = 1,
01357         Warning = 2,
01358
01359         FailureBit = 0x10,
01360
01361         ExpressionFailed = FailureBit | 1,
01362         ExplicitFailure = FailureBit | 2,
01363
01364         Exception = 0x100 | FailureBit,
01365
01366         ThrewException = Exception | 1,
01367         DidntThrowException = Exception | 2,
01368     };

```



```

01369         FatalErrorCondition = 0x200 | FailureBit
01370     }; };
01371
01372 bool isOk( ResultWas::OfType resultType );
01373 bool isJustInfo( int flags );
01374
01375 // ResultDisposition::Flags enum
01376 struct ResultDisposition { enum Flags {
01377     Normal = 0x01,
01378
01379     ContinueOnFailure = 0x02, // Failures fail test, but execution continues
01380     FalseTest = 0x04, // Prefix expression with !
01381     SuppressFail = 0x08 // Failures are reported but do not fail the test
01382 }; };
01383
01384 ResultDisposition::Flags operator | ( ResultDisposition::Flags lhs, ResultDisposition::Flags rhs
01385 );
01386
01387 bool shouldContinueOnFailure( int flags );
01388 inline bool isFalseTest( int flags ) { return ( flags & ResultDisposition::FalseTest ) != 0; }
01389 bool shouldSuppressFailure( int flags );
01390
01391 } // end namespace Catch
01392
01393 // end catch_result_type.h
01394 namespace Catch {
01395     struct AssertionInfo
01396     {
01397         StringRef macroName;
01398         SourceLineInfo lineInfo;
01399         StringRef capturedExpression;
01400         ResultDisposition::Flags resultDisposition;
01401
01402         // We want to delete this constructor but a compiler bug in 4.8 means
01403         // the struct is then treated as non-aggregate
01404         //AssertionInfo() = delete;
01405     };
01406
01407 } // end namespace Catch
01408
01409 // end catch_assertioninfo.h
01410 // start catch_decomposer.h
01411
01412 // start catch_tostring.h
01413
01414 #include <vector>
01415 #include <cstdint>
01416 #include <type_traits>
01417 #include <string>
01418 // start catch_stream.h
01419
01420 #include <iosfwd>
01421 #include <cstdint>
01422 #include <ostream>
01423
01424 namespace Catch {
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         ReusableStringStream();
01441         ~ReusableStringStream();
01442
01443         auto str() const -> std::string;
01444
01445         template<typename T>
01446         auto operator << ( T const& value ) -> ReusableStringStream& {
01447             *m_oss << value;
01448             return *this;
01449         }
01450         auto get() -> std::ostream& { return *m_oss; }
01451     };

```

```

01455     };
01456 }
01457
01458 // end catch_stream.h
01459 // start catch_interfaces_enum_values_registry.h
01460
01461 #include <vector>
01462
01463 namespace Catch {
01464
01465     namespace Detail {
01466         struct EnumInfo {
01467             StringRef m_name;
01468             std::vector<std::pair<int, StringRef>> m_values;
01469
01470             ~EnumInfo();
01471
01472             StringRef lookup( int value ) const;
01473         };
01474     } // namespace Detail
01475
01476     struct IMutableEnumValuesRegistry {
01477         virtual ~IMutableEnumValuesRegistry();
01478
01479         virtual Detail::EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums,
01480             std::vector<int> const& values ) = 0;
01481
01482         template<typename E>
01483         Detail::EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums,
01484             std::initializer_list<E> values ) {
01485             static_assert(sizeof(int) >= sizeof(E), "Cannot serialize enum to int");
01486             std::vector<int> intValues;
01487             intValues.reserve( values.size() );
01488             for( auto enumValue : values )
01489                 intValues.push_back( static_cast<int>( enumValue ) );
01490             return registerEnum( enumName, allEnums, intValues );
01491         }
01492     };
01493
01494 } // Catch
01495
01496 // end catch_interfaces_enum_values_registry.h
01497
01498 #ifdef CATCH_CONFIG_CPP17_STRING_VIEW
01499 #include <string_view>
01500 #endif
01501
01502 #ifdef __OBJC__
01503 // start catch_objc_arc.hpp
01504
01505 #import <Foundation/Foundation.h>
01506
01507 #ifdef __has_feature
01508 #define CATCH_ARC_ENABLED __has_feature(objc_arc)
01509 #else
01510 #define CATCH_ARC_ENABLED 0
01511 #endif
01512
01513 void arcSafeRelease( NSObject* obj );
01514 id performOptionalSelector( id obj, SEL sel );
01515
01516 #if !CATCH_ARC_ENABLED
01517 inline void arcSafeRelease( NSObject* obj ) {
01518     [obj release];
01519 }
01520 inline id performOptionalSelector( id obj, SEL sel ) {
01521     if( [obj respondsToSelector: sel] )
01522         return [obj performSelector: sel];
01523     return nil;
01524 }
01525 #define CATCH_UNSAFE_UNRETAINED
01526 #define CATCH_ARC_STRONG
01527 #else
01528 inline void arcSafeRelease( NSObject* ){}
01529 inline id performOptionalSelector( id obj, SEL sel ) {
01530     #ifdef __clang__
01531     #pragma clang diagnostic push
01532     #pragma clang diagnostic ignored "-Warc-performSelector-leaks"
01533     if( [obj respondsToSelector: sel] )
01534         return [obj performSelector: sel];
01535     #endif
01536     return nil;
01537 }
01538 #define CATCH_UNSAFE_UNRETAINED __unsafe_unretained
01539 #endif

```

```

01540 #define CATCH_ARC_STRONG __strong
01541 #endif
01542
01543 // end catch_objc_arc.hpp
01544 #endif
01545
01546 #ifndef _MSC_VER
01547 #pragma warning(push)
01548 #pragma warning(disable:4180) // We attempt to stream a function (address) by const&, which MSVC
    complains about but is harmless
01549 #endif
01550
01551 namespace Catch {
01552     namespace Detail {
01553
01554         extern const std::string unprintableString;
01555
01556         std::string rawMemoryToString( const void *object, std::size_t size );
01557
01558         template<typename T>
01559         std::string rawMemoryToString( const T& object ) {
01560             return rawMemoryToString( &object, sizeof(object) );
01561         }
01562
01563         template<typename T>
01564         class IsStreamInsertable {
01565             template<typename Stream, typename U>
01566             static auto test(int)
01567             -> decltype(std::declval<Stream&>() « std::declval<U>(), std::true_type());
01568
01569             template<typename, typename>
01570             static auto test(...) -> std::false_type;
01571
01572         public:
01573             static const bool value = decltype(test<std::ostream, const T&>(0))::value;
01574         };
01575
01576         template<typename E>
01577         std::string convertUnknownEnumToString( E e );
01578
01579         template<typename T>
01580         typename std::enable_if<
01581             !std::is_enum<T>::value && !std::is_base_of<std::exception, T>::value,
01582             std::string>::type convertUnstreamable( T const& ) {
01583             return Detail::unprintableString;
01584         }
01585
01586         template<typename T>
01587         typename std::enable_if<
01588             !std::is_enum<T>::value && std::is_base_of<std::exception, T>::value,
01589             std::string>::type convertUnstreamable( T const& ex ) {
01590             return ex.what();
01591         }
01592
01593         template<typename T>
01594         typename std::enable_if<
01595             std::is_enum<T>::value
01596             , std::string>::type convertUnstreamable( T const& value ) {
01597             return convertUnknownEnumToString( value );
01598         }
01599
01600 #if defined(_MANAGED)
01601         template<typename T>
01602         std::string clrReferenceToString( T^ ref ) {
01603             if (ref == nullptr)
01604                 return std::string("null");
01605             auto bytes = System::Text::Encoding::UTF8->GetBytes(ref->ToString());
01606             cli::pin_ptr<System::Byte> p = &bytes[0];
01607             return std::string(reinterpret_cast<char const *>(p), bytes->Length);
01608         }
01609 #endif
01610
01611     } // namespace Detail
01612
01613     // If we decide for C++14, change these to enable_if_ts
01614     template <typename T, typename = void>
01615     struct StringMaker {
01616         template <typename Fake = T>
01617         static
01618         typename std::enable_if<::Catch::Detail::IsStreamInsertable<Fake>::value, std::string>::type
01619         convert(const Fake& value) {
01620             ReusableStringStream rss;
01621             // NB: call using the function-like syntax to avoid ambiguity with
01622             // user-defined templated operator« under clang.
01623             rss.operator«(value);
01624             return rss.str();
01625         }
01626     };

```

```

01627     template <typename Fake = T>
01628     static
01629     typename std::enable_if<!::Catch::Detail::IsStreamInsertable<Fake>::value, std::string>::type
01630     convert( const Fake& value ) {
01631     #if !defined(CATCH_CONFIG_FALLBACK_STRINGIFIER)
01632         return Detail::convertUnstreamable(value);
01633     #else
01634         return CATCH_CONFIG_FALLBACK_STRINGIFIER(value);
01635     #endif
01636     }
01637 };
01638
01639 namespace Detail {
01640
01641     // This function dispatches all stringification requests inside of Catch.
01642     // Should be preferably called fully qualified, like ::Catch::Detail::stringify
01643     template <typename T>
01644     std::string stringify(const T& e) {
01645         return ::Catch::StringMaker<typename std::remove_cv<typename
std::remove_reference<T>::type>::type>::convert(e);
01646     }
01647
01648     template<typename E>
01649     std::string convertUnknownEnumToString( E e ) {
01650         return ::Catch::Detail::stringify(static_cast<typename std::underlying_type<E>::type>(e));
01651     }
01652
01653     #if defined(_MANAGED)
01654     template <typename T>
01655     std::string stringify( T^ e ) {
01656         return ::Catch::StringMaker<T^>::convert(e);
01657     }
01658     #endif
01659
01660     } // namespace Detail
01661
01662     // Some predefined specializations
01663
01664     template<>
01665     struct StringMaker<std::string> {
01666         static std::string convert(const std::string& str);
01667     };
01668
01669     #ifndef CATCH_CONFIG_CPP17_STRING_VIEW
01670     template<>
01671     struct StringMaker<std::string_view> {
01672         static std::string convert(std::string_view str);
01673     };
01674     #endif
01675
01676     template<>
01677     struct StringMaker<char const*> {
01678         static std::string convert(char const* str);
01679     };
01680
01681     template<>
01682     struct StringMaker<char*> {
01683         static std::string convert(char* str);
01684     };
01685
01686     #ifndef CATCH_CONFIG_WCHAR
01687     template<>
01688     struct StringMaker<std::wstring> {
01689         static std::string convert(const std::wstring& wstr);
01690     };
01691
01692     # if defined CATCH_CONFIG_CPP17_STRING_VIEW
01693     template<>
01694     struct StringMaker<std::wstring_view> {
01695         static std::string convert(std::wstring_view str);
01696     };
01697     # endif
01698
01699     template<>
01700     struct StringMaker<wchar_t const*> {
01701         static std::string convert(wchar_t const* str);
01702     };
01703
01704     template<>
01705     struct StringMaker<wchar_t*> {
01706         static std::string convert(wchar_t* str);
01707     };
01708     #endif
01709
01710     // TBD: Should we use `strlen` to ensure that we don't go out of the buffer,
01711     // while keeping string semantics?
01712     template<int SZ>
01713     struct StringMaker<char[SZ]> {
01714         static std::string convert(char const* str) {

```

```

01713         return ::Catch::Detail::stringify(std::string{ str });
01714     }
01715 };
01716 template<int SZ>
01717 struct StringMaker<signed char[SZ]> {
01718     static std::string convert(signed char const* str) {
01719         return ::Catch::Detail::stringify(std::string{ reinterpret_cast<char const *>(str) });
01720     }
01721 };
01722 template<int SZ>
01723 struct StringMaker<unsigned char[SZ]> {
01724     static std::string convert(unsigned char const* str) {
01725         return ::Catch::Detail::stringify(std::string{ reinterpret_cast<char const *>(str) });
01726     }
01727 };
01728
01729 #if defined(CATCH_CONFIG_CPP17_BYTE)
01730 template<>
01731 struct StringMaker<std::byte> {
01732     static std::string convert(std::byte value);
01733 };
01734 #endif // defined(CATCH_CONFIG_CPP17_BYTE)
01735 template<>
01736 struct StringMaker<int> {
01737     static std::string convert(int value);
01738 };
01739 template<>
01740 struct StringMaker<long> {
01741     static std::string convert(long value);
01742 };
01743 template<>
01744 struct StringMaker<long long> {
01745     static std::string convert(long long value);
01746 };
01747 template<>
01748 struct StringMaker<unsigned int> {
01749     static std::string convert(unsigned int value);
01750 };
01751 template<>
01752 struct StringMaker<unsigned long> {
01753     static std::string convert(unsigned long value);
01754 };
01755 template<>
01756 struct StringMaker<unsigned long long> {
01757     static std::string convert(unsigned long long value);
01758 };
01759
01760 template<>
01761 struct StringMaker<bool> {
01762     static std::string convert(bool b);
01763 };
01764
01765 template<>
01766 struct StringMaker<char> {
01767     static std::string convert(char c);
01768 };
01769 template<>
01770 struct StringMaker<signed char> {
01771     static std::string convert(signed char c);
01772 };
01773 template<>
01774 struct StringMaker<unsigned char> {
01775     static std::string convert(unsigned char c);
01776 };
01777
01778 template<>
01779 struct StringMaker<std::nullptr_t> {
01780     static std::string convert(std::nullptr_t);
01781 };
01782
01783 template<>
01784 struct StringMaker<float> {
01785     static std::string convert(float value);
01786     static int precision;
01787 };
01788
01789 template<>
01790 struct StringMaker<double> {
01791     static std::string convert(double value);
01792     static int precision;
01793 };
01794
01795 template <typename T>
01796 struct StringMaker<T*> {
01797     template <typename U>
01798     static std::string convert(U* p) {
01799         if (p) {

```

```

01800         return ::Catch::Detail::rawMemoryToString(p);
01801     } else {
01802         return "nullptr";
01803     }
01804 }
01805 };
01806
01807 template <typename R, typename C>
01808 struct StringMaker<R C::*> {
01809     static std::string convert(R C::* p) {
01810         if (p) {
01811             return ::Catch::Detail::rawMemoryToString(p);
01812         } else {
01813             return "nullptr";
01814         }
01815     }
01816 };
01817
01818 #if defined(_MANAGED)
01819     template <typename T>
01820     struct StringMaker<T^> {
01821         static std::string convert( T^ ref ) {
01822             return ::Catch::Detail::clrReferenceToString(ref);
01823         }
01824     };
01825 #endif
01826
01827 namespace Detail {
01828     template<typename InputIterator, typename Sentinel = InputIterator>
01829     std::string rangeToString(InputIterator first, Sentinel last) {
01830         ReusableStringStream rss;
01831         rss << "{ ";
01832         if (first != last) {
01833             rss << ::Catch::Detail::stringify(*first);
01834             for (++first; first != last; ++first)
01835                 rss << ", " << ::Catch::Detail::stringify(*first);
01836         }
01837         rss << " ";
01838         return rss.str();
01839     }
01840 }
01841
01842 #ifdef __OBJC__
01843     template<>
01844     struct StringMaker<NSString*> {
01845         static std::string convert(NSString * nsstring) {
01846             if (!nsstring)
01847                 return "nil";
01848             return std::string("@") + [nsstring UTF8String];
01849         }
01850     };
01851     template<>
01852     struct StringMaker<NSObject*> {
01853         static std::string convert(NSObject* nsObject) {
01854             return ::Catch::Detail::stringify([nsObject description]);
01855         }
01856     };
01857
01858     namespace Detail {
01859         inline std::string stringify( NSString* nsstring ) {
01860             return StringMaker<NSString*>::convert( nsstring );
01861         }
01862     } // namespace Detail
01863 #endif // __OBJC__
01864
01865 } // namespace Catch
01866
01867 // Separate std-lib types stringification, so it can be selectively enabled
01868 // This means that we do not bring in
01869
01870 #if defined(CATCH_CONFIG_ENABLE_ALL_STRINGMAKERS)
01871 # define CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER
01872 # define CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER
01873 # define CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER
01874 # define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
01875 # define CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER
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;

```

```

01888         rss << "{ "
01889             << ::Catch::Detail::stringify(pair.first)
01890             << ", "
01891             << ::Catch::Detail::stringify(pair.second)
01892             << " }";
01893         return rss.str();
01894     }
01895 };
01896 }
01897 #endif // CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER
01898
01899 #if defined(CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER) && defined(CATCH_CONFIG_CPP17_OPTIONAL)
01900 #include <optional>
01901 namespace Catch {
01902     template<typename T>
01903     struct StringMaker<std::optional<T> > {
01904         static std::string convert(const std::optional<T>& optional) {
01905             ReusableStringStream rss;
01906             if (optional.has_value()) {
01907                 rss << ::Catch::Detail::stringify(*optional);
01908             } else {
01909                 rss << "{ }";
01910             }
01911             return rss.str();
01912         }
01913     };
01914 }
01915 #endif // CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER
01916
01917 // Separate std::tuple specialization
01918 #if defined(CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER)
01919 #include <tuple>
01920 namespace Catch {
01921     namespace Detail {
01922         template<
01923             typename Tuple,
01924             std::size_t N = 0,
01925             bool = (N < std::tuple_size<Tuple>::value)
01926         >
01927         struct TupleElementPrinter {
01928             static void print(const Tuple& tuple, std::ostream& os) {
01929                 os << (N ? ", " : " ")
01930                     << ::Catch::Detail::stringify(std::get<N>(tuple));
01931                 TupleElementPrinter<Tuple, N + 1>::print(tuple, os);
01932             }
01933         };
01934
01935         template<
01936             typename Tuple,
01937             std::size_t N
01938         >
01939         struct TupleElementPrinter<Tuple, N, false> {
01940             static void print(const Tuple&, std::ostream&) {}
01941         };
01942     };
01943
01944     template<typename ...Types>
01945     struct StringMaker<std::tuple<Types...> > {
01946         static std::string convert(const std::tuple<Types...>& tuple) {
01947             ReusableStringStream rss;
01948             rss << '{';
01949             Detail::TupleElementPrinter<std::tuple<Types...>::print(tuple, rss.get());
01950             rss << " }";
01951             return rss.str();
01952         }
01953     };
01954 }
01955 #endif // CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER
01956
01957 #if defined(CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER) && defined(CATCH_CONFIG_CPP17_VARIANT)
01958 #include <variant>
01959 namespace Catch {
01960     template<
01961     struct StringMaker<std::monostate> {
01962         static std::string convert(const std::monostate&) {
01963             return "{ }";
01964         }
01965     };
01966
01967     template<typename... Elements>
01968     struct StringMaker<std::variant<Elements...> > {
01969         static std::string convert(const std::variant<Elements...>& variant) {
01970             if (variant.valueless_by_exception()) {
01971                 return "{valueless variant}";
01972             } else {
01973                 return std::visit(

```

```

01975         [] (const auto& value) {
01976             return ::Catch::Detail::stringify(value);
01977         },
01978         variant
01979     );
01980     }
01981 }
01982 };
01983 }
01984 #endif // CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER
01985
01986 namespace Catch {
01987     // Import begin/ end from std here
01988     using std::begin;
01989     using std::end;
01990
01991     namespace detail {
01992         template <typename...>
01993         struct void_type {
01994             using type = void;
01995         };
01996
01997         template <typename T, typename = void>
01998         struct is_range_impl : std::false_type {
01999         };
02000
02001         template <typename T>
02002         struct is_range_impl<T, typename void_type<decltype(begin(std::declval<T>()))>::type> :
02003             std::true_type {
02004         };
02005     } // namespace detail
02006
02007     template <typename T>
02008     struct is_range : detail::is_range_impl<T> {
02009     };
02010
02011     #if defined(_MANAGED) // Managed types are never ranges
02012     template <typename T>
02013     struct is_range<T^> {
02014         static const bool value = false;
02015     };
02016 #endif
02017
02018     template<typename Range>
02019     std::string rangeToString( Range const& range ) {
02020         return ::Catch::Detail::rangeToString( begin( range ), end( range ) );
02021     }
02022
02023     // Handle vector<bool> specially
02024     template<typename Allocator>
02025     std::string rangeToString( std::vector<bool, Allocator> const& v ) {
02026         ReusableStringStream rss;
02027         rss << "{ ";
02028         bool first = true;
02029         for( bool b : v ) {
02030             if( first )
02031                 first = false;
02032             else
02033                 rss << ", ";
02034             rss << ::Catch::Detail::stringify( b );
02035         }
02036         rss << " ";
02037         return rss.str();
02038     }
02039
02040     template<typename R>
02041     struct StringMaker<R, typename std::enable_if<is_range<R>::value &&
02042         !::Catch::Detail::IsStreamInsertable<R>::value>::type> {
02043         static std::string convert( R const& range ) {
02044             return rangeToString( range );
02045         }
02046     };
02047
02048     template <typename T, int SZ>
02049     struct StringMaker<T[SZ]> {
02050         static std::string convert( T const(&arr)[SZ] ) {
02051             return rangeToString(arr);
02052         }
02053     };
02054 } // namespace Catch
02055
02056 // Separate std::chrono::duration specialization
02057 #if defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
02058 #include <ctime>
02059 #include <ratio>
02060 #include <chrono>

```



```

02060
02061 namespace Catch {
02062
02063     template <class Ratio>
02064     struct ratio_string {
02065         static std::string symbol();
02066     };
02067
02068     template <class Ratio>
02069     std::string ratio_string<Ratio>::symbol() {
02070         Catch::ReusableStringStream rss;
02071         rss << '[' << Ratio::num << '/'
02072             << Ratio::den << ']';
02073         return rss.str();
02074     }
02075
02076     template <>
02077     struct ratio_string<std::atto> {
02078         static std::string symbol();
02079     };
02080
02081     template <>
02082     struct ratio_string<std::femto> {
02083         static std::string symbol();
02084     };
02085
02086     template <>
02087     struct ratio_string<std::pico> {
02088         static std::string symbol();
02089     };
02090
02091     template <>
02092     struct ratio_string<std::nano> {
02093         static std::string symbol();
02094     };
02095
02096     template <>
02097     struct ratio_string<std::micro> {
02098         static std::string symbol();
02099     };
02100
02101     template <>
02102     struct ratio_string<std::milli> {
02103         static std::string symbol();
02104     };
02105
02106     // std::chrono::duration specializations
02107     template<typename Value, typename Ratio>
02108     struct StringMaker<std::chrono::duration<Value, Ratio> const& duration> {
02109         static std::string convert(std::chrono::duration<Value, Ratio> const& duration) {
02110             ReusableStringStream rss;
02111             rss << duration.count() << ' ' << ratio_string<Ratio>::symbol() << 's';
02112             return rss.str();
02113         }
02114     };
02115
02116     template<typename Value>
02117     struct StringMaker<std::chrono::duration<Value, std::ratio<1>> const& duration> {
02118         static std::string convert(std::chrono::duration<Value, std::ratio<1>> const& duration) {
02119             ReusableStringStream rss;
02120             rss << duration.count() << " s";
02121             return rss.str();
02122         }
02123     };
02124
02125     template<typename Value>
02126     struct StringMaker<std::chrono::duration<Value, std::ratio<60>> const& duration> {
02127         static std::string convert(std::chrono::duration<Value, std::ratio<60>> const& duration) {
02128             ReusableStringStream rss;
02129             rss << duration.count() << " m";
02130             return rss.str();
02131         }
02132     };
02133
02134     template<typename Value>
02135     struct StringMaker<std::chrono::duration<Value, std::ratio<3600>> const& duration> {
02136         static std::string convert(std::chrono::duration<Value, std::ratio<3600>> const& duration) {
02137             ReusableStringStream rss;
02138             rss << duration.count() << " h";
02139             return rss.str();
02140         }
02141     };
02142
02143     // std::chrono::time_point specialization
02144     // Generic time_point cannot be specialized, only std::chrono::time_point<system_clock>
02145     template<typename Clock, typename Duration>
02146     struct StringMaker<std::chrono::time_point<Clock, Duration> const& time_point> {
02147         static std::string convert(std::chrono::time_point<Clock, Duration> const& time_point) {
02148             return ::Catch::Detail::stringify(time_point.time_since_epoch()) + " since epoch";
02149         }
02150     };
02151
02152     // std::chrono::time_point<system_clock> specialization
02153     template<typename Duration>
02154     struct StringMaker<std::chrono::time_point<std::chrono::system_clock, Duration> const& time_point> {
02155         static std::string convert(std::chrono::time_point<std::chrono::system_clock, Duration> const& time_point) {
02156             return ::Catch::Detail::stringify(time_point.time_since_epoch()) + " since epoch";
02157         }
02158     };
02159 }

```

```

02148         auto converted = std::chrono::system_clock::to_time_t(time_point);
02149
02150 #ifdef _MSC_VER
02151     std::tm timeInfo = {};
02152     gmtime_s(&timeInfo, &converted);
02153 #else
02154     std::tm* timeInfo = std::gmtime(&converted);
02155 #endif
02156
02157     auto const timeStampSize = sizeof("2017-01-16T17:06:45Z");
02158     char timeStamp[timeStampSize];
02159     const char * const fmt = "%Y-%m-%dT%H:%M:%SZ";
02160
02161 #ifdef _MSC_VER
02162     std::strftime(timeStamp, timeStampSize, fmt, &timeInfo);
02163 #else
02164     std::strftime(timeStamp, timeStampSize, fmt, timeInfo);
02165 #endif
02166     return std::string(timeStamp);
02167 }
02168 };
02169 }
02170 #endif // CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
02171
02172 #define INTERNAL_CATCH_REGISTER_ENUM( enumName, ... ) \
02173 namespace Catch { \
02174     template<> struct StringMaker<enumName> { \
02175         static std::string convert( enumName value ) { \
02176             static const auto& enumInfo = \
02177                 ::Catch::getMutableRegistryHub().getMutableEnumValuesRegistry().registerEnum( #enumName, #__VA_ARGS__, \
02178                 { #__VA_ARGS__ } ); \
02179             return static_cast<std::string>(enumInfo.lookup( static_cast<int>( value ) )); \
02180         } \
02181     }; \
02182
02182 #define CATCH_REGISTER_ENUM( enumName, ... ) INTERNAL_CATCH_REGISTER_ENUM( enumName, #__VA_ARGS__ )
02183
02184 #ifdef _MSC_VER
02185 #pragma warning(pop)
02186 #endif
02187
02188 // end catch_tostring.h
02189 #include <iosfwd>
02190
02191 #ifdef _MSC_VER
02192 #pragma warning(push)
02193 #pragma warning(disable:4389) // '==' : signed/unsigned mismatch
02194 #pragma warning(disable:4018) // more "signed/unsigned mismatch"
02195 #pragma warning(disable:4312) // Converting int to T* using reinterpret_cast (issue on x64 platform)
02196 #pragma warning(disable:4180) // qualifier applied to function type has no meaning
02197 #pragma warning(disable:4800) // Forcing result to true or false
02198 #endif
02199
02200 namespace Catch {
02201
02202     struct ITransientExpression {
02203     public:
02204         auto isBinaryExpression() const -> bool { return m_isBinaryExpression; }
02205         auto getResult() const -> bool { return m_result; }
02206         virtual void streamReconstructedExpression( std::ostream &os ) const = 0;
02207
02208     private:
02209         ITransientExpression( bool isBinaryExpression, bool result )
02210             :   m_isBinaryExpression( isBinaryExpression ),
02211                 m_result( result )
02212         {}
02213
02214         // We don't actually need a virtual destructor, but many static analysers
02215         // complain if it's not here :-|
02216         virtual ~ITransientExpression();
02217
02218     private:
02219         bool m_isBinaryExpression;
02220         bool m_result;
02221     };
02222
02223     void formatReconstructedExpression( std::ostream &os, std::string const& lhs, StringRef op,
02224         std::string const& rhs );
02225
02226     template<typename LhsT, typename RhsT>
02227     class BinaryExpr : public ITransientExpression {
02228     public:
02229         BinaryExpr( LhsT lhs, StringRef op, RhsT rhs )
02230             :   m_lhs( lhs ),
02231                 m_op( op ),
0231                 m_rhs( rhs )
02232         {}
02233
02234         void streamReconstructedExpression( std::ostream &os ) const override {
02235             formatReconstructedExpression( os, Catch::Detail::stringify( m_lhs ), m_op, Catch::Detail::stringify( m_rhs )

```

```

    );
02232     }
02233
02234     public:
02235         BinaryExpr( bool comparisonResult, LhsT lhs, StringRef op, RhsT rhs )
02236             : ITransientExpression{ true, comparisonResult },
02237               m_lhs( lhs ),
02238               m_op( op ),
02239               m_rhs( rhs )
02240         {}
02241
02242         template<typename T>
02243         auto operator && ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02244             static_assert(always_false<T>::value,
02245                 "chained comparisons are not supported inside assertions, "
02246                 "wrap the expression inside parentheses, or decompose it");
02247         }
02248
02249         template<typename T>
02250         auto operator || ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02251             static_assert(always_false<T>::value,
02252                 "chained comparisons are not supported inside assertions, "
02253                 "wrap the expression inside parentheses, or decompose it");
02254         }
02255
02256         template<typename T>
02257         auto operator == ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02258             static_assert(always_false<T>::value,
02259                 "chained comparisons are not supported inside assertions, "
02260                 "wrap the expression inside parentheses, or decompose it");
02261         }
02262
02263         template<typename T>
02264         auto operator != ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02265             static_assert(always_false<T>::value,
02266                 "chained comparisons are not supported inside assertions, "
02267                 "wrap the expression inside parentheses, or decompose it");
02268         }
02269
02270         template<typename T>
02271         auto operator > ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02272             static_assert(always_false<T>::value,
02273                 "chained comparisons are not supported inside assertions, "
02274                 "wrap the expression inside parentheses, or decompose it");
02275         }
02276
02277         template<typename T>
02278         auto operator < ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02279             static_assert(always_false<T>::value,
02280                 "chained comparisons are not supported inside assertions, "
02281                 "wrap the expression inside parentheses, or decompose it");
02282         }
02283
02284         template<typename T>
02285         auto operator >= ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02286             static_assert(always_false<T>::value,
02287                 "chained comparisons are not supported inside assertions, "
02288                 "wrap the expression inside parentheses, or decompose it");
02289         }
02290
02291         template<typename T>
02292         auto operator <= ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
02293             static_assert(always_false<T>::value,
02294                 "chained comparisons are not supported inside assertions, "
02295                 "wrap the expression inside parentheses, or decompose it");
02296         }
02297     };
02298
02299     template<typename LhsT>
02300     class UnaryExpr : public ITransientExpression {
02301     public:
02302         LhsT m_lhs;
02303
02304         void streamReconstructedExpression( std::ostream &os ) const override {
02305             os << Catch::Detail::stringify( m_lhs );
02306         }
02307     public:
02308         explicit UnaryExpr( LhsT lhs )
02309             : ITransientExpression{ false, static_cast<bool>(lhs) },
02310             m_lhs( lhs )
02311         {}
02312     };
02313
02314     // Specialised comparison functions to handle equality comparisons between ints and pointers (NULL
    deduces as an int)
02315     template<typename LhsT, typename RhsT>
02316     auto compareEqual( LhsT const& lhs, RhsT const& rhs ) -> bool { return static_cast<bool>(lhs ==

```

```

    rhs); }
02317     template<typename T>
02318     auto compareEqual( T* const& lhs, int rhs ) -> bool { return lhs == reinterpret_cast<void const*>(
rhs ); }
02319     template<typename T>
02320     auto compareEqual( T* const& lhs, long rhs ) -> bool { return lhs == reinterpret_cast<void
const*>( rhs ); }
02321     template<typename T>
02322     auto compareEqual( int lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs )
== rhs; }
02323     template<typename T>
02324     auto compareEqual( long lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs )
== rhs; }
02325
02326     template<typename LhsT, typename RhsT>
02327     auto compareNotEqual( LhsT const& lhs, RhsT&& rhs ) -> bool { return static_cast<bool>(lhs !=
rhs); }
02328     template<typename T>
02329     auto compareNotEqual( T* const& lhs, int rhs ) -> bool { return lhs != reinterpret_cast<void
const*>( rhs ); }
02330     template<typename T>
02331     auto compareNotEqual( T* const& lhs, long rhs ) -> bool { return lhs != reinterpret_cast<void
const*>( rhs ); }
02332     template<typename T>
02333     auto compareNotEqual( int lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs
) != rhs; }
02334     template<typename T>
02335     auto compareNotEqual( long lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>(
lhs ) != rhs; }
02336
02337     template<typename LhsT>
02338     class ExprLhs {
02339     LhsT m_lhs;
02340     public:
02341     explicit ExprLhs( LhsT lhs ) : m_lhs( lhs ) {}
02342
02343     template<typename RhsT>
02344     auto operator == ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02345         return { compareEqual( m_lhs, rhs ), m_lhs, "==", rhs };
02346     }
02347     auto operator == ( bool rhs ) -> BinaryExpr<LhsT, bool> const {
02348         return { m_lhs == rhs, m_lhs, "==", rhs };
02349     }
02350
02351     template<typename RhsT>
02352     auto operator != ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02353         return { compareNotEqual( m_lhs, rhs ), m_lhs, "!=", rhs };
02354     }
02355     auto operator != ( bool rhs ) -> BinaryExpr<LhsT, bool> const {
02356         return { m_lhs != rhs, m_lhs, "!=", rhs };
02357     }
02358
02359     template<typename RhsT>
02360     auto operator > ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02361         return { static_cast<bool>(m_lhs > rhs), m_lhs, ">", rhs };
02362     }
02363     template<typename RhsT>
02364     auto operator < ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02365         return { static_cast<bool>(m_lhs < rhs), m_lhs, "<", rhs };
02366     }
02367     template<typename RhsT>
02368     auto operator >= ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02369         return { static_cast<bool>(m_lhs >= rhs), m_lhs, ">=", rhs };
02370     }
02371     template<typename RhsT>
02372     auto operator <= ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
02373         return { static_cast<bool>(m_lhs <= rhs), m_lhs, "<=", rhs };
02374     }
02375     template <typename RhsT>
02376     auto operator | (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
02377         return { static_cast<bool>(m_lhs | rhs), m_lhs, "|", rhs };
02378     }
02379     template <typename RhsT>
02380     auto operator & (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
02381         return { static_cast<bool>(m_lhs & rhs), m_lhs, "&", rhs };
02382     }
02383     template <typename RhsT>
02384     auto operator ^ (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
02385         return { static_cast<bool>(m_lhs ^ rhs), m_lhs, "^", rhs };
02386     }
02387
02388     template<typename RhsT>
02389     auto operator && ( RhsT const& ) -> BinaryExpr<LhsT, RhsT const&> const {
02390         static_assert(always_false<RhsT>::value,
02391             "operator&& is not supported inside assertions, "
02392             "wrap the expression inside parentheses, or decompose it");
02393     }

```

```

02394
02395     template<typename RhsT>
02396     auto operator || ( RhsT const& ) -> BinaryExpr<LhsT, RhsT const&> const {
02397         static_assert(always_false<RhsT>::value,
02398             "operator|| is not supported inside assertions, "
02399             "wrap the expression inside parentheses, or decompose it");
02400     }
02401
02402     auto makeUnaryExpr() const -> UnaryExpr<LhsT> {
02403         return UnaryExpr<LhsT>{ m_lhs };
02404     }
02405 };
02406
02407 void handleExpression( ITransientExpression const& expr );
02408
02409 template<typename T>
02410 void handleExpression( ExprLhs<T> const& expr ) {
02411     handleExpression( expr.makeUnaryExpr() );
02412 }
02413
02414 struct Decomposer {
02415     template<typename T>
02416     auto operator <= ( T const& lhs ) -> ExprLhs<T const&> {
02417         return ExprLhs<T const&>{ lhs };
02418     }
02419
02420     auto operator <=( bool value ) -> ExprLhs<bool> {
02421         return ExprLhs<bool>{ value };
02422     }
02423 };
02424
02425 } // end namespace Catch
02426
02427 #ifdef _MSC_VER
02428 #pragma warning(pop)
02429 #endif
02430
02431 // end catch_decomposer.h
02432 // start catch_interfaces_capture.h
02433
02434 #include <string>
02435 #include <chrono>
02436
02437 namespace Catch {
02438
02439     class AssertionResult;
02440     struct AssertionInfo;
02441     struct SectionInfo;
02442     struct SectionEndInfo;
02443     struct MessageInfo;
02444     struct MessageBuilder;
02445     struct Counts;
02446     struct AssertionReaction;
02447     struct SourceLineInfo;
02448
02449     struct ITransientExpression;
02450     struct IGeneratorTracker;
02451
02452     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
02453     struct BenchmarkInfo;
02454     template <typename Duration = std::chrono::duration<double, std::nano>
02455     struct BenchmarkStats;
02456     #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
02457
02458     struct IResultCapture {
02459     public:
02460         virtual ~IResultCapture();
02461
02462         virtual bool sectionStarted( SectionInfo const& sectionInfo,
02463             Counts& assertions ) = 0;
02464         virtual void sectionEnded( SectionEndInfo const& endInfo ) = 0;
02465         virtual void sectionEndedEarly( SectionEndInfo const& endInfo ) = 0;
02466
02467         virtual auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo
02468         ) -> IGeneratorTracker& = 0;
02469
02470     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
02471         virtual void benchmarkPreparing( std::string const& name ) = 0;
02472         virtual void benchmarkStarting( BenchmarkInfo const& info ) = 0;
02473         virtual void benchmarkEnded( BenchmarkStats<> const& stats ) = 0;
02474         virtual void benchmarkFailed( std::string const& error ) = 0;
02475     #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
02476
02477         virtual void pushScopedMessage( MessageInfo const& message ) = 0;
02478         virtual void popScopedMessage( MessageInfo const& message ) = 0;
02479
02480         virtual void emplaceUnscopedMessage( MessageBuilder const& builder ) = 0;

```

```

02480
02481     virtual void handleFatalErrorCondition( StringRef message ) = 0;
02482
02483     virtual void handleExpr
02484         (   AssertionInfo const& info,
02485             ITransientExpression const& expr,
02486             AssertionReaction& reaction ) = 0;
02487     virtual void handleMessage
02488         (   AssertionInfo const& info,
02489             ResultWas::OfType resultType,
02490             StringRef const& message,
02491             AssertionReaction& reaction ) = 0;
02492     virtual void handleUnexpectedExceptionNotThrown
02493         (   AssertionInfo const& info,
02494             AssertionReaction& reaction ) = 0;
02495     virtual void handleUnexpectedInflightException
02496         (   AssertionInfo const& info,
02497             std::string const& message,
02498             AssertionReaction& reaction ) = 0;
02499     virtual void handleIncomplete
02500         (   AssertionInfo const& info ) = 0;
02501     virtual void handleNonExpr
02502         (   AssertionInfo const& info,
02503             ResultWas::OfType resultType,
02504             AssertionReaction& reaction ) = 0;
02505
02506     virtual bool lastAssertionPassed() = 0;
02507     virtual void assertionPassed() = 0;
02508
02509     // Deprecated, do not use:
02510     virtual std::string getCurrentTestName() const = 0;
02511     virtual const AssertionResult* getLastResult() const = 0;
02512     virtual void exceptionEarlyReported() = 0;
02513 };
02514
02515 IResultCapture& getResultCapture();
02516 }
02517
02518 // end catch_interfaces_capture.h
02519 namespace Catch {
02520
02521     struct TestFailureException{};
02522     struct AssertionResultData;
02523     struct IResultCapture;
02524     class RunContext;
02525
02526     class LazyExpression {
02527     friend class AssertionHandler;
02528     friend struct AssertionStats;
02529     friend class RunContext;
02530
02531     ITransientExpression const* m_transientExpression = nullptr;
02532     bool m_isNegated;
02533     public:
02534         LazyExpression( bool isNegated );
02535         LazyExpression( LazyExpression const& other );
02536         LazyExpression& operator = ( LazyExpression const& ) = delete;
02537
02538         explicit operator bool() const;
02539
02540         friend auto operator < ( std::ostream& os, LazyExpression const& lazyExpr ) -> std::ostream&;
02541     };
02542
02543     struct AssertionReaction {
02544         bool shouldDebugBreak = false;
02545         bool shouldThrow = false;
02546     };
02547
02548     class AssertionHandler {
02549     public:
02550         AssertionInfo m_assertionInfo;
02551         AssertionReaction m_reaction;
02552         bool m_completed = false;
02553         IResultCapture& m_resultCapture;
02554
02555     public:
02556         AssertionHandler
02557             (   StringRef const& macroName,
02558                 SourceLineInfo const& lineInfo,
02559                 StringRef capturedExpression,
02560                 ResultDisposition::Flags resultDisposition );
02561         ~AssertionHandler() {
02562             if ( !m_completed ) {
02563                 m_resultCapture.handleIncomplete( m_assertionInfo );
02564             }
02565         }
02566         template<typename T>

```

```

02567     void handleExpr( ExprLhs<T> const& expr ) {
02568         handleExpr( expr.makeUnaryExpr() );
02569     }
02570     void handleExpr( ITransientExpression const& expr );
02571
02572     void handleMessage( ResultWas::OfType resultType, StringRef const& message );
02573
02574     void handleExceptionThrownAsExpected();
02575     void handleUnexpectedExceptionNotThrown();
02576     void handleExceptionNotThrownAsExpected();
02577     void handleThrowingCallSkipped();
02578     void handleUnexpectedInflightException();
02579
02580     void complete();
02581     void setCompleted();
02582
02583     // query
02584     auto allowThrows() const -> bool;
02585 };
02586
02587     void handleExceptionMatchExpr( AssertionHandler& handler, std::string const& str, StringRef const&
matcherString );
02588 } // namespace Catch
02589
02590 // end catch_assertionhandler.h
02591 // start catch_message.h
02592
02593 #include <string>
02594 #include <vector>
02595
02596 namespace Catch {
02597     struct MessageInfo {
02598         MessageInfo( StringRef const& _macroName,
02599                     SourceLineInfo const& _lineInfo,
02600                     ResultWas::OfType _type );
02601
02602         StringRef macroName;
02603         std::string message;
02604         SourceLineInfo lineInfo;
02605         ResultWas::OfType type;
02606         unsigned int sequence;
02607
02608         bool operator == ( MessageInfo const& other ) const;
02609         bool operator < ( MessageInfo const& other ) const;
02610     private:
02611         static unsigned int globalCount;
02612     };
02613
02614     struct MessageStream {
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         ScopedMessage( ScopedMessage&& old );
02643         ~ScopedMessage();
02644
02645         MessageInfo m_info;
02646         bool m_moved;
02647     };
02648
02649     class Capturer {

```



```

02743     Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition); \
02744     if( catchAssertionHandler.allowThrows() ) \
02745         try { \
02746             static_cast<void>(__VA_ARGS__); \
02747             catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
02748         } \
02749         catch( ... ) { \
02750             catchAssertionHandler.handleExceptionThrownAsExpected(); \
02751         } \
02752     else \
02753         catchAssertionHandler.handleThrowingCallSkipped(); \
02754     INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02755 } while( false )
02756
02757 #define INTERNAL_CATCH_THROWS_AS( macroName, exceptionType, resultDisposition, expr ) \
02758 do { \
02759     Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
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,
Catch::StringRef(), resultDisposition ); \
02779     catchAssertionHandler.handleMessage( messageType, ( Catch::MessageStream() << __VA_ARGS__ +
::Catch::StreamEndStop() ).m_stream.str() ); \
02780     INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02781 } while( false )
02782
02783 #define INTERNAL_CATCH_CAPTURE( varName, macroName, ... ) \
02784 auto varName = Catch::Capturer( macroName, CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info,
__VA_ARGS__ ); \
02785 varName.captureValues( 0, __VA_ARGS__ )
02786
02787 #define INTERNAL_CATCH_INFO( macroName, log ) \
02788 Catch::ScopedMessage INTERNAL_CATCH_UNIQUE_NAME( scopedMessage )( Catch::MessageBuilder(
macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info ) << log );
02789
02790 #define INTERNAL_CATCH_UNSCOPED_INFO( macroName, log ) \
02791 Catch::getResultCapture().emplaceUnscopedMessage( Catch::MessageBuilder( macroName##_catch_sr,
CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info ) << log )
02792
02793 // Although this is matcher-based, it can be used with just a string
02794 #define INTERNAL_CATCH_THROWS_STR_MATCHES( macroName, resultDisposition, matcher, ... ) \
02795 do { \
02796     Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), " CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
02797     if( catchAssertionHandler.allowThrows() ) \
02798         try { \
02799             static_cast<void>(__VA_ARGS__); \
02800             catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
02801         } \
02802         catch( ... ) { \
02803             Catch::handleExceptionMatchExpr( catchAssertionHandler, matcher, #matcher##_catch_sr
); \
02804         } \
02805     else \
02806         catchAssertionHandler.handleThrowingCallSkipped(); \
02807     INTERNAL_CATCH_REACT( catchAssertionHandler ) \
02808 } while( false )
02809
02810 #endif // CATCH_CONFIG_DISABLE
02811
02812 // end catch_capture.hpp
02813 // start catch_section.h
02814
02815 // start catch_section_info.h
02816
02817 // start catch_totals.h
02818
02819 #include <cstdint>
02820
02821

```

```

02827 namespace Catch {
02828
02829     struct Counts {
02830         Counts operator - ( Counts const& other ) const;
02831         Counts& operator += ( Counts const& other );
02832
02833         std::size_t total() const;
02834         bool allPassed() const;
02835         bool allOk() const;
02836
02837         std::size_t passed = 0;
02838         std::size_t failed = 0;
02839         std::size_t failedButOk = 0;
02840     };
02841
02842     struct Totals {
02843
02844         Totals operator - ( Totals const& other ) const;
02845         Totals& operator += ( Totals const& other );
02846
02847         Totals delta( Totals const& prevTotals ) const;
02848
02849         int error = 0;
02850         Counts assertions;
02851         Counts testCases;
02852     };
02853 }
02854
02855 // end catch_totals.h
02856 #include <string>
02857
02858 namespace Catch {
02859
02860     struct SectionInfo {
02861         SectionInfo
02862             ( SourceLineInfo const& _lineInfo,
02863               std::string const& _name );
02864
02865         // Deprecated
02866         SectionInfo
02867             ( SourceLineInfo const& _lineInfo,
02868               std::string const& _name,
02869               std::string const& ) : SectionInfo( _lineInfo, _name ) {}
02870
02871         std::string name;
02872         std::string description; // !Deprecated: this will always be empty
02873         SourceLineInfo lineInfo;
02874     };
02875
02876     struct SectionEndInfo {
02877         SectionInfo sectionInfo;
02878         Counts prevAssertions;
02879         double durationInSeconds;
02880     };
02881
02882 } // end namespace Catch
02883
02884 // end catch_section_info.h
02885 // start catch_timer.h
02886
02887 #include <cstdint>
02888
02889 namespace Catch {
02890
02891     auto getCurrentNanosecondsSinceEpoch() -> uint64_t;
02892     auto getEstimatedClockResolution() -> uint64_t;
02893
02894     class Timer {
02895     public:
02896         uint64_t m_nanoseconds = 0;
02897         void start();
02898         auto getElapsedNanoseconds() const -> uint64_t;
02899         auto getElapsedMicroseconds() const -> uint64_t;
02900         auto getElapsedMilliseconds() const -> unsigned int;
02901         auto getElapsedSeconds() const -> double;
02902     };
02903
02904 } // namespace Catch
02905
02906 // end catch_timer.h
02907 #include <string>
02908
02909 namespace Catch {
02910
02911     class Section : NonCopyable {
02912     public:
02913         Section( SectionInfo const& info );

```

```

02914     ~Section();
02915
02916     // This indicates whether the section should be executed or not
02917     explicit operator bool() const;
02918
02919     private:
02920         SectionInfo m_info;
02921
02922         std::string m_name;
02923         Counts m_assertions;
02924         bool m_sectionIncluded;
02925         Timer m_timer;
02926     };
02927
02928 } // end namespace Catch
02929
02930 #define INTERNAL_CATCH_SECTION( ... ) \
02931     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
02932     CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
02933     if( Catch::Section const& INTERNAL_CATCH_UNIQUE_NAME( catch_internal_Section ) = \
02934         Catch::SectionInfo( CATCH_INTERNAL_LINEINFO, __VA_ARGS__ ) ) \
02935         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
02936
02937 #define INTERNAL_CATCH_DYNAMIC_SECTION( ... ) \
02938     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
02939     CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
02940     if( Catch::Section const& INTERNAL_CATCH_UNIQUE_NAME( catch_internal_Section ) = \
02941         Catch::SectionInfo( CATCH_INTERNAL_LINEINFO, (Catch::ReusableStringStream() << __VA_ARGS__).str() ) ) \
02942         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
02943
02944 // end catch_section.h
02945 // start catch_interfaces_exception.h
02946
02947 // start catch_interfaces_registry_hub.h
02948
02949 #include <string>
02950 #include <memory>
02951
02952 namespace Catch {
02953
02954     class TestCase;
02955     struct ITestCaseRegistry;
02956     struct IExceptionTranslatorRegistry;
02957     struct IExceptionTranslator;
02958     struct IReporterRegistry;
02959     struct IReporterFactory;
02960     struct ITagAliasRegistry;
02961     struct IMutableEnumValuesRegistry;
02962
02963     class StartupExceptionRegistry;
02964
02965     using IReporterFactoryPtr = std::shared_ptr<IReporterFactory>;
02966
02967     struct IRegistryHub {
02968         virtual ~IRegistryHub();
02969
02970         virtual IReporterRegistry const& getReporterRegistry() const = 0;
02971         virtual ITestCaseRegistry const& getTestCaseRegistry() const = 0;
02972         virtual ITagAliasRegistry const& getTagAliasRegistry() const = 0;
02973         virtual IExceptionTranslatorRegistry const& getExceptionTranslatorRegistry() const = 0;
02974
02975         virtual StartupExceptionRegistry const& getStartupExceptionRegistry() const = 0;
02976     };
02977
02978     struct IMutableRegistryHub {
02979         virtual ~IMutableRegistryHub();
02980         virtual void registerReporter( std::string const& name, IReporterFactoryPtr const& factory ) =
02981             0;
02982         virtual void registerListener( IReporterFactoryPtr const& factory ) = 0;
02983         virtual void registerTest( TestCase const& testInfo ) = 0;
02984         virtual void registerTranslator( const IExceptionTranslator* translator ) = 0;
02985         virtual void registerTagAlias( std::string const& alias, std::string const& tag,
02986             SourceLineInfo const& lineInfo ) = 0;
02987         virtual void registerStartupException() noexcept = 0;
02988         virtual IMutableEnumValuesRegistry& getMutableEnumValuesRegistry() = 0;
02989     };
02990
02991     IRegistryHub const& getRegistryHub();
02992     IMutableRegistryHub& getMutableRegistryHub();
02993     void cleanUp();
02994     std::string translateActiveException();
02995
02996 } // end catch_interfaces_registry_hub.h
02997
02998 #if defined(CATCH_CONFIG_DISABLE)
02999 #define INTERNAL_CATCH_TRANSLATE_EXCEPTION_NO_REG( translatorName, signature) \
03000     static std::string translate( signature )
03001
03002 #else
03003 #define INTERNAL_CATCH_TRANSLATE_EXCEPTION_NO_REG( translatorName, signature) \
03004     static std::string translate( signature )
03005
03006 #endif

```

```

02997         static std::string translatorName( signature )
02998     #endif
02999
03000     #include <exception>
03001     #include <string>
03002     #include <vector>
03003
03004     namespace Catch {
03005         using exceptionTranslateFunction = std::string(*)();
03006
03007         struct IExceptionTranslator;
03008         using ExceptionTranslators = std::vector<std::unique_ptr<IExceptionTranslator const>;
03009
03010         struct IExceptionTranslator {
03011             virtual ~IExceptionTranslator();
03012             virtual std::string translate( ExceptionTranslators::const_iterator it,
ExceptionTranslators::const_iterator itEnd ) const = 0;
03013         };
03014
03015         struct IExceptionTranslatorRegistry {
03016             virtual ~IExceptionTranslatorRegistry();
03017
03018             virtual std::string translateActiveException() const = 0;
03019         };
03020
03021         class ExceptionTranslatorRegistrar {
03022             template<typename T>
03023             class ExceptionTranslator : public IExceptionTranslator {
03024             public:
03025
03026                 ExceptionTranslator( std::string(*translateFunction)( T& ) )
03027                     : m_translateFunction( translateFunction )
03028                 {}
03029
03030                 std::string translate( ExceptionTranslators::const_iterator it,
ExceptionTranslators::const_iterator itEnd ) const override {
03031             #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
03032                 return "";
03033             #else
03034                 try {
03035                     if( it == itEnd )
03036                         std::rethrow_exception(std::current_exception());
03037                     else
03038                         return (*it)->translate( it+1, itEnd );
03039                 }
03040                 catch( T& ex ) {
03041                     return m_translateFunction( ex );
03042                 }
03043             #endif
03044         }
03045
03046         protected:
03047             std::string(*m_translateFunction)( T& );
03048     };
03049
03050     public:
03051         template<typename T>
03052         ExceptionTranslatorRegistrar( std::string(*translateFunction)( T& ) ) {
03053             getMutableRegistryHub().registerTranslator
03054                 ( new ExceptionTranslator<T>( translateFunction ) );
03055         }
03056     };
03057 }
03058
03060 #define INTERNAL_CATCH_TRANSLATE_EXCEPTION2( translatorName, signature ) \
03061     static std::string translatorName( signature ); \
03062     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
03063     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
03064     namespace{ Catch::ExceptionTranslatorRegistrar INTERNAL_CATCH_UNIQUE_NAME(
catch_internal_ExceptionRegistrar )( &translatorName ); } \
03065     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
03066     static std::string translatorName( signature )
03067
03068 #define INTERNAL_CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION2(
INTERNAL_CATCH_UNIQUE_NAME( catch_internal_ExceptionTranslator ), signature )
03069
03070 // end catch_interfaces_exception.h
03071 // start catch_approx.h
03072
03073 #include <type_traits>
03074
03075     namespace Catch {
03076         namespace Detail {
03077
03078             class Approx {
03079             private:
03080                 bool equalityComparisonImpl(double other) const;

```

```

03081         // Validates the new margin (margin >= 0)
03082         // out-of-line to avoid including stdexcept in the header
03083         void setMargin(double margin);
03084         // Validates the new epsilon (0 < epsilon < 1)
03085         // out-of-line to avoid including stdexcept in the header
03086         void setEpsilon(double epsilon);
03087
03088     public:
03089         explicit Approx ( double value );
03090
03091         static Approx custom();
03092
03093         Approx operator-() const;
03094
03095     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03096         Approx operator()( T const& value ) const {
03097         Approx approx( static_cast<double>(value) );
03098         approx.m_epsilon = m_epsilon;
03099         approx.m_margin = m_margin;
03100         approx.m_scale = m_scale;
03101         return approx;
03102     }
03103
03104     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03105         explicit Approx( T const& value ): Approx(static_cast<double>(value))
03106     {}
03107
03108     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03109         friend bool operator == ( const T& lhs, Approx const& rhs ) {
03110         auto lhs_v = static_cast<double>(lhs);
03111         return rhs.equalityComparisonImpl(lhs_v);
03112     }
03113
03114     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03115         friend bool operator == ( Approx const& lhs, const T& rhs ) {
03116         return operator==( rhs, lhs );
03117     }
03118
03119     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03120         friend bool operator != ( T const& lhs, Approx const& rhs ) {
03121         return !operator==( lhs, rhs );
03122     }
03123
03124     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03125         friend bool operator != ( Approx const& lhs, T const& rhs ) {
03126         return !operator==( rhs, lhs );
03127     }
03128
03129     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03130         friend bool operator <= ( T const& lhs, Approx const& rhs ) {
03131         return static_cast<double>(lhs) < rhs.m_value || lhs == rhs;
03132     }
03133
03134     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03135         friend bool operator <= ( Approx const& lhs, T const& rhs ) {
03136         return lhs.m_value < static_cast<double>(rhs) || lhs == rhs;
03137     }
03138
03139     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03140         friend bool operator >= ( T const& lhs, Approx const& rhs ) {
03141         return static_cast<double>(lhs) > rhs.m_value || lhs == rhs;
03142     }
03143
03144     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03145         friend bool operator >= ( Approx const& lhs, T const& rhs ) {
03146         return lhs.m_value > static_cast<double>(rhs) || lhs == rhs;
03147     }
03148
03149     template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03150         Approx& epsilon( T const& newEpsilon ) {
03151         double epsilonAsDouble = static_cast<double>(newEpsilon);
03152         setEpsilon(epsilonAsDouble);
03153         return *this;
03154     }
03155
03156

```

```

    template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03157        Approx& margin( T const& newMargin ) {
03158            double marginAsDouble = static_cast<double>(newMargin);
03159            setMargin(marginAsDouble);
03160            return *this;
03161        }
03162
03163    template <typename T, typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03164        Approx& scale( T const& newScale ) {
03165        m_scale = static_cast<double>(newScale);
03166        return *this;
03167        }
03168
03169        std::string toString() const;
03170
03171        private:
03172            double m_epsilon;
03173            double m_margin;
03174            double m_scale;
03175            double m_value;
03176        };
03177    } // end namespace Detail
03178
03179    namespace literals {
03180        Detail::Approx operator "" _a(long double val);
03181        Detail::Approx operator "" _a(unsigned long long val);
03182    } // end namespace literals
03183
03184    template<>
03185    struct StringMaker<Catch::Detail::Approx> {
03186        static std::string convert(Catch::Detail::Approx const& value);
03187    };
03188
03189 } // end namespace Catch
03190
03191 // end catch_approx.h
03192 // start catch_string_manip.h
03193
03194 #include <string>
03195 #include <iosfwd>
03196 #include <vector>
03197
03198 namespace Catch {
03199
03200    bool startsWith( std::string const& s, std::string const& prefix );
03201    bool startsWith( std::string const& s, char prefix );
03202    bool endsWith( std::string const& s, std::string const& suffix );
03203    bool endsWith( std::string const& s, char suffix );
03204    bool contains( std::string const& s, std::string const& infix );
03205    void toLowerInPlace( std::string& s );
03206    std::string toLower( std::string const& s );
03207    std::string trim( std::string const& str );
03210    StringRef trim(StringRef ref);
03211
03212    // !!! Be aware, returns refs into original string - make sure original string outlives them
03213    std::vector<StringRef> splitStringRef( StringRef str, char delimiter );
03214    bool replaceInPlace( std::string& str, std::string const& replaceThis, std::string const& withThis
03215 );
03216
03217    struct pluralise {
03218        pluralise( std::size_t count, std::string const& label );
03219
03220        friend std::ostream& operator < ( std::ostream& os, pluralise const& pluraliser );
03221
03222        std::size_t m_count;
03223        std::string m_label;
03224    };
03225
03226 // end catch_string_manip.h
03227 #ifndef CATCH_CONFIG_DISABLE_MATCHERS
03228 // start catch_capture_matchers.h
03229
03230 // start catch_matchers.h
03231
03232 #include <string>
03233 #include <vector>
03234
03235 namespace Catch {
03236     namespace Matchers {
03237         namespace Impl {
03238
03239             template<typename ArgT> struct MatchAllOf;
03240             template<typename ArgT> struct MatchAnyOf;
03241             template<typename ArgT> struct MatchNotOf;
03242

```

```

03243     class MatcherUntypedBase {
03244     public:
03245         MatcherUntypedBase() = default;
03246         MatcherUntypedBase ( MatcherUntypedBase const& ) = default;
03247         MatcherUntypedBase& operator = ( MatcherUntypedBase const& ) = delete;
03248         std::string toString() const;
03249
03250     protected:
03251         virtual ~MatcherUntypedBase();
03252         virtual std::string describe() const = 0;
03253         mutable std::string m_cachedToString;
03254     };
03255
03256 #ifdef __clang__
03257 # pragma clang diagnostic push
03258 # pragma clang diagnostic ignored "-Wnon-virtual-dtor"
03259 #endif
03260
03261     template<typename ObjectT>
03262     struct MatcherMethod {
03263         virtual bool match( ObjectT const& arg ) const = 0;
03264     };
03265
03266 #if defined(__OBJC__)
03267     // Hack to fix Catch GH issue #1661. Could use id for generic Object support.
03268     // use of const for Object pointers is very uncommon and under ARC it causes some kind of
03269     // signature mismatch that breaks compilation
03270     template<>
03271     struct MatcherMethod<NSString*> {
03272         virtual bool match( NSString* arg ) const = 0;
03273     };
03274 #endif
03275 #ifdef __clang__
03276 # pragma clang diagnostic pop
03277 #endif
03278
03279     template<typename T>
03280     struct MatcherBase : MatcherUntypedBase, MatcherMethod<T> {
03281
03282         MatchAllOf<T> operator && ( MatcherBase const& other ) const;
03283         MatchAnyOf<T> operator || ( MatcherBase const& other ) const;
03284         MatchNotOf<T> operator ! ( ) const;
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
03312         MatchAllOf<ArgT> operator && ( MatcherBase<ArgT> const& other ) {
03313             auto copy(*this);
03314             copy.m_matchers.push_back( &other );
03315             return copy;
03316         }
03317
03318         std::vector<MatcherBase<ArgT> const*> m_matchers;
03319     };
03320
03321     template<typename ArgT>
03322     struct MatchAnyOf : MatcherBase<ArgT> {
03323
03324         bool match( ArgT const& arg ) const override {
03325             for( auto matcher : m_matchers ) {
03326                 if (matcher->match(arg))
03327                     return true;
03328             }
03329             return false;
03330         }
03331     };

```

```

03329     }
03330     std::string describe() const override {
03331         std::string description;
03332         description.reserve( 4 + m_matchers.size()*32 );
03333         description += " ( ";
03334         bool first = true;
03335         for( auto matcher : m_matchers ) {
03336             if( first )
03337                 first = false;
03338             else
03339                 description += " or ";
03340             description += matcher->toString();
03341         }
03342         description += " )";
03343         return description;
03344     }
03345
03346     MatchAnyOf<ArgT> operator || ( MatcherBase<ArgT> const& other ) {
03347         auto copy(*this);
03348         copy.m_matchers.push_back( &other );
03349         return copy;
03350     }
03351
03352     std::vector<MatcherBase<ArgT> const*> m_matchers;
03353 };
03354
03355 template<typename ArgT>
03356 struct MatchNotOf : MatcherBase<ArgT> {
03357
03358     MatchNotOf( MatcherBase<ArgT> const& underlyingMatcher ) : m_underlyingMatcher(
underlyingMatcher ) {}
03359
03360     bool match( ArgT const& arg ) const override {
03361         return !m_underlyingMatcher.match( arg );
03362     }
03363
03364     std::string describe() const override {
03365         return "not " + m_underlyingMatcher.toString();
03366     }
03367     MatcherBase<ArgT> const& m_underlyingMatcher;
03368 };
03369
03370 template<typename T>
03371 MatchAllOf<T> MatcherBase<T>::operator && ( MatcherBase const& other ) const {
03372     return MatchAllOf<T>() && *this && other;
03373 }
03374
03375 template<typename T>
03376 MatchAnyOf<T> MatcherBase<T>::operator || ( MatcherBase const& other ) const {
03377     return MatchAnyOf<T>() || *this || other;
03378 }
03379
03380 template<typename T>
03381 MatchNotOf<T> MatcherBase<T>::operator ! () const {
03382     return MatchNotOf<T>( *this );
03383 }
03384
03385 } // namespace Impl
03386
03387 } // namespace Matchers
03388
03389 using namespace Matchers;
03390 using Matchers::Impl::MatcherBase;
03391
03392 } // namespace Catch
03393 // end catch_matchers.h
03394 // start catch_matchers_exception.hpp
03395
03396 namespace Catch {
03397     namespace Matchers {
03398         namespace Exception {
03399
03400             class ExceptionMessageMatcher : public MatcherBase<std::exception> {
03401             public:
03402                 ExceptionMessageMatcher( std::string const& message ) :
03403                     m_message( message )
03404                 {}
03405
03406                 bool match( std::exception const& ex ) const override;
03407
03408                 std::string describe() const override;
03409             };
03410
03411         } // namespace Exception
03412
03413         Exception::ExceptionMessageMatcher Message( std::string const& message );
03414

```



```

03415
03416     } // namespace Matchers
03417 } // namespace Catch
03418
03419 // end catch_matchers_exception.hpp
03420 // start catch_matchers_floating.h
03421
03422 namespace Catch {
03423     namespace Matchers {
03424
03425         namespace Floating {
03426
03427             enum class FloatingPointKind : uint8_t;
03428
03429             struct WithinAbsMatcher : MatcherBase<double> {
03430                 WithinAbsMatcher(double target, double margin);
03431                 bool match(double const& matchee) const override;
03432                 std::string describe() const override;
03433             private:
03434                 double m_target;
03435                 double m_margin;
03436             };
03437
03438             struct WithinUlpMatcher : MatcherBase<double> {
03439                 WithinUlpMatcher(double target, uint64_t ulps, FloatingPointKind baseType);
03440                 bool match(double const& matchee) const override;
03441                 std::string describe() const override;
03442             private:
03443                 double m_target;
03444                 uint64_t m_ulps;
03445                 FloatingPointKind m_type;
03446             };
03447
03448             // Given IEEE-754 format for floats and doubles, we can assume
03449             // that float -> double promotion is lossless. Given this, we can
03450             // assume that if we do the standard relative comparison of
03451             // |lhs - rhs| <= epsilon * max(fabs(lhs), fabs(rhs)), then we get
03452             // the same result if we do this for floats, as if we do this for
03453             // doubles that were promoted from floats.
03454             struct WithinRelMatcher : MatcherBase<double> {
03455                 WithinRelMatcher(double target, double epsilon);
03456                 bool match(double const& matchee) const override;
03457                 std::string describe() const override;
03458             private:
03459                 double m_target;
03460                 double m_epsilon;
03461             };
03462
03463         } // namespace Floating
03464
03465         // The following functions create the actual matcher objects.
03466         // This allows the types to be inferred
03467         Floating::WithinUlpMatcher WithinULP(double target, uint64_t maxUlpDiff);
03468         Floating::WithinUlpMatcher WithinULP(float target, uint64_t maxUlpDiff);
03469         Floating::WithinAbsMatcher WithinAbs(double target, double margin);
03470         Floating::WithinRelMatcher WithinRel(double target, double eps);
03471         // defaults epsilon to 100*numeric_limits<double>::epsilon()
03472         Floating::WithinRelMatcher WithinRel(double target);
03473         Floating::WithinRelMatcher WithinRel(float target, float eps);
03474         // defaults epsilon to 100*numeric_limits<float>::epsilon()
03475         Floating::WithinRelMatcher WithinRel(float target);
03476
03477     } // namespace Matchers
03478 } // namespace Catch
03479
03480 // end catch_matchers_floating.h
03481 // start catch_matchers_generic.hpp
03482
03483 #include <functional>
03484 #include <string>
03485
03486 namespace Catch {
03487     namespace Matchers {
03488         namespace Generic {
03489
03490             namespace Detail {
03491                 std::string finalizeDescription(const std::string& desc);
03492             }
03493
03494             template <typename T>
03495             class PredicateMatcher : public MatcherBase<T> {
03496                 std::function<bool(T const&)> m_predicate;
03497                 std::string m_description;
03498             public:
03499                 PredicateMatcher(std::function<bool(T const&)> const& elem, std::string const& descr)
03500                     :m_predicate(std::move(elem)),

```

```

03502         m_description(Detail::finalizeDescription(descr))
03503     {}
03504
03505     bool match( T const& item ) const override {
03506         return m_predicate(item);
03507     }
03508
03509     std::string describe() const override {
03510         return m_description;
03511     }
03512 };
03513
03514 } // namespace Generic
03515
03516 // The following functions create the actual matcher objects.
03517 // The user has to explicitly specify type to the function, because
03518 // inferring std::function<bool(T const&)> is hard (but possible) and
03519 // requires a lot of TMP.
03520 template<typename T>
03521 Generic::PredicateMatcher<T> Predicate(std::function<bool(T const&)> const& predicate,
03522 std::string const& description = "") {
03523     return Generic::PredicateMatcher<T>(predicate, description);
03524 }
03525 } // namespace Matchers
03526 } // namespace Catch
03527
03528 // end catch_matchers_generic.hpp
03529 // start catch_matchers_string.h
03530
03531 #include <string>
03532
03533 namespace Catch {
03534     namespace Matchers {
03535         namespace StdString {
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         } // namespace StdString
03581
03582         // The following functions create the actual matcher objects.
03583         // This allows the types to be inferred
03584

```

```

03588     StdString::EqualsMatcher Equals( std::string const& str, CaseSensitive::Choice caseSensitivity
= CaseSensitive::Yes );
03589     StdString::ContainsMatcher Contains( std::string const& str, CaseSensitive::Choice
caseSensitivity = CaseSensitive::Yes );
03590     StdString::EndsWithMatcher EndsWith( std::string const& str, CaseSensitive::Choice
caseSensitivity = CaseSensitive::Yes );
03591     StdString::StartsWithMatcher StartsWith( std::string const& str, CaseSensitive::Choice
caseSensitivity = CaseSensitive::Yes );
03592     StdString::RegexMatcher Matches( std::string const& regex, CaseSensitive::Choice
caseSensitivity = CaseSensitive::Yes );
03593
03594     } // namespace Matchers
03595 } // namespace Catch
03596
03597 // end catch_matchers_string.h
03598 // start catch_matchers_vector.h
03599
03600 #include <algorithm>
03601
03602 namespace Catch {
03603     namespace Matchers {
03604         namespace Vector {
03605             template<typename T, typename Alloc>
03606             struct ContainsElementMatcher : MatcherBase<std::vector<T, Alloc> > {
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                 ContainsMatcher(std::vector<T, AllocComp> const &comparator) : m_comparator(
comparator ) {}
03628
03629                 bool match(std::vector<T, AllocMatch> const &v) const override {
03630                     // !TBD: see note in EqualsMatcher
03631                     if (m_comparator.size() > v.size())
03632                         return false;
03633                     for (auto const& comparator : m_comparator) {
03634                         auto present = false;
03635                         for (const auto& el : v) {
03636                             if (el == comparator) {
03637                                 present = true;
03638                                 break;
03639                             }
03640                         }
03641                         if (!present) {
03642                             return false;
03643                         }
03644                     }
03645                     return true;
03646                 }
03647
03648                 std::string describe() const override {
03649                     return "Contains: " + ::Catch::Detail::stringify( m_comparator );
03650                 }
03651
03652                 std::vector<T, AllocComp> const& m_comparator;
03653             };
03654
03655             template<typename T, typename AllocComp, typename AllocMatch>
03656             struct EqualsMatcher : MatcherBase<std::vector<T, AllocMatch> > {
03657                 EqualsMatcher(std::vector<T, AllocComp> const &comparator) : m_comparator( comparator
) {}
03658
03659                 bool match(std::vector<T, AllocMatch> const &v) const override {
03660                     // !TBD: This currently works if all elements can be compared using !=
03661                     // - a more general approach would be via a compare template that defaults
03662                     // to using !=. but could be specialised for, e.g. std::vector<T, Alloc> etc
03663                     // - then just call that directly
03664                     if (m_comparator.size() != v.size())

```

```

03668         return false;
03669     for (std::size_t i = 0; i < v.size(); ++i)
03670         if (m_comparator[i] != v[i])
03671             return false;
03672     return true;
03673 }
03674 std::string describe() const override {
03675     return "Equals: " + ::Catch::Detail::stringify( m_comparator );
03676 }
03677 std::vector<T, AllocComp> const& m_comparator;
03678 };
03679
03680 template<typename T, typename AllocComp, typename AllocMatch>
03681 struct ApproxMatcher : MatcherBase<std::vector<T, AllocMatch> {
03682
03683     ApproxMatcher(std::vector<T, AllocComp> const& comparator) : m_comparator( comparator
03684 ) {}
03685
03686     bool match(std::vector<T, AllocMatch> const &v) const override {
03687         if (m_comparator.size() != v.size())
03688             return false;
03689         for (std::size_t i = 0; i < v.size(); ++i)
03690             if (m_comparator[i] != approx(v[i]))
03691                 return false;
03692         return true;
03693     }
03694     std::string describe() const override {
03695         return "is approx: " + ::Catch::Detail::stringify( m_comparator );
03696     }
03697
03698     template <typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03699     ApproxMatcher& epsilon( T const& newEpsilon ) {
03700         approx.epsilon(newEpsilon);
03701         return *this;
03702     }
03703
03704     template <typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03705     ApproxMatcher& margin( T const& newMargin ) {
03706         approx.margin(newMargin);
03707         return *this;
03708     }
03709
03710     template <typename = typename std::enable_if<std::is_constructible<double, T::value>::type>
03711     ApproxMatcher& scale( T const& newScale ) {
03712         approx.scale(newScale);
03713         return *this;
03714     }
03715
03716     std::vector<T, AllocComp> const& m_comparator;
03717     mutable Catch::Detail::Approx approx = Catch::Detail::Approx::custom();
03718 };
03719
03720 template<typename T, typename AllocComp, typename AllocMatch>
03721 struct UnorderedEqualsMatcher : MatcherBase<std::vector<T, AllocMatch> {
03722     UnorderedEqualsMatcher(std::vector<T, AllocComp> const& target) : m_target(target) {}
03723     bool match(std::vector<T, AllocMatch> const& vec) const override {
03724         if (m_target.size() != vec.size()) {
03725             return false;
03726         }
03727         return std::is_permutation(m_target.begin(), m_target.end(), vec.begin());
03728     }
03729     std::string describe() const override {
03730         return "UnorderedEquals: " + ::Catch::Detail::stringify(m_target);
03731     }
03732     private:
03733     std::vector<T, AllocComp> const& m_target;
03734 };
03735
03736 } // namespace Vector
03737
03738 // The following functions create the actual matcher objects.
03739 // This allows the types to be inferred
03740
03741 template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
03742 Vector::ContainsMatcher<T, AllocComp, AllocMatch> Contains( std::vector<T, AllocComp> const&
03743 comparator ) {
03744     return Vector::ContainsMatcher<T, AllocComp, AllocMatch>( comparator );
03745 }
03746
03747 template<typename T, typename Alloc = std::allocator<T>
03748 Vector::ContainsElementMatcher<T, Alloc> VectorContains( T const& comparator ) {
03749     return Vector::ContainsElementMatcher<T, Alloc>( comparator );
03750 }
03751
03752 template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
03753 Vector::EqualsMatcher<T, AllocComp, AllocMatch> Equals( std::vector<T, AllocComp> const&

```

```

    comparator ) {
03750         return Vector::EqualsMatcher<T, AllocComp, AllocMatch>( comparator );
03751     }
03752
03753     template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
03754     Vector::ApproxMatcher<T, AllocComp, AllocMatch> Approx( std::vector<T, AllocComp> const&
comparator ) {
03755         return Vector::ApproxMatcher<T, AllocComp, AllocMatch>( comparator );
03756     }
03757
03758     template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
03759     Vector::UnorderedEqualsMatcher<T, AllocComp, AllocMatch> UnorderedEquals( std::vector<T,
AllocComp> const& target ) {
03760         return Vector::UnorderedEqualsMatcher<T, AllocComp, AllocMatch>( target );
03761     }
03762
03763     } // namespace Matchers
03764 } // namespace Catch
03765
03766 // end catch_matchers_vector.h
03767 namespace Catch {
03768
03769     template<typename ArgT, typename MatcherT>
03770     class MatchExpr : public ITransientExpression {
03771     public:
03772         ArgT const& m_arg;
03773         MatcherT m_matcher;
03774         StringRef m_matcherString;
03775     public:
03776         MatchExpr( ArgT const& arg, MatcherT const& matcher, StringRef const& matcherString )
03777             : ITransientExpression( true, matcher.match( arg ) ),
03778               m_arg( arg ),
03779               m_matcher( matcher ),
03780               m_matcherString( matcherString ) {}
03781
03782         void streamReconstructedExpression( std::ostream &os ) const override {
03783             auto matcherAsString = m_matcher.toString();
03784             os << Catch::Detail::stringify( m_arg ) << ' ' <<
03785               if( matcherAsString == Detail::unprintableString )
03786                 os << m_matcherString;
03787             else
03788                 os << matcherAsString;
03789         }
03790     };
03791
03792     using StringMatcher = Matchers::Impl::MatcherBase<std::string>;
03793
03794     void handleExceptionMatchExpr( AssertionHandler& handler, StringMatcher const& matcher, StringRef
const& matcherString );
03795
03796     template<typename ArgT, typename MatcherT>
03797     auto makeMatchExpr( ArgT const& arg, MatcherT const& matcher, StringRef const& matcherString ) ->
MatchExpr<ArgT, MatcherT> {
03798         return MatchExpr<ArgT, MatcherT>( arg, matcher, matcherString );
03799     }
03800 } // namespace Catch
03801
03802 #define INTERNAL_CHECK_THAT( macroName, matcher, resultDisposition, arg ) \
03803     do { \
03804         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
CATCH_INTERNAL_STRINGIFY(arg) ", " CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
03805         INTERNAL_CATCH_TRY { \
03806             catchAssertionHandler.handleExpr( Catch::makeMatchExpr( arg, matcher, #matcher##_catch_sr \
) ); \
03807         } INTERNAL_CATCH_CATCH( catchAssertionHandler ) \
03808         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
03809     } while( false )
03810
03811 #define INTERNAL_CATCH_THROWS_MATCHES( macroName, exceptionType, resultDisposition, matcher, ... ) \
03812     do { \
03813         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
CATCH_INTERNAL_STRINGIFY(__VA_ARGS__) ", " CATCH_INTERNAL_STRINGIFY(exceptionType) ", " \
CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
03814         if( catchAssertionHandler.allowThrows() ) \
03815             try { \
03816                 static_cast<void>( __VA_ARGS__ ); \
03817                 catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
03818             } \
03819             catch( exceptionType const& ex ) { \
03820                 catchAssertionHandler.handleExpr( Catch::makeMatchExpr( ex, matcher, \
#matcher##_catch_sr ) ); \
03821             } \
03822         catch( ... ) { \
03823             catchAssertionHandler.handleUnexpectedInflightException(); \
03824         } \
03825     } \
03826     else \
03827         ;

```

```

03829         catchAssertionHandler.handleThrowingCallSkipped(); \
03830         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
03831     } while( false )
03832
03833 // end catch_capture_matchers.h
03834 #endif
03835 // start catch_generators.hpp
03836
03837 // start catch_interfaces_generatortracker.h
03838
03839 #include <memory>
03840
03841 namespace Catch {
03842
03843     namespace Generators {
03844         class GeneratorUntypedBase {
03845         public:
03846             GeneratorUntypedBase() = default;
03847             virtual ~GeneratorUntypedBase();
03848             // Attempts to move the generator to the next element
03849             //
03850             // Returns true iff the move succeeded (and a valid element
03851             // can be retrieved).
03852             virtual bool next() = 0;
03853         };
03854         using GeneratorBasePtr = std::unique_ptr<GeneratorUntypedBase>;
03855     } // namespace Generators
03856
03857     struct IGeneratorTracker {
03858     public:
03859         virtual ~IGeneratorTracker();
03860         virtual auto hasGenerator() const -> bool = 0;
03861         virtual auto getGenerator() const -> Generators::GeneratorBasePtr const& = 0;
03862         virtual void setGenerator( Generators::GeneratorBasePtr&& generator ) = 0;
03863     };
03864 } // namespace Catch
03865
03866 // end catch_interfaces_generatortracker.h
03867 // start catch_enforce.h
03868
03869 #include <exception>
03870
03871 namespace Catch {
03872
03873     #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
03874         template <typename Ex>
03875         [[noreturn]]
03876         void throw_exception(Ex const& e) {
03877             throw e;
03878         }
03879     #else // Exceptions are disabled
03880         [[noreturn]]
03881         void throw_exception(std::exception const& e);
03882     #endif
03883
03884     [[noreturn]]
03885     void throw_logic_error(std::string const& msg);
03886     [[noreturn]]
03887     void throw_domain_error(std::string const& msg);
03888     [[noreturn]]
03889     void throw_runtime_error(std::string const& msg);
03890 } // namespace Catch;
03891
03892 #define CATCH_MAKE_MSG(...) \
03893     (Catch::ReusableStringStream() << __VA_ARGS__).str()
03894
03895 #define CATCH_INTERNAL_ERROR(...) \
03896     Catch::throw_logic_error(CATCH_MAKE_MSG( CATCH_INTERNAL_LINEINFO << "Internal Catch2 error: " << __VA_ARGS__))
03897
03898 #define CATCH_ERROR(...) \
03899     Catch::throw_domain_error(CATCH_MAKE_MSG( __VA_ARGS__ ))
03900
03901 #define CATCH_RUNTIME_ERROR(...) \
03902     Catch::throw_runtime_error(CATCH_MAKE_MSG( __VA_ARGS__ ))
03903
03904 #define CATCH_ENFORCE( condition, ... ) \
03905     do{ if( !(condition) ) CATCH_ERROR( __VA_ARGS__ ); } while(false)
03906
03907 // end catch_enforce.h
03908
03909 #include <memory>
03910 #include <vector>
03911 #include <cassert>
03912
03913 #include <utility>

```

```

03915 #include <exception>
03916
03917 namespace Catch {
03918
03919     class GeneratorException : public std::exception {
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             template<typename T>
03938             struct IGenerator : GeneratorUntypedBase {
03939                 virtual ~IGenerator() = default;
03940
03941                 // Returns the current element of the generator
03942                 // \Precondition The generator is either freshly constructed,
03943                 // or the last call to `next()` returned true
03944                 virtual T const& get() const = 0;
03945                 using type = T;
03946             };
03947
03948             template<typename T>
03949             class SingleValueGenerator final : public IGenerator<T> {
03950     public:
03951                 SingleValueGenerator(T&& value) : m_value(std::move(value)) {}
03952
03953                 T const& get() const override {
03954                     return m_value;
03955                 }
03956                 bool next() override {
03957                     return false;
03958                 }
03959             };
03960
03961             template<typename T>
03962             class FixedValuesGenerator final : public IGenerator<T> {
03963     public:
03964                 FixedValuesGenerator( std::initializer_list<T> values ) : m_values( values ) {}
03965
03966                 T const& get() const override {
03967                     return m_values[m_idx];
03968                 }
03969                 bool next() override {
03970                     ++m_idx;
03971                     return m_idx < m_values.size();
03972                 }
03973             };
03974
03975             template <typename T>
03976             class GeneratorWrapper final {
03977     public:
03978                 GeneratorWrapper(std::unique_ptr<IGenerator<T>> generator):
03979                     m_generator(std::move(generator))
03980                 {}
03981                 T const& get() const {
03982                     return m_generator->get();
03983                 }
03984                 bool next() {
03985                     return m_generator->next();
03986                 }
03987             };
03988
03989             template <typename T>
03990             GeneratorWrapper<T> value(T&& value) {
03991

```

```

04002         return
04003     GeneratorWrapper<T>(pf::make_unique<SingleValueGenerator<T>>(std::forward<T>(value)));
04004     }
04005     template <typename T>
04006     GeneratorWrapper<T> values(std::initializer_list<T> values) {
04007         return GeneratorWrapper<T>(pf::make_unique<FixedValuesGenerator<T>>(values));
04008     }
04009     template<typename T>
04010     class Generators : public IGenerator<T> {
04011     public:
04012         std::vector<GeneratorWrapper<T> m_generators;
04013         size_t m_current = 0;
04014
04015         void populate(GeneratorWrapper<T>&& generator) {
04016             m_generators.emplace_back(std::move(generator));
04017         }
04018         void populate(T&& val) {
04019             m_generators.emplace_back(value(std::forward<T>(val)));
04020         }
04021         template<typename U>
04022         void populate(U&& val) {
04023             populate(T(std::forward<U>(val)));
04024         }
04025         template<typename U, typename... Gs>
04026         void populate(U& valueOrGenerator, Gs &&... moreGenerators) {
04027             populate(std::forward<U>(valueOrGenerator));
04028             populate(std::forward<Gs>(moreGenerators)...);
04029         }
04030     public:
04031         template <typename... Gs>
04032         Generators(Gs &&... moreGenerators) {
04033             m_generators.reserve(sizeof...(Gs));
04034             populate(std::forward<Gs>(moreGenerators)...);
04035         }
04036
04037         T const& get() const override {
04038             return m_generators[m_current].get();
04039         }
04040
04041         bool next() override {
04042             if (m_current >= m_generators.size()) {
04043                 return false;
04044             }
04045             const bool current_status = m_generators[m_current].next();
04046             if (!current_status) {
04047                 ++m_current;
04048             }
04049             return m_current < m_generators.size();
04050         }
04051     };
04052
04053     template<typename... Ts>
04054     GeneratorWrapper<std::tuple<Ts...> table( std::initializer_list<std::tuple<typename
04055     std::decay<Ts>::type...> tuples ) {
04056         return values<std::tuple<Ts...>( tuples );
04057     }
04058
04059     // Tag type to signal that a generator sequence should convert arguments to a specific type
04060     template <typename T>
04061     struct as {};
04062
04063     template<typename T, typename... Gs>
04064     auto makeGenerators( GeneratorWrapper<T>&& generator, Gs &&... moreGenerators ) ->
04065     Generators<T> {
04066         return Generators<T>(std::move(generator), std::forward<Gs>(moreGenerators)...);
04067     }
04068     template<typename T>
04069     auto makeGenerators( GeneratorWrapper<T>&& generator ) -> Generators<T> {
04070         return Generators<T>(std::move(generator));
04071     }
04072     template<typename T, typename... Gs>
04073     auto makeGenerators( T&& val, Gs &&... moreGenerators ) -> Generators<T> {
04074         return makeGenerators( value( std::forward<T>( val ) ), std::forward<Gs>( moreGenerators
04075     )... );
04076     }
04077     template<typename T, typename U, typename... Gs>
04078     auto makeGenerators( as<T>, U&& val, Gs &&... moreGenerators ) -> Generators<T> {
04079         return makeGenerators( value( T( std::forward<U>( val ) ) ), std::forward<Gs>(
04080     moreGenerators )... );
04081     }
04082     auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
04083     IGeneratorTracker&;
04084
04085     template<typename L>
04086     // Note: The type after -> is weird, because VS2015 cannot parse

```



```

04083         //         the expression used in the typedef inside, when it is in
04084         //         return type. Yeah.
04085         auto generate( StringRef generatorName, SourceLineInfo const& lineInfo, L const&
generatorExpression ) -> decltype( std::declval<decltype(generatorExpression())>().get() ) {
04086             using UnderlyingType = typename decltype(generatorExpression())::type;
04087
04088             IGeneratorTracker& tracker = acquireGeneratorTracker( generatorName, lineInfo );
04089             if (!tracker.hasGenerator()) {
04090                 tracker.setGenerator(pf::make_unique<Generators<UnderlyingType>>(generatorExpression()));
04091             }
04092
04093             auto const& generator = static_cast<IGenerator<UnderlyingType>> const&>(
*tracker.getGenerator() );
04094             return generator.get();
04095         }
04096     } // namespace Generators
04097 } // namespace Catch
04098
04099 #define GENERATE( ... ) \
04100     Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
04101         CATCH_INTERNAL_LINEINFO, \
04102         [ ]{ using namespace Catch::Generators; return makeGenerators( \
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
04103
04104 #define GENERATE_COPY( ... ) \
04105     Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
04106         CATCH_INTERNAL_LINEINFO, \
04107         [=]{ using namespace Catch::Generators; return makeGenerators( \
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
04108
04109 #define GENERATE_REF( ... ) \
04110     Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
04111         CATCH_INTERNAL_LINEINFO, \
04112         [&]{ using namespace Catch::Generators; return makeGenerators( \
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
04113 // end catch_generators.hpp
04114 // start catch_generators_generic.hpp
04115
04116 namespace Catch {
04117     namespace Generators {
04118
04119         template <typename T>
04120         class TakeGenerator : public IGenerator<T> {
04121             GeneratorWrapper<T> m_generator;
04122             size_t m_returned = 0;
04123             size_t m_target;
04124         public:
04125             TakeGenerator(size_t target, GeneratorWrapper<T>&& generator):
04126                 m_generator(std::move(generator)),
04127                 m_target(target)
04128             {
04129                 assert(target != 0 && "Empty generators are not allowed");
04130             }
04131             T const& get() const override {
04132                 return m_generator.get();
04133             }
04134             bool next() override {
04135                 ++m_returned;
04136                 if (m_returned >= m_target) {
04137                     return false;
04138                 }
04139
04140                 const auto success = m_generator.next();
04141                 // If the underlying generator does not contain enough values
04142                 // then we cut short as well
04143                 if (!success) {
04144                     m_returned = m_target;
04145                 }
04146                 return success;
04147             }
04148         };
04149
04150         template <typename T>
04151         GeneratorWrapper<T> take(size_t target, GeneratorWrapper<T>&& generator) {
04152             return GeneratorWrapper<T>(pf::make_unique<TakeGenerator<T>>(target,
std::move(generator)));
04153         }
04154
04155         template <typename T, typename Predicate>
04156         class FilterGenerator : public IGenerator<T> {
04157             GeneratorWrapper<T> m_generator;
04158             Predicate m_predicate;
04159         public:
04160             template <typename P = Predicate>
04161             FilterGenerator(P&& pred, GeneratorWrapper<T>&& generator):
04162                 m_generator(std::move(generator)),

```

```

04163         m_predicate(std::forward<P>(pred))
04164     {
04165         if (!m_predicate(m_generator.get())) {
04166             // It might happen that there are no values that pass the
04167             // filter. In that case we throw an exception.
04168             auto has_initial_value = nextImpl();
04169             if (!has_initial_value) {
04170                 Catch::throw_exception(GeneratorException("No valid value found in filtered
generator"));
04171             }
04172         }
04173     }
04174
04175     T const& get() const override {
04176         return m_generator.get();
04177     }
04178
04179     bool next() override {
04180         return nextImpl();
04181     }
04182
04183 private:
04184     bool nextImpl() {
04185         bool success = m_generator.next();
04186         if (!success) {
04187             return false;
04188         }
04189         while (!m_predicate(m_generator.get()) && (success = m_generator.next()) == true);
04190         return success;
04191     }
04192 };
04193
04194 template <typename T, typename Predicate>
04195 GeneratorWrapper<T> filter(Predicate&& pred, GeneratorWrapper<T>&& generator) {
04196     return
GeneratorWrapper<T>(std::unique_ptr<IGenerator<T>>(pf::make_unique<FilterGenerator<T, Predicate>>(std::forward<Predicate>
std::move(generator))));
04197 }
04198
04199 template <typename T>
04200 class RepeatGenerator : public IGenerator<T> {
04201     static_assert(!std::is_same<T, bool>::value,
04202         "RepeatGenerator currently does not support bools"
04203         "because of std::vector<bool> specialization");
04204     GeneratorWrapper<T> m_generator;
04205     mutable std::vector<T> m_returned;
04206     size_t m_target_repeats;
04207     size_t m_current_repeat = 0;
04208     size_t m_repeat_index = 0;
04209 public:
04210     RepeatGenerator(size_t repeats, GeneratorWrapper<T>&& generator):
04211         m_generator(std::move(generator)),
04212         m_target_repeats(repeats)
04213     {
04214         assert(m_target_repeats > 0 && "Repeat generator must repeat at least once");
04215     }
04216
04217     T const& get() const override {
04218         if (m_current_repeat == 0) {
04219             m_returned.push_back(m_generator.get());
04220             return m_returned.back();
04221         }
04222         return m_returned[m_repeat_index];
04223     }
04224
04225     bool next() override {
04226         // There are 2 basic cases:
04227         // 1) We are still reading the generator
04228         // 2) We are reading our own cache
04229
04230         // In the first case, we need to poke the underlying generator.
04231         // If it happily moves, we are left in that state, otherwise it is time to start
reading from our cache
04232         if (m_current_repeat == 0) {
04233             const auto success = m_generator.next();
04234             if (!success) {
04235                 ++m_current_repeat;
04236             }
04237             return m_current_repeat < m_target_repeats;
04238         }
04239
04240         // In the second case, we need to move indices forward and check that we haven't run
up against the end
04241         ++m_repeat_index;
04242         if (m_repeat_index == m_returned.size()) {
04243             m_repeat_index = 0;
04244             ++m_current_repeat;

```

```

04245         }
04246         return m_current_repeat < m_target_repeats;
04247     }
04248 };
04249
04250 template <typename T>
04251 GeneratorWrapper<T> repeat(size_t repeats, GeneratorWrapper<T>&& generator) {
04252     return GeneratorWrapper<T>(pf::make_unique<RepeatGenerator<T>>(repeats,
std::move(generator)));
04253 }
04254
04255 template <typename T, typename U, typename Func>
04256 class MapGenerator : public IGenerator<T> {
04257     // TBD: provide static assert for mapping function, for friendly error message
04258     GeneratorWrapper<U> m_generator;
04259     Func m_function;
04260     // To avoid returning dangling reference, we have to save the values
04261     T m_cache;
04262 public:
04263     template <typename F2 = Func>
04264     MapGenerator(F2&& function, GeneratorWrapper<U>&& generator) :
04265         m_generator(std::move(generator)),
04266         m_function(std::forward<F2>(function)),
04267         m_cache(m_function(m_generator.get()))
04268     {}
04269
04270     T const& get() const override {
04271         return m_cache;
04272     }
04273     bool next() override {
04274         const auto success = m_generator.next();
04275         if (success) {
04276             m_cache = m_function(m_generator.get());
04277         }
04278         return success;
04279     }
04280 };
04281
04282 template <typename Func, typename U, typename T = FunctionReturnType<Func, U>
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, typename U, typename Func>
04290 GeneratorWrapper<T> map(Func&& function, GeneratorWrapper<U>&& generator) {
04291     return GeneratorWrapper<T>(
04292         pf::make_unique<MapGenerator<T, U, Func>>(std::forward<Func>(function),
std::move(generator))
04293     );
04294 }
04295
04296 template <typename T>
04297 class ChunkGenerator final : public IGenerator<std::vector<T>> {
04298     std::vector<T> m_chunk;
04299     size_t m_chunk_size;
04300     GeneratorWrapper<T> m_generator;
04301     bool m_used_up = false;
04302 public:
04303     ChunkGenerator(size_t size, GeneratorWrapper<T> generator) :
04304         m_chunk_size(size), m_generator(std::move(generator))
04305     {
04306         m_chunk.reserve(m_chunk_size);
04307         if (m_chunk_size != 0) {
04308             m_chunk.push_back(m_generator.get());
04309             for (size_t i = 1; i < m_chunk_size; ++i) {
04310                 if (!m_generator.next()) {
04311                     Catch::throw_exception(GeneratorException("Not enough values to initialize
the first chunk"));
04312                 }
04313                 m_chunk.push_back(m_generator.get());
04314             }
04315         }
04316     }
04317     std::vector<T> const& get() const override {
04318         return m_chunk;
04319     }
04320     bool next() override {
04321         m_chunk.clear();
04322         for (size_t idx = 0; idx < m_chunk_size; ++idx) {
04323             if (!m_generator.next()) {
04324                 return false;
04325             }
04326             m_chunk.push_back(m_generator.get());
04327         }

```

```

04328         return true;
04329     }
04330 };
04331
04332     template <typename T>
04333     GeneratorWrapper<std::vector<T>> chunk(size_t size, GeneratorWrapper<T>&& generator) {
04334         return GeneratorWrapper<std::vector<T>>{
04335             pf::make_unique<ChunkGenerator<T>>(size, std::move(generator))
04336         };
04337     }
04338
04339 } // namespace Generators
04340 } // namespace Catch
04341
04342 // end catch_generators_generic.hpp
04343 // start catch_generators_specific.hpp
04344
04345 // start catch_context.h
04346
04347 #include <memory>
04348
04349 namespace Catch {
04350
04351     struct IResultCapture;
04352     struct IRunner;
04353     struct IConfig;
04354     struct IMutableContext;
04355
04356     using IConfigPtr = std::shared_ptr<IConfig const>;
04357
04358     struct IContext
04359     {
04360         virtual ~IContext();
04361
04362         virtual IResultCapture* getResultCapture() = 0;
04363         virtual IRunner* getRunner() = 0;
04364         virtual IConfigPtr const& getConfig() const = 0;
04365     };
04366
04367     struct IMutableContext : IContext
04368     {
04369         virtual ~IMutableContext();
04370         virtual void setResultCapture( IResultCapture* resultCapture ) = 0;
04371         virtual void setRunner( IRunner* runner ) = 0;
04372         virtual void setConfig( IConfigPtr const& config ) = 0;
04373
04374     private:
04375         static IMutableContext *currentContext;
04376         friend IMutableContext& getCurrentMutableContext();
04377         friend void cleanUpContext();
04378         static void createContext();
04379     };
04380
04381     inline IMutableContext& getCurrentMutableContext()
04382     {
04383         if( !IMutableContext::currentContext )
04384             IMutableContext::createContext();
04385         // NOLINTNEXTLINE(clang-analyzer-core.uninitialized.UndefReturn)
04386         return *IMutableContext::currentContext;
04387     }
04388
04389     inline IContext& getCurrentContext()
04390     {
04391         return getCurrentMutableContext();
04392     }
04393
04394     void cleanUpContext();
04395
04396     class SimplePcg32;
04397     SimplePcg32& rng();
04398 }
04399
04400 // end catch_context.h
04401 // start catch_interfaces_config.h
04402
04403 // start catch_option.hpp
04404
04405 namespace Catch {
04406
04407     // An optional type
04408     template<typename T>
04409     class Option {
04410     public:
04411         Option() : nullableValue( nullptr ) {}
04412         Option( T const& _value )
04413             : nullableValue( new( storage ) T( _value ) )
04414         {}

```

```

04415         Option( Option const& _other )
04416             : nullableValue( _other ? new( storage ) T( *_other ) : nullptr )
04417         {}
04418
04419     ~Option() {
04420         reset();
04421     }
04422
04423     Option& operator= ( Option const& _other ) {
04424         if( &_other != this ) {
04425             reset();
04426             if( _other )
04427                 nullableValue = new( storage ) T( *_other );
04428         }
04429         return *this;
04430     }
04431     Option& operator = ( T const& _value ) {
04432         reset();
04433         nullableValue = new( storage ) T( _value );
04434         return *this;
04435     }
04436
04437     void reset() {
04438         if( nullableValue )
04439             nullableValue->~T();
04440         nullableValue = nullptr;
04441     }
04442
04443     T& operator*() { return *nullableValue; }
04444     T const& operator*() const { return *nullableValue; }
04445     T* operator->() { return nullableValue; }
04446     const T* operator->() const { return nullableValue; }
04447
04448     T valueOr( T const& defaultValue ) const {
04449         return nullableValue ? *nullableValue : defaultValue;
04450     }
04451
04452     bool some() const { return nullableValue != nullptr; }
04453     bool none() const { return nullableValue == nullptr; }
04454
04455     bool operator !() const { return nullableValue == nullptr; }
04456     explicit operator bool() const {
04457         return some();
04458     }
04459
04460     private:
04461         T *nullableValue;
04462         alignas(alignof(T)) char storage[sizeof(T)];
04463     };
04464
04465 } // end namespace Catch
04466
04467 // end catch_option.hpp
04468 #include <chrono>
04469 #include <iosfwd>
04470 #include <string>
04471 #include <vector>
04472 #include <memory>
04473
04474 namespace Catch {
04475
04476     enum class Verbosity {
04477         Quiet = 0,
04478         Normal,
04479         High
04480     };
04481
04482     struct WarnAbout { enum What {
04483         Nothing = 0x00,
04484         NoAssertions = 0x01,
04485         NoTests = 0x02
04486     }; };
04487
04488     struct ShowDurations { enum OrNot {
04489         DefaultForReporter,
04490         Always,
04491         Never
04492     }; };
04493     struct RunTests { enum InWhatOrder {
04494         InDeclarationOrder,
04495         InLexicographicalOrder,
04496         InRandomOrder
04497     }; };
04498     struct UseColour { enum YesOrNo {
04499         Auto,
04500         Yes,
04501         No

```

```

04502     }; };
04503     struct WaitForKeypress { enum When {
04504         Never,
04505         BeforeStart = 1,
04506         BeforeExit = 2,
04507         BeforeStartAndExit = BeforeStart | BeforeExit
04508     }; };
04509
04510     class TestSpec;
04511
04512     struct IConfig : NonCopyable {
04513
04514         virtual ~IConfig();
04515
04516         virtual bool allowThrows() const = 0;
04517         virtual std::ostream& stream() const = 0;
04518         virtual std::string name() const = 0;
04519         virtual bool includeSuccessfulResults() const = 0;
04520         virtual bool shouldDebugBreak() const = 0;
04521         virtual bool warnAboutMissingAssertions() const = 0;
04522         virtual bool warnAboutNoTests() const = 0;
04523         virtual int abortAfter() const = 0;
04524         virtual bool showInvisibles() const = 0;
04525         virtual ShowDurations::OrNot showDurations() const = 0;
04526         virtual double minDuration() const = 0;
04527         virtual TestSpec const& testSpec() const = 0;
04528         virtual bool hasTestFilters() const = 0;
04529         virtual std::vector<std::string> const& getTestsOrTags() const = 0;
04530         virtual RunTests::InWhatOrder runOrder() const = 0;
04531         virtual unsigned int rngSeed() const = 0;
04532         virtual UseColour::YesOrNo useColour() const = 0;
04533         virtual std::vector<std::string> const& getSectionsToRun() const = 0;
04534         virtual Verbosity verbosity() const = 0;
04535
04536         virtual bool benchmarkNoAnalysis() const = 0;
04537         virtual int benchmarkSamples() const = 0;
04538         virtual double benchmarkConfidenceInterval() const = 0;
04539         virtual unsigned int benchmarkResamples() const = 0;
04540         virtual std::chrono::milliseconds benchmarkWarmupTime() const = 0;
04541     };
04542
04543     using IConfigPtr = std::shared_ptr<IConfig const>;
04544 }
04545
04546 // end catch_interfaces_config.h
04547 // start catch_random_number_generator.h
04548
04549 #include <cstdint>
04550
04551 namespace Catch {
04552
04553     // This is a simple implementation of C++11 Uniform Random Number
04554     // Generator. It does not provide all operators, because Catch2
04555     // does not use it, but it should behave as expected inside stdlib's
04556     // distributions.
04557     // The implementation is based on the PCG family (http://pcg-random.org)
04558     class SimplePcg32 {
04559     public:
04560         using state_type = std::uint64_t;
04561         using result_type = std::uint32_t;
04562         static constexpr result_type (min)() {
04563             return 0;
04564         }
04565         static constexpr result_type (max)() {
04566             return static_cast<result_type>(-1);
04567         }
04568
04569         // Provide some default initial state for the default constructor
04570         SimplePcg32():SimplePcg32(0xed743cc4U) {}
04571
04572         explicit SimplePcg32(result_type seed_);
04573
04574         void seed(result_type seed_);
04575         void discard(uint64_t skip);
04576
04577         result_type operator()();
04578
04579     private:
04580         friend bool operator==(SimplePcg32 const& lhs, SimplePcg32 const& rhs);
04581         friend bool operator!=(SimplePcg32 const& lhs, SimplePcg32 const& rhs);
04582
04583         // In theory we also need operator< and operator>
04584         // In practice we do not use them, so we will skip them for now
04585
04586         std::uint64_t m_state;
04587         // This part of the state determines which "stream" of the numbers
04588         // is chosen -- we take it as a constant for Catch2, so we only

```

```

04589         // need to deal with seeding the main state.
04590         // Picked by reading 8 bytes from `/dev/random` :-)
04591         static const std::uint64_t s_inc = (0x13ed0cc53f939476ULL « 1ULL) | 1ULL;
04592     };
04593
04594 } // end namespace Catch
04595
04596 // end catch_random_number_generator.h
04597 #include <random>
04598
04599 namespace Catch {
04600     namespace Generators {
04601
04602         template <typename Float>
04603         class RandomFloatingGenerator final : public IGenerator<Float> {
04604             Catch::SimplePcg32& m_rng;
04605             std::uniform_real_distribution<Float> m_dist;
04606             Float m_current_number;
04607         public:
04608
04609             RandomFloatingGenerator(Float a, Float b):
04610                 m_rng(rng()),
04611                 m_dist(a, b) {
04612                 static_cast<void>(next());
04613             }
04614
04615             Float const& get() const override {
04616                 return m_current_number;
04617             }
04618             bool next() override {
04619                 m_current_number = m_dist(m_rng);
04620                 return true;
04621             }
04622         };
04623
04624         template <typename Integer>
04625         class RandomIntegerGenerator final : public IGenerator<Integer> {
04626             Catch::SimplePcg32& m_rng;
04627             std::uniform_int_distribution<Integer> m_dist;
04628             Integer m_current_number;
04629         public:
04630
04631             RandomIntegerGenerator(Integer a, Integer b):
04632                 m_rng(rng()),
04633                 m_dist(a, b) {
04634                 static_cast<void>(next());
04635             }
04636
04637             Integer const& get() const override {
04638                 return m_current_number;
04639             }
04640             bool next() override {
04641                 m_current_number = m_dist(m_rng);
04642                 return true;
04643             }
04644         };
04645
04646         // TODO: Ideally this would be also constrained against the various char types,
04647         // but I don't expect users to run into that in practice.
04648         template <typename T>
04649         typename std::enable_if<std::is_integral<T>::value && !std::is_same<T, bool>::value,
04650             GeneratorWrapper<T>>::type
04651         random(T a, T b) {
04652             return GeneratorWrapper<T>(
04653                 pf::make_unique<RandomIntegerGenerator<T>>(a, b)
04654             );
04655         }
04656
04657         template <typename T>
04658         typename std::enable_if<std::is_floating_point<T>::value,
04659             GeneratorWrapper<T>>::type
04660         random(T a, T b) {
04661             return GeneratorWrapper<T>(
04662                 pf::make_unique<RandomFloatingGenerator<T>>(a, b)
04663             );
04664         }
04665
04666         template <typename T>
04667         class RangeGenerator final : public IGenerator<T> {
04668             T m_current;
04669             T m_end;
04670             T m_step;
04671             bool m_positive;
04672         public:
04673             RangeGenerator(T const& start, T const& end, T const& step):
04674                 m_current(start),

```

```

04676         m_end(end),
04677         m_step(step),
04678         m_positive(m_step > T(0))
04679     {
04680         assert(m_current != m_end && "Range start and end cannot be equal");
04681         assert(m_step != T(0) && "Step size cannot be zero");
04682         assert((m_positive && m_current <= m_end) || (!m_positive && m_current >= m_end)) &&
"Step moves away from end");
04683     }
04684
04685     RangeGenerator(T const& start, T const& end):
04686         RangeGenerator(start, end, (start < end) ? T(1) : T(-1))
04687     {}
04688
04689     T const& get() const override {
04690         return m_current;
04691     }
04692
04693     bool next() override {
04694         m_current += m_step;
04695         return (m_positive) ? (m_current < m_end) : (m_current > m_end);
04696     }
04697 };
04698
04699 template <typename T>
04700 GeneratorWrapper<T> range(T const& start, T const& end, T const& step) {
04701     static_assert(std::is_arithmetic<T>::value && !std::is_same<T, bool>::value, "Type must be
numeric");
04702     return GeneratorWrapper<T>(pf::make_unique<RangeGenerator<T>>(start, end, step));
04703 }
04704
04705 template <typename T>
04706 GeneratorWrapper<T> range(T const& start, T const& end) {
04707     static_assert(std::is_integral<T>::value && !std::is_same<T, bool>::value, "Type must be
an integer");
04708     return GeneratorWrapper<T>(pf::make_unique<RangeGenerator<T>>(start, end));
04709 }
04710
04711 template <typename T>
04712 class IteratorGenerator final : public IGenerator<T> {
04713     static_assert(!std::is_same<T, bool>::value,
04714         "IteratorGenerator currently does not support bools"
04715         "because of std::vector<bool> specialization");
04716
04717     std::vector<T> m_elems;
04718     size_t m_current = 0;
04719 public:
04720     template <typename InputIterator, typename InputSentinel>
04721     IteratorGenerator(InputIterator first, InputSentinel last):m_elems(first, last) {
04722         if (m_elems.empty()) {
04723             Catch::throw_exception(GeneratorException("IteratorGenerator received no valid
values"));
04724         }
04725     }
04726
04727     T const& get() const override {
04728         return m_elems[m_current];
04729     }
04730
04731     bool next() override {
04732         ++m_current;
04733         return m_current != m_elems.size();
04734     }
04735 };
04736
04737 template <typename InputIterator,
04738     typename InputSentinel,
04739     typename ResultType = typename std::iterator_traits<InputIterator>::value_type>
04740 GeneratorWrapper<ResultType> from_range(InputIterator from, InputSentinel to) {
04741     return GeneratorWrapper<ResultType>(pf::make_unique<IteratorGenerator<ResultType>>(from,
to));
04742 }
04743
04744 template <typename Container,
04745     typename ResultType = typename Container::value_type>
04746 GeneratorWrapper<ResultType> from_range(Container const& cnt) {
04747     return
GeneratorWrapper<ResultType>(pf::make_unique<IteratorGenerator<ResultType>>(cnt.begin(), cnt.end()));
04748 }
04749
04750 } // namespace Generators
04751 } // namespace Catch
04752
04753 // end catch_generators_specific.hpp
04754
04755 // These files are included here so the single_include script doesn't put them
04756 // in the conditionally compiled sections

```



```

04757 // start catch_test_case_info.h
04758
04759 #include <string>
04760 #include <vector>
04761 #include <memory>
04762
04763 #ifdef __clang__
04764 #pragma clang diagnostic push
04765 #pragma clang diagnostic ignored "-Wpadded"
04766 #endif
04767
04768 namespace Catch {
04769
04770     struct ITestInvoker;
04771
04772     struct TestCaseInfo {
04773         enum SpecialProperties{
04774             None = 0,
04775             IsHidden = 1 << 1,
04776             ShouldFail = 1 << 2,
04777             MayFail = 1 << 3,
04778             Throws = 1 << 4,
04779             NonPortable = 1 << 5,
04780             Benchmark = 1 << 6
04781         };
04782
04783         TestCaseInfo(    std::string const& _name,
04784                         std::string const& _className,
04785                         std::string const& _description,
04786                         std::vector<std::string> const& _tags,
04787                         SourceLineInfo const& _lineInfo );
04788
04789         friend void setTags( TestCaseInfo& testCaseInfo, std::vector<std::string> tags );
04790
04791         bool isHidden() const;
04792         bool throws() const;
04793         bool okToFail() const;
04794         bool expectedToFail() const;
04795
04796         std::string tagsAsString() const;
04797
04798         std::string name;
04799         std::string className;
04800         std::string description;
04801         std::vector<std::string> tags;
04802         std::vector<std::string> lcTags;
04803         SourceLineInfo lineInfo;
04804         SpecialProperties properties;
04805     };
04806
04807     class TestCase : public TestCaseInfo {
04808     public:
04809
04810         TestCase( ITestInvoker* testInvoker, TestCaseInfo&& info );
04811
04812         TestCase withName( std::string const& _newName ) const;
04813
04814         void invoke() const;
04815
04816         TestCaseInfo const& getTestCaseInfo() const;
04817
04818         bool operator == ( TestCase const& other ) const;
04819         bool operator < ( TestCase const& other ) const;
04820
04821     private:
04822         std::shared_ptr<ITestInvoker> test;
04823     };
04824
04825     TestCase makeTestCase( ITestInvoker* testInvoker,
04826                           std::string const& className,
04827                           NameAndTags const& nameAndTags,
04828                           SourceLineInfo const& lineInfo );
04829 }
04830
04831 #ifdef __clang__
04832 #pragma clang diagnostic pop
04833 #endif
04834
04835 // end catch_test_case_info.h
04836 // start catch_interfaces_runner.h
04837
04838 namespace Catch {
04839
04840     struct IRunner {
04841         virtual ~IRunner();
04842         virtual bool aborting() const = 0;
04843     };

```

```

04844 }
04845
04846 // end catch_interfaces_runner.h
04847
04848 #ifdef __OBJC__
04849 // start catch_objc.hpp
04850
04851 #import <objc/runtime.h>
04852
04853 #include <string>
04854
04855 // NB. Any general catch headers included here must be included
04856 // in catch.hpp first to make sure they are included by the single
04857 // header for non objc-usage
04858
04860 // This protocol is really only here for (self) documenting purposes, since
04861 // all its methods are optional.
04862 @protocol OcFixture
04863
04864 @optional
04865
04866 -(void) setUp;
04867 -(void) tearDown;
04868
04869 @end
04870
04871 namespace Catch {
04872
04873     class OcMethod : public ITestInvoker {
04874     public:
04875         OcMethod( Class cls, SEL sel ) : m_cls( cls ), m_sel( sel ) {}
04876
04877         virtual void invoke() const {
04878             id obj = [[m_cls alloc] init];
04879
04880             performOptionalSelector( obj, @selector(setUp) );
04881             performOptionalSelector( obj, m_sel );
04882             performOptionalSelector( obj, @selector(tearDown) );
04883
04884             arcSafeRelease( obj );
04885         }
04886     private:
04887         virtual ~OcMethod() {}
04888
04889         Class m_cls;
04890         SEL m_sel;
04891     };
04892
04893     namespace Detail{
04894
04895         inline std::string getAnnotation( Class cls,
04896                                         std::string const& annotationName,
04897                                         std::string const& testCaseName ) {
04898             NSString* selStr = [[NSString alloc] initWithFormat:@"Catch_%s_%s",
04899                             annotationName.c_str(), testCaseName.c_str()];
04900             SEL sel = NSSelectorFromString( selStr );
04901             arcSafeRelease( selStr );
04902             id value = performOptionalSelector( cls, sel );
04903             if( value )
04904                 return [(NSString*)value UTF8String];
04905             return "";
04906         }
04907     }
04908
04909     inline std::size_t registerTestMethods() {
04910         std::size_t noTestMethods = 0;
04911         int noClasses = objc_getClassList( nullptr, 0 );
04912
04913         Class* classes = (CATCH_UNSAFE_UNRETAINED Class *)malloc( sizeof(Class) * noClasses);
04914         objc_getClassList( classes, noClasses );
04915
04916         for( int c = 0; c < noClasses; c++ ) {
04917             Class cls = classes[c];
04918             {
04919                 u_int count;
04920                 Method* methods = class_copyMethodList( cls, &count );
04921                 for( u_int m = 0; m < count ; m++ ) {
04922                     SEL selector = method_getName(methods[m]);
04923                     std::string methodName = sel_getName(selector);
04924                     if( startsWith( methodName, "Catch_TestCase_" ) ) {
04925                         std::string testCaseName = methodName.substr( 15 );
04926                         std::string name = Detail::getAnnotation( cls, "Name", testCaseName );
04927                         std::string desc = Detail::getAnnotation( cls, "Description", testCaseName );
04928                         const char* className = class_getName( cls );
04929
04930                         getMutableRegistryHub().registerTest( makeTestCase( new OcMethod( cls,

```

```

        selector ), className, NameAndTags( name.c_str(), desc.c_str() ), SourceLineInfo("",0) ) );
04931         noTestMethods++;
04932     }
04933     }
04934     free(methods);
04935 }
04936 }
04937     return noTestMethods;
04938 }
04939
04940 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
04941
04942     namespace Matchers {
04943         namespace Impl {
04944             namespace NSStringMatchers {
04945
04946                 struct StringHolder : MatcherBase<NSString*>{
04947                     StringHolder( NSString* substr ) : m_substr( [substr copy] ){}
04948                     StringHolder( StringHolder const& other ) : m_substr( [other.m_substr copy] ){}
04949                     StringHolder() {
04950                         arcSafeRelease( m_substr );
04951                     }
04952
04953                     bool match( NSString* str ) const override {
04954                         return false;
04955                     }
04956
04957                     NSString* CATCH_ARC_STRONG m_substr;
04958                 };
04959
04960                 struct Equals : StringHolder {
04961                     Equals( NSString* substr ) : StringHolder( substr ){}
04962
04963                     bool match( NSString* str ) const override {
04964                         return (str != nil || m_substr == nil ) &&
04965                             [str isEqualToString:m_substr];
04966                     }
04967
04968                     std::string describe() const override {
04969                         return "equals string: " + Catch::Detail::stringify( m_substr );
04970                     }
04971                 };
04972
04973                 struct Contains : StringHolder {
04974                     Contains( NSString* substr ) : StringHolder( substr ){}
04975
04976                     bool match( NSString* str ) const override {
04977                         return (str != nil || m_substr == nil ) &&
04978                             [str rangeOfString:m_substr].location != NSNotFound;
04979                     }
04980
04981                     std::string describe() const override {
04982                         return "contains string: " + Catch::Detail::stringify( m_substr );
04983                     }
04984                 };
04985
04986                 struct StartsWith : StringHolder {
04987                     StartsWith( NSString* substr ) : StringHolder( substr ){}
04988
04989                     bool match( NSString* str ) const override {
04990                         return (str != nil || m_substr == nil ) &&
04991                             [str rangeOfString:m_substr].location == 0;
04992                     }
04993
04994                     std::string describe() const override {
04995                         return "starts with: " + Catch::Detail::stringify( m_substr );
04996                     }
04997                 };
04998                 struct EndsWith : StringHolder {
04999                     EndsWith( NSString* substr ) : StringHolder( substr ){}
05000
05001                     bool match( NSString* str ) const override {
05002                         return (str != nil || m_substr == nil ) &&
05003                             [str rangeOfString:m_substr].location == [str length] - [m_substr length];
05004                     }
05005
05006                     std::string describe() const override {
05007                         return "ends with: " + Catch::Detail::stringify( m_substr );
05008                     }
05009                 };
05010
05011             } // namespace NSStringMatchers
05012         } // namespace Impl
05013
05014         inline Impl::NSStringMatchers::Equals
05015             Equals( NSString* substr ){ return Impl::NSStringMatchers::Equals( substr ); }
05016

```

```

05017         inline Impl::NSStringMatchers::Contains
05018             Contains( NSString* substr ){ return Impl::NSStringMatchers::Contains( substr ); }
05019
05020         inline Impl::NSStringMatchers::StartsWith
05021             StartsWith( NSString* substr ){ return Impl::NSStringMatchers::StartsWith( substr ); }
05022
05023         inline Impl::NSStringMatchers::EndsWith
05024             EndsWith( NSString* substr ){ return Impl::NSStringMatchers::EndsWith( substr ); }
05025
05026     } // namespace Matchers
05027
05028     using namespace Matchers;
05029
05029 #endif // CATCH_CONFIG_DISABLE_MATCHERS
05031
05032 } // namespace Catch
05033
05035 #define OC_MAKE_UNIQUE_NAME( root, uniqueSuffix ) root##uniqueSuffix
05036 #define OC_TEST_CASE2( name, desc, uniqueSuffix ) \
05037 +(NSString*) OC_MAKE_UNIQUE_NAME( Catch_Name_test_, uniqueSuffix ) \
05038 { \
05039 return @ name; \
05040 } \
05041 +(NSString*) OC_MAKE_UNIQUE_NAME( Catch_Description_test_, uniqueSuffix ) \
05042 { \
05043 return @ desc; \
05044 } \
05045 -(void) OC_MAKE_UNIQUE_NAME( Catch_TestCase_test_, uniqueSuffix )
05046
05047 #define OC_TEST_CASE( name, desc ) OC_TEST_CASE2( name, desc, __LINE__ )
05048
05049 // end catch_objc.hpp
05050 #endif
05051
05052 // Benchmarking needs the externally-facing parts of reporters to work
05053 #if defined(CATCH_CONFIG_EXTERNAL_INTERFACES) || defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
05054 // start catch_external_interfaces.h
05055
05056 // start catch_reporter_bases.hpp
05057
05058 // start catch_interfaces_reporter.h
05059
05060 // start catch_config.hpp
05061
05062 // start catch_test_spec_parser.h
05063
05064 #ifdef __clang__
05065 #pragma clang diagnostic push
05066 #pragma clang diagnostic ignored "-Wpadded"
05067 #endif
05068
05069 // start catch_test_spec.h
05070
05071 #ifdef __clang__
05072 #pragma clang diagnostic push
05073 #pragma clang diagnostic ignored "-Wpadded"
05074 #endif
05075
05076 // start catch_wildcard_pattern.h
05077
05078 namespace Catch
05079 {
05080     class WildcardPattern {
05081     public:
05082         enum WildcardPosition {
05083             NoWildcard = 0,
05084             WildcardAtStart = 1,
05085             WildcardAtEnd = 2,
05086             WildcardAtBothEnds = WildcardAtStart | WildcardAtEnd
05087         };
05088
05089         WildcardPattern( std::string const& pattern, CaseSensitive::Choice caseSensitivity );
05090         virtual ~WildcardPattern() = default;
05091         virtual bool matches( std::string const& str ) const;
05092
05093     private:
05094         std::string normaliseString( std::string const& str ) const;
05095         CaseSensitive::Choice m_caseSensitivity;
05096         WildcardPosition m_wildcard = NoWildcard;
05097         std::string m_pattern;
05098     };
05099 }
05100
05101 // end catch_wildcard_pattern.h
05102 #include <string>
05103 #include <vector>

```

```

05105 #include <memory>
05106
05107 namespace Catch {
05108
05109     struct IConfig;
05110
05111     class TestSpec {
05112     public:
05113         explicit TestSpec( std::string const& name );
05114         virtual ~TestSpec();
05115         virtual bool matches( TestCaseInfo const& testCase ) const = 0;
05116         std::string const& name() const;
05117     private:
05118         std::string const m_name;
05119     };
05120
05121     using PatternPtr = std::shared_ptr<Pattern>;
05122
05123     class NamePattern : public Pattern {
05124     public:
05125         explicit NamePattern( std::string const& name, std::string const& filterString );
05126         bool matches( TestCaseInfo const& testCase ) const override;
05127     private:
05128         WildcardPattern m_wildcardPattern;
05129     };
05130
05131     class TagPattern : public Pattern {
05132     public:
05133         explicit TagPattern( std::string const& tag, std::string const& filterString );
05134         bool matches( TestCaseInfo const& testCase ) const override;
05135     private:
05136         std::string m_tag;
05137     };
05138
05139     class ExcludedPattern : public Pattern {
05140     public:
05141         explicit ExcludedPattern( PatternPtr const& underlyingPattern );
05142         bool matches( TestCaseInfo const& testCase ) const override;
05143     private:
05144         PatternPtr m_underlyingPattern;
05145     };
05146
05147     struct Filter {
05148         std::vector<PatternPtr> m_patterns;
05149
05150         bool matches( TestCaseInfo const& testCase ) const;
05151         std::string name() const;
05152     };
05153
05154     public:
05155         struct FilterMatch {
05156             std::string name;
05157             std::vector<TestCase const*> tests;
05158         };
05159         using Matches = std::vector<FilterMatch>;
05160         using vectorStrings = std::vector<std::string>;
05161
05162         bool hasFilters() const;
05163         bool matches( TestCaseInfo const& testCase ) const;
05164         Matches matchesByFilter( std::vector<TestCase> const& testCases, IConfig const& config )
05165     const;
05166         const vectorStrings & getInvalidArgs() const;
05167     private:
05168         std::vector<Filter> m_filters;
05169         std::vector<std::string> m_invalidArgs;
05170         friend class TestSpecParser;
05171     };
05172 }
05173
05174 #ifdef __clang__
05175 #pragma clang diagnostic pop
05176 #endif
05177
05178 // end catch_test_spec.h
05179 // start catch_interfaces_tag_alias_registry.h
05180
05181 #include <string>
05182
05183 namespace Catch {
05184
05185     struct TagAlias;
05186
05187     struct ITagAliasRegistry {
05188         virtual ~ITagAliasRegistry();
05189         // Nullptr if not present
05190         virtual TagAlias const* find( std::string const& alias ) const = 0;

```

```

05191         virtual std::string expandAliases( std::string const& unexpandedTestSpec ) const = 0;
05192
05193         static ITagAliasRegistry const& get();
05194     };
05195
05196 } // end namespace Catch
05197
05198 // end catch_interfaces_tag_alias_registry.h
05199 namespace Catch {
05200
05201     class TestSpecParser {
05202     public:
05203         enum Mode{ None, Name, QuotedName, Tag, EscapedName };
05204         Mode m_mode = None;
05205         Mode lastMode = None;
05206         bool m_exclusion = false;
05207         std::size_t m_pos = 0;
05208         std::size_t m_realPatternPos = 0;
05209         std::string m_arg;
05210         std::string m_substring;
05211         std::string m_patternName;
05212         std::vector<std::size_t> m_escapeChars;
05213         TestSpec::Filter m_currentFilter;
05214         TestSpec m_testSpec;
05215         ITagAliasRegistry const* m_tagAliases = nullptr;
05216
05217     public:
05218         TestSpecParser( ITagAliasRegistry const& tagAliases );
05219
05220         TestSpecParser& parse( std::string const& arg );
05221         TestSpec testSpec();
05222
05223     private:
05224         bool visitChar( char c );
05225         void startNewMode( Mode mode );
05226         bool processNoneChar( char c );
05227         void processNameChar( char c );
05228         bool processOtherChar( char c );
05229         void endMode();
05230         void escape();
05231         bool isControlChar( char c ) const;
05232         void saveLastMode();
05233         void revertBackToLastMode();
05234         void addFilter();
05235         bool separate();
05236
05237         // Handles common preprocessing of the pattern for name/tag patterns
05238         std::string preprocessPattern();
05239         // Adds the current pattern as a test name
05240         void addNamePattern();
05241         // Adds the current pattern as a tag
05242         void addTagPattern();
05243
05244         inline void addCharToPattern(char c) {
05245             m_substring += c;
05246             m_patternName += c;
05247             m_realPatternPos++;
05248         }
05249     };
05250
05251     TestSpec parseTestSpec( std::string const& arg );
05252 } // namespace Catch
05253
05254 #ifdef __clang__
05255 #pragma clang diagnostic pop
05256 #endif
05257
05258 // end catch_test_spec_parser.h
05259 // Libstdc++ doesn't like incomplete classes for unique_ptr
05260
05261 #include <memory>
05262 #include <vector>
05263 #include <string>
05264
05265 #ifndef CATCH_CONFIG_CONSOLE_WIDTH
05266 #define CATCH_CONFIG_CONSOLE_WIDTH 80
05267 #endif
05268
05269 namespace Catch {
05270
05271     struct IStream;
05272
05273     struct ConfigData {
05274         bool listTests = false;
05275         bool listTags = false;
05276         bool listReporters = false;
05277         bool listTestNamesOnly = false;

```

```

05278
05279     bool showSuccessfulTests = false;
05280     bool shouldDebugBreak = false;
05281     bool noThrow = false;
05282     bool showHelp = false;
05283     bool showInvisibles = false;
05284     bool filenamesAsTags = false;
05285     bool libIdentify = false;
05286
05287     int abortAfter = -1;
05288     unsigned int rngSeed = 0;
05289
05290     bool benchmarkNoAnalysis = false;
05291     unsigned int benchmarkSamples = 100;
05292     double benchmarkConfidenceInterval = 0.95;
05293     unsigned int benchmarkResamples = 100000;
05294     std::chrono::milliseconds::rep benchmarkWarmupTime = 100;
05295
05296     Verbosity verbosity = Verbosity::Normal;
05297     WarnAbout::What warnings = WarnAbout::Nothing;
05298     ShowDurations::OrNot showDurations = ShowDurations::DefaultForReporter;
05299     double minDuration = -1;
05300     RunTests::InWhatOrder runOrder = RunTests::InDeclarationOrder;
05301     UseColour::YesOrNo useColour = UseColour::Auto;
05302     WaitForKeypress::When waitForKeypress = WaitForKeypress::Never;
05303
05304     std::string outputFilename;
05305     std::string name;
05306     std::string processName;
05307 #ifndef CATCH_CONFIG_DEFAULT_REPORTER
05308 #define CATCH_CONFIG_DEFAULT_REPORTER "console"
05309 #endif
05310     std::string reporterName = CATCH_CONFIG_DEFAULT_REPORTER;
05311 #undef CATCH_CONFIG_DEFAULT_REPORTER
05312
05313     std::vector<std::string> testsOrTags;
05314     std::vector<std::string> sectionsToRun;
05315 };
05316
05317 class Config : public IConfig {
05318 public:
05319
05320     Config() = default;
05321     Config( ConfigData const& data );
05322     virtual ~Config() = default;
05323
05324     std::string const& getFilename() const;
05325
05326     bool listTests() const;
05327     bool listTestNamesOnly() const;
05328     bool listTags() const;
05329     bool listReporters() const;
05330
05331     std::string getProcessName() const;
05332     std::string const& getReporterName() const;
05333
05334     std::vector<std::string> const& getTestsOrTags() const override;
05335     std::vector<std::string> const& getSectionsToRun() const override;
05336
05337     TestSpec const& testSpec() const override;
05338     bool hasTestFilters() const override;
05339
05340     bool showHelp() const;
05341
05342     // IConfig interface
05343     bool allowThrows() const override;
05344     std::ostream& stream() const override;
05345     std::string name() const override;
05346     bool includeSuccessfulResults() const override;
05347     bool warnAboutMissingAssertions() const override;
05348     bool warnAboutNoTests() const override;
05349     ShowDurations::OrNot showDurations() const override;
05350     double minDuration() const override;
05351     RunTests::InWhatOrder runOrder() const override;
05352     unsigned int rngSeed() const override;
05353     UseColour::YesOrNo useColour() const override;
05354     bool shouldDebugBreak() const override;
05355     int abortAfter() const override;
05356     bool showInvisibles() const override;
05357     Verbosity verbosity() const override;
05358     bool benchmarkNoAnalysis() const override;
05359     int benchmarkSamples() const override;
05360     double benchmarkConfidenceInterval() const override;
05361     unsigned int benchmarkResamples() const override;
05362     std::chrono::milliseconds benchmarkWarmupTime() const override;
05363
05364 private:

```

```

05365
05366     IStream const* openStream();
05367     ConfigData m_data;
05368
05369     std::unique_ptr<IStream const> m_stream;
05370     TestSpec m_testSpec;
05371     bool m_hasTestFilters = false;
05372 };
05373
05374 } // end namespace Catch
05375
05376 // end catch_config.hpp
05377 // start catch_assertionresult.h
05378
05379 #include <string>
05380
05381 namespace Catch {
05382
05383     struct AssertionResultData
05384     {
05385         AssertionResultData() = delete;
05386
05387         AssertionResultData( ResultWas::OfType _resultType, LazyExpression const& _lazyExpression );
05388
05389         std::string message;
05390         mutable std::string reconstructedExpression;
05391         LazyExpression lazyExpression;
05392         ResultWas::OfType resultType;
05393
05394         std::string reconstructExpression() const;
05395     };
05396
05397     class AssertionResult {
05398     public:
05399         AssertionResult() = delete;
05400         AssertionResult( AssertionInfo const& info, AssertionResultData const& data );
05401
05402         bool isOk() const;
05403         bool succeeded() const;
05404         ResultWas::OfType getResultType() const;
05405         bool hasExpression() const;
05406         bool hasMessage() const;
05407         std::string getExpression() const;
05408         std::string getExpressionInMacro() const;
05409         bool hasExpandedExpression() const;
05410         std::string getExpandedExpression() const;
05411         std::string getMessage() const;
05412         SourceLineInfo getSourceInfo() const;
05413        StringRef getTestMacroName() const;
05414
05415         //protected:
05416         AssertionInfo m_info;
05417         AssertionResultData m_resultData;
05418     };
05419
05420 } // end namespace Catch
05421
05422 // end catch_assertionresult.h
05423 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
05424 // start catch_estimate.hpp
05425
05426 // Statistics estimates
05427
05428
05429 namespace Catch {
05430     namespace Benchmark {
05431         template <typename Duration>
05432         struct Estimate {
05433             Duration point;
05434             Duration lower_bound;
05435             Duration upper_bound;
05436             double confidence_interval;
05437
05438             template <typename Duration2>
05439             operator Estimate<Duration2>() const {
05440                 return { point, lower_bound, upper_bound, confidence_interval };
05441             }
05442         };
05443     } // namespace Benchmark
05444 } // namespace Catch
05445
05446 // end catch_estimate.hpp
05447 // start catch_outlier_classification.hpp
05448
05449 // Outlier information
05450
05451 namespace Catch {

```



```

05452     namespace Benchmark {
05453         struct OutlierClassification {
05454             int samples_seen = 0;
05455             int low_severe = 0;    // more than 3 times IQR below Q1
05456             int low_mild = 0;     // 1.5 to 3 times IQR below Q1
05457             int high_mild = 0;    // 1.5 to 3 times IQR above Q3
05458             int high_severe = 0;  // more than 3 times IQR above Q3
05459
05460             int total() const {
05461                 return low_severe + low_mild + high_mild + high_severe;
05462             }
05463         };
05464     } // namespace Benchmark
05465 } // namespace Catch
05466
05467 // end catch_outlier_classification.hpp
05468
05469 #include <iterator>
05470 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
05471
05472 #include <string>
05473 #include <iosfwd>
05474 #include <map>
05475 #include <set>
05476 #include <memory>
05477 #include <algorithm>
05478
05479 namespace Catch {
05480
05481     struct ReporterConfig {
05482         explicit ReporterConfig( IConfigPtr const& _fullConfig );
05483
05484         ReporterConfig( IConfigPtr const& _fullConfig, std::ostream& _stream );
05485
05486         std::ostream& stream() const;
05487         IConfigPtr fullConfig() const;
05488
05489     private:
05490         std::ostream* m_stream;
05491         IConfigPtr m_fullConfig;
05492     };
05493
05494     struct ReporterPreferences {
05495         bool shouldRedirectStdOut = false;
05496         bool shouldReportAllAssertions = false;
05497     };
05498
05499     template<typename T>
05500     struct LazyStat : Option<T> {
05501         LazyStat& operator=( T const& _value ) {
05502             Option<T>::operator=( _value );
05503             used = false;
05504             return *this;
05505         }
05506         void reset() {
05507             Option<T>::reset();
05508             used = false;
05509         }
05510         bool used = false;
05511     };
05512
05513     struct TestRunInfo {
05514         TestRunInfo( std::string const& _name );
05515         std::string name;
05516     };
05517
05518     struct GroupInfo {
05519         GroupInfo( std::string const& _name,
05520                   std::size_t _groupIndex,
05521                   std::size_t _groupsCount );
05522
05523         std::string name;
05524         std::size_t groupIndex;
05525         std::size_t groupsCounts;
05526     };
05527
05528     struct AssertionStats {
05529         AssertionStats( AssertionResult const& _assertionResult,
05530                         std::vector<MessageInfo> const& _infoMessages,
05531                         Totals const& _totals );
05532
05533         AssertionStats( AssertionStats const& ) = default;
05534         AssertionStats( AssertionStats && ) = default;
05535         AssertionStats& operator = ( AssertionStats const& ) = delete;
05536         AssertionStats& operator = ( AssertionStats && ) = delete;
05537         virtual ~AssertionStats();
05538
05539         AssertionResult assertionResult;

```

```

05539         std::vector<MessageInfo> infoMessages;
05540         Totals totals;
05541     };
05542
05543     struct SectionStats {
05544         SectionStats( SectionInfo const& _sectionInfo,
05545             Counts const& _assertions,
05546             double _durationInSeconds,
05547             bool _missingAssertions );
05548         SectionStats( SectionStats const& ) = default;
05549         SectionStats( SectionStats && ) = default;
05550         SectionStats& operator = ( SectionStats const& ) = default;
05551         SectionStats& operator = ( SectionStats && ) = default;
05552         virtual ~SectionStats();
05553
05554         SectionInfo sectionInfo;
05555         Counts assertions;
05556         double durationInSeconds;
05557         bool missingAssertions;
05558     };
05559
05560     struct TestCaseStats {
05561         TestCaseStats( TestCaseInfo const& _testInfo,
05562             Totals const& _totals,
05563             std::string const& _stdOut,
05564             std::string const& _stdErr,
05565             bool _aborting );
05566
05567         TestCaseStats( TestCaseStats const& ) = default;
05568         TestCaseStats( TestCaseStats && ) = default;
05569         TestCaseStats& operator = ( TestCaseStats const& ) = default;
05570         TestCaseStats& operator = ( TestCaseStats && ) = default;
05571         virtual ~TestCaseStats();
05572
05573         TestCaseInfo testInfo;
05574         Totals totals;
05575         std::string stdOut;
05576         std::string stdErr;
05577         bool aborting;
05578     };
05579
05580     struct TestGroupStats {
05581         TestGroupStats( GroupInfo const& _groupInfo,
05582             Totals const& _totals,
05583             bool _aborting );
05584         TestGroupStats( GroupInfo const& _groupInfo );
05585
05586         TestGroupStats( TestGroupStats const& ) = default;
05587         TestGroupStats( TestGroupStats && ) = default;
05588         TestGroupStats& operator = ( TestGroupStats const& ) = default;
05589         TestGroupStats& operator = ( TestGroupStats && ) = default;
05590         virtual ~TestGroupStats();
05591
05592         GroupInfo groupInfo;
05593         Totals totals;
05594         bool aborting;
05595     };
05596
05597     struct TestRunStats {
05598         TestRunStats( TestRunInfo const& _runInfo,
05599             Totals const& _totals,
05600             bool _aborting );
05601
05602         TestRunStats( TestRunStats const& ) = default;
05603         TestRunStats( TestRunStats && ) = default;
05604         TestRunStats& operator = ( TestRunStats const& ) = default;
05605         TestRunStats& operator = ( TestRunStats && ) = default;
05606         virtual ~TestRunStats();
05607
05608         TestRunInfo runInfo;
05609         Totals totals;
05610         bool aborting;
05611     };
05612
05613 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
05614     struct BenchmarkInfo {
05615         std::string name;
05616         double estimatedDuration;
05617         int iterations;
05618         int samples;
05619         unsigned int resamples;
05620         double clockResolution;
05621         double clockCost;
05622     };
05623
05624     template <class Duration>
05625     struct BenchmarkStats {

```

```

05626     BenchmarkInfo info;
05627
05628     std::vector<Duration> samples;
05629     Benchmark::Estimate<Duration> mean;
05630     Benchmark::Estimate<Duration> standardDeviation;
05631     Benchmark::OutlierClassification outliers;
05632     double outlierVariance;
05633
05634     template <typename Duration2>
05635     operator BenchmarkStats<Duration2>() const {
05636         std::vector<Duration2> samples2;
05637         samples2.reserve(samples.size());
05638         std::transform(samples.begin(), samples.end(), std::back_inserter(samples2), [](Duration
05639 d) { return Duration2(d); });
05640         return {
05641             info,
05642             std::move(samples2),
05643             mean,
05644             standardDeviation,
05645             outliers,
05646             outlierVariance,
05647         };
05648     };
05649 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
05650
05651     struct IStreamingReporter {
05652         virtual ~IStreamingReporter() = default;
05653
05654         // Implementing class must also provide the following static methods:
05655         // static std::string getDescription();
05656         // static std::set<Verbosity> getSupportedVerbsities()
05657
05658         virtual ReporterPreferences getPreferences() const = 0;
05659
05660         virtual void noMatchingTestCases( std::string const& spec ) = 0;
05661
05662         virtual void reportInvalidArguments(std::string const&) {}
05663
05664         virtual void testRunStarting( TestRunInfo const& testRunInfo ) = 0;
05665         virtual void testGroupStarting( GroupInfo const& groupInfo ) = 0;
05666
05667         virtual void testCaseStarting( TestCaseInfo const& testInfo ) = 0;
05668         virtual void sectionStarting( SectionInfo const& sectionInfo ) = 0;
05669
05670 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
05671         virtual void benchmarkPreparing( std::string const& ) {}
05672         virtual void benchmarkStarting( BenchmarkInfo const& ) {}
05673         virtual void benchmarkEnded( BenchmarkStats<> const& ) {}
05674         virtual void benchmarkFailed( std::string const& ) {}
05675 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
05676
05677         virtual void assertionStarting( AssertionInfo const& assertionInfo ) = 0;
05678
05679         // The return value indicates if the messages buffer should be cleared:
05680         virtual bool assertionEnded( AssertionStats const& assertionStats ) = 0;
05681
05682         virtual void sectionEnded( SectionStats const& sectionStats ) = 0;
05683         virtual void testCaseEnded( TestCaseStats const& testCaseStats ) = 0;
05684         virtual void testGroupEnded( TestGroupStats const& testGroupStats ) = 0;
05685         virtual void testRunEnded( TestRunStats const& testRunStats ) = 0;
05686
05687         virtual void skipTest( TestCaseInfo const& testInfo ) = 0;
05688
05689         // Default empty implementation provided
05690         virtual void fatalErrorEncountered( StringRef name );
05691
05692         virtual bool isMulti() const;
05693     };
05694     using IStreamingReporterPtr = std::unique_ptr<IStreamingReporter>;
05695
05696     struct IReporterFactory {
05697         virtual ~IReporterFactory();
05698         virtual IStreamingReporterPtr create( ReporterConfig const& config ) const = 0;
05699         virtual std::string getDescription() const = 0;
05700     };
05701     using IReporterFactoryPtr = std::shared_ptr<IReporterFactory>;
05702
05703     struct IReporterRegistry {
05704         using FactoryMap = std::map<std::string, IReporterFactoryPtr>;
05705         using Listeners = std::vector<IReporterFactoryPtr>;
05706
05707         virtual ~IReporterRegistry();
05708         virtual IStreamingReporterPtr create( std::string const& name, IConfigPtr const& config )
05709 const = 0;
05709         virtual FactoryMap const& getFactories() const = 0;
05710         virtual Listeners const& getListeners() const = 0;

```

```

05711     };
05712
05713 } // end namespace Catch
05714
05715 // end catch_interfaces_reporter.h
05716 #include <algorithm>
05717 #include <cstring>
05718 #include <cfloat>
05719 #include <cstdio>
05720 #include <cassert>
05721 #include <memory>
05722 #include <ostream>
05723
05724 namespace Catch {
05725     void prepareExpandedExpression( AssertionResult& result );
05726
05727     // Returns double formatted as %.3f (format expected on output)
05728     std::string getFormattedDuration( double duration );
05729
05730     bool shouldShowDuration( IConfig const& config, double duration );
05731
05732     std::string serializeFilters( std::vector<std::string> const& container );
05733
05734     template<typename DerivedT>
05735     struct StreamingReporterBase : IStreamingReporter {
05736     StreamingReporterBase( ReporterConfig const& _config )
05737         : m_config( _config.fullConfig() ),
05738           stream( _config.stream() ) {
05739         {
05740             m_reporterPrefs.shouldRedirectStdOut = false;
05741             if ( !DerivedT::getSupportedVerbsosities().count( m_config->verbosity() ) )
05742                 CATCH_ERROR( "Verbosity level not supported by this reporter" );
05743         }
05744
05745         ReporterPreferences getPreferences() const override {
05746             return m_reporterPrefs;
05747         }
05748
05749         static std::set<Verbosity> getSupportedVerbsosities() {
05750             return { Verbosity::Normal };
05751         }
05752
05753         ~StreamingReporterBase() override = default;
05754
05755         void noMatchingTestCases( std::string const& ) override {}
05756
05757         void reportInvalidArguments( std::string const& ) override {}
05758
05759         void testRunStarting( TestRunInfo const& _testRunInfo ) override {
05760             currentTestRunInfo = _testRunInfo;
05761         }
05762
05763         void testGroupStarting( GroupInfo const& _groupInfo ) override {
05764             currentGroupInfo = _groupInfo;
05765         }
05766
05767         void testCaseStarting( TestCaseInfo const& _testInfo ) override {
05768             currentTestCaseInfo = _testInfo;
05769         }
05770
05771         void sectionStarting( SectionInfo const& _sectionInfo ) override {
05772             m_sectionStack.push_back( _sectionInfo );
05773         }
05774
05775         void sectionEnded( SectionStats const& /* _sectionStats */ ) override {
05776             m_sectionStack.pop_back();
05777         }
05778
05779         void testCaseEnded( TestCaseStats const& /* _testCaseStats */ ) override {
05780             currentTestCaseInfo.reset();
05781         }
05782
05783         void testGroupEnded( TestGroupStats const& /* _testGroupStats */ ) override {
05784             currentGroupInfo.reset();
05785         }
05786
05787         void testRunEnded( TestRunStats const& /* _testRunStats */ ) override {
05788             currentTestCaseInfo.reset();
05789             currentGroupInfo.reset();
05790             currentTestRunInfo.reset();
05791         }
05792
05793         void skipTest( TestCaseInfo const& ) override {
05794             // Don't do anything with this by default.
05795             // It can optionally be overridden in the derived class.
05796         }
05797
05798         IConfigPtr m_config;
05799         std::ostream& stream;
05800     };

```

```

05799     LazyStat<TestRunInfo> currentTestRunInfo;
05800     LazyStat<GroupInfo> currentGroupInfo;
05801     LazyStat<TestCaseInfo> currentTestCaseInfo;
05802
05803     std::vector<SectionInfo> m_sectionStack;
05804     ReporterPreferences m_reporterPrefs;
05805 };
05806
05807 template<typename DerivedT>
05808 struct CumulativeReporterBase : IStreamingReporter {
05809     template<typename T, typename ChildNodeT>
05810     struct Node {
05811         explicit Node( T const& _value ) : value( _value ) {}
05812         virtual ~Node() {}
05813
05814         using ChildNodes = std::vector<std::shared_ptr<ChildNodeT>;
05815         T value;
05816         ChildNodes children;
05817     };
05818     struct SectionNode {
05819         explicit SectionNode(SectionStats const& _stats) : stats(_stats) {}
05820         virtual ~SectionNode() = default;
05821
05822         bool operator == (SectionNode const& other) const {
05823             return stats.sectionInfo.lineInfo == other.stats.sectionInfo.lineInfo;
05824         }
05825         bool operator == (std::shared_ptr<SectionNode> const& other) const {
05826             return operator==( *other );
05827         }
05828
05829         SectionStats stats;
05830         using ChildSections = std::vector<std::shared_ptr<SectionNode>;
05831         using Assertions = std::vector<AssertionStats>;
05832         ChildSections childSections;
05833         Assertions assertions;
05834         std::string stdOut;
05835         std::string stdErr;
05836     };
05837
05838     struct BySectionInfo {
05839         BySectionInfo( SectionInfo const& other ) : m_other( other ) {}
05840         BySectionInfo( BySectionInfo const& other ) : m_other( other.m_other ) {}
05841         bool operator() (std::shared_ptr<SectionNode> const& node) const {
05842             return ((node->stats.sectionInfo.name == m_other.name) &&
05843                 (node->stats.sectionInfo.lineInfo == m_other.lineInfo));
05844         }
05845         void operator=(BySectionInfo const&) = delete;
05846
05847     private:
05848         SectionInfo const& m_other;
05849     };
05850
05851     using TestCaseNode = Node<TestCaseStats, SectionNode>;
05852     using TestGroupNode = Node<TestGroupStats, TestCaseNode>;
05853     using TestRunNode = Node<TestRunStats, TestGroupNode>;
05854
05855     CumulativeReporterBase( ReporterConfig const& _config )
05856         : m_config( _config.fullConfig() ),
05857           stream( _config.stream() )
05858     {
05859         m_reporterPrefs.shouldRedirectStdOut = false;
05860         if( !DerivedT::getSupportedVerbsosities().count( m_config->verbosity() ) )
05861             CATCH_ERROR( "Verbosity level not supported by this reporter" );
05862     }
05863     ~CumulativeReporterBase() override = default;
05864
05865     ReporterPreferences getPreferences() const override {
05866         return m_reporterPrefs;
05867     }
05868
05869     static std::set<Verbosity> getSupportedVerbsosities() {
05870         return { Verbosity::Normal };
05871     }
05872
05873     void testRunStarting( TestRunInfo const& ) override {}
05874     void testGroupStarting( GroupInfo const& ) override {}
05875
05876     void testCaseStarting( TestCaseInfo const& ) override {}
05877
05878     void sectionStarting( SectionInfo const& sectionInfo ) override {
05879         SectionStats incompleteStats( sectionInfo, Counts(), 0, false );
05880         std::shared_ptr<SectionNode> node;
05881         if( m_sectionStack.empty() ) {
05882             if( !m_rootSection )
05883                 m_rootSection = std::make_shared<SectionNode>( incompleteStats );
05884             node = m_rootSection;
05885         }

```

```

05886         else {
05887             SectionNode& parentNode = *m_sectionStack.back();
05888             auto it =
05889                 std::find_if( parentNode.childSections.begin(),
05890                             parentNode.childSections.end(),
05891                             BySectionInfo( sectionInfo ) );
05892             if( it == parentNode.childSections.end() ) {
05893                 node = std::make_shared<SectionNode>( incompleteStats );
05894                 parentNode.childSections.push_back( node );
05895             }
05896             else
05897                 node = *it;
05898         }
05899         m_sectionStack.push_back( node );
05900         m_deepestSection = std::move(node);
05901     }
05902
05903     void assertionStarting(AssertionInfo const& override) {}
05904
05905     bool assertionEnded(AssertionStats const& assertionStats) override {
05906         assert(!m_sectionStack.empty());
05907         // AssertionResult holds a pointer to a temporary DecomposedExpression,
05908         // which getExpandedExpression() calls to build the expression string.
05909         // Our section stack copy of the assertionResult will likely outlive the
05910         // temporary, so it must be expanded or discarded now to avoid calling
05911         // a destroyed object later.
05912         prepareExpandedExpression(const_cast<AssertionResult&>( assertionStats.assertionResult ));
05913     };
05914     SectionNode& sectionNode = *m_sectionStack.back();
05915     sectionNode.assertions.push_back(assertionStats);
05916     return true;
05917 }
05918 void sectionEnded(SectionStats const& sectionStats) override {
05919     assert(!m_sectionStack.empty());
05920     SectionNode& node = *m_sectionStack.back();
05921     node.stats = sectionStats;
05922     m_sectionStack.pop_back();
05923 }
05924 void testCaseEnded(TestCaseStats const& testCaseStats) override {
05925     auto node = std::make_shared<TestCaseNode>(testCaseStats);
05926     assert(m_sectionStack.size() == 0);
05927     node->children.push_back(m_rootSection);
05928     m_testCases.push_back(node);
05929     m_rootSection.reset();
05930
05931     assert(m_deepestSection);
05932     m_deepestSection->stdOut = testCaseStats.stdOut;
05933     m_deepestSection->stdErr = testCaseStats.stdErr;
05934 }
05935 void testGroupEnded(TestGroupStats const& testGroupStats) override {
05936     auto node = std::make_shared<TestGroupNode>(testGroupStats);
05937     node->children.swap(m_testCases);
05938     m_testGroups.push_back(node);
05939 }
05940 void testRunEnded(TestRunStats const& testRunStats) override {
05941     auto node = std::make_shared<TestRunNode>(testRunStats);
05942     node->children.swap(m_testGroups);
05943     m_testRuns.push_back(node);
05944     testRunEndedCumulative();
05945 }
05946 virtual void testRunEndedCumulative() = 0;
05947
05948 void skipTest(TestCaseInfo const& override) {}
05949
05950 IConfigPtr m_config;
05951 std::ostream& stream;
05952 std::vector<AssertionStats> m_assertions;
05953 std::vector<std::vector<std::shared_ptr<SectionNode>>> m_sections;
05954 std::vector<std::shared_ptr<TestCaseNode>> m_testCases;
05955 std::vector<std::shared_ptr<TestGroupNode>> m_testGroups;
05956
05957 std::vector<std::shared_ptr<TestRunNode>> m_testRuns;
05958
05959 std::shared_ptr<SectionNode> m_rootSection;
05960 std::shared_ptr<SectionNode> m_deepestSection;
05961 std::vector<std::shared_ptr<SectionNode>> m_sectionStack;
05962 ReporterPreferences m_reporterPrefs;
05963 };
05964
05965 template<char C>
05966 char const* getLineOfChars() {
05967     static char line[CATCH_CONFIG_CONSOLE_WIDTH] = {0};
05968     if( !*line ) {
05969         std::memset( line, C, CATCH_CONFIG_CONSOLE_WIDTH-1 );
05970         line[CATCH_CONFIG_CONSOLE_WIDTH-1] = 0;
05971     }
05972     return line;

```

```

05972     }
05973
05974     struct TestEventListenerBase : StreamingReporterBase<TestEventListenerBase> {
05975         TestEventListenerBase( ReporterConfig const& _config );
05976
05977         static std::set<Verbosity> getSupportedVerbsosities();
05978
05979         void assertionStarting( AssertionInfo const& ) override;
05980         bool assertionEnded( AssertionStats const& ) override;
05981     };
05982
05983 } // end namespace Catch
05984
05985 // end catch_reporter_bases.hpp
05986 // start catch_console_colour.h
05987
05988 namespace Catch {
05989
05990     struct Colour {
05991         enum Code {
05992             None = 0,
05993
05994             White,
05995             Red,
05996             Green,
05997             Blue,
05998             Cyan,
05999             Yellow,
06000             Grey,
06001
06002             Bright = 0x10,
06003
06004             BrightRed = Bright | Red,
06005             BrightGreen = Bright | Green,
06006             LightGrey = Bright | Grey,
06007             BrightWhite = Bright | White,
06008             BrightYellow = Bright | Yellow,
06009
06010             // By intention
06011             FileName = LightGrey,
06012             Warning = BrightYellow,
06013             ResultError = BrightRed,
06014             ResultSuccess = BrightGreen,
06015             ResultExpectedFailure = Warning,
06016
06017             Error = BrightRed,
06018             Success = Green,
06019
06020             OriginalExpression = Cyan,
06021             ReconstructedExpression = BrightYellow,
06022
06023             SecondaryText = LightGrey,
06024             Headers = White
06025         };
06026
06027         // Use constructed object for RAII guard
06028         Colour( Code _colourCode );
06029         Colour( Colour&& other ) noexcept;
06030         Colour& operator=( Colour&& other ) noexcept;
06031         ~Colour();
06032
06033         // Use static method for one-shot changes
06034         static void use( Code _colourCode );
06035
06036     private:
06037         bool m_moved = false;
06038     };
06039
06040     std::ostream& operator << ( std::ostream& os, Colour const& );
06041
06042 } // end namespace Catch
06043
06044 // end catch_console_colour.h
06045 // start catch_reporter_registrars.hpp
06046
06047 namespace Catch {
06048
06049     template<typename T>
06050     class ReporterRegistrar {
06051     public:
06052         class ReporterFactory : public IReporterFactory {
06053         public:
06054             IStreamingReporterPtr create( ReporterConfig const& config ) const override {
06055                 return std::unique_ptr<T>( new T( config ) );
06056             }
06057         };
06058     };

```

```

06059         std::string getDescription() const override {
06060             return T::getDescription();
06061         }
06062     };
06063
06064     public:
06065
06066         explicit ReporterRegistrar( std::string const& name ) {
06067             getMutableRegistryHub().registerReporter( name, std::make_shared<ReporterFactory>() );
06068         }
06069     };
06070
06071     template<typename T>
06072     class ListenerRegistrar {
06073
06074         class ListenerFactory : public IReporterFactory {
06075
06076             IStreamingReporterPtr create( ReporterConfig const& config ) const override {
06077                 return std::unique_ptr<T>( new T( config ) );
06078             }
06079             std::string getDescription() const override {
06080                 return std::string();
06081             }
06082         };
06083
06084         public:
06085
06086             ListenerRegistrar() {
06087                 getMutableRegistryHub().registerListener( std::make_shared<ListenerFactory>() );
06088             }
06089     };
06090 }
06091
06092 #if !defined(CATCH_CONFIG_DISABLE)
06093
06094 #define CATCH_REGISTER_REPORTER( name, reporterType ) \
06095     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
06096     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
06097     namespace{ Catch::ReporterRegistrar<reporterType> catch_internal_RegistrarFor##reporterType( name
06098 ); } \
06099     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
06100
06101 #define CATCH_REGISTER_LISTENER( listenerType ) \
06102     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
06103     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
06104     namespace{ Catch::ListenerRegistrar<listenerType> catch_internal_RegistrarFor##listenerType; } \
06105     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
06106 #else // CATCH_CONFIG_DISABLE
06107 #define CATCH_REGISTER_REPORTER(name, reporterType)
06108 #define CATCH_REGISTER_LISTENER(listenerType)
06109 #endif // CATCH_CONFIG_DISABLE
06110
06111 // end catch_reporter_registrars.hpp
06112 // Allow users to base their work off existing reporters
06113 // start catch_reporter_compact.h
06114
06115 namespace Catch {
06116
06117     struct CompactReporter : StreamingReporterBase<CompactReporter> {
06118
06119         using StreamingReporterBase::StreamingReporterBase;
06120
06121         ~CompactReporter() override;
06122
06123         static std::string getDescription();
06124
06125         void noMatchingTestCases( std::string const& spec ) override;
06126
06127         void assertionStarting( AssertionInfo const& ) override;
06128
06129         bool assertionEnded( AssertionStats const& _assertionStats ) override;
06130
06131         void sectionEnded( SectionStats const& _sectionStats ) override;
06132
06133         void testRunEnded( TestRunStats const& _testRunStats ) override;
06134     };
06135
06136 } // end namespace Catch
06137
06138 } // end namespace Catch
06139
06140 // end catch_reporter_compact.h
06141 // start catch_reporter_console.h
06142
06143 #if defined(_MSC_VER)
06144 #pragma warning(push)

```



```

06145 #pragma warning(disable:4061) // Not all labels are EXPLICITLY handled in switch
06146 // Note that 4062 (not all labels are handled
06147 // and default is missing) is enabled
06148 #endif
06149
06150 namespace Catch {
06151     // Fwd decls
06152     struct SummaryColumn;
06153     class TablePrinter;
06154
06155     struct ConsoleReporter : StreamingReporterBase<ConsoleReporter> {
06156         std::unique_ptr<TablePrinter> m_tablePrinter;
06157
06158         ConsoleReporter(ReporterConfig const& config);
06159         ~ConsoleReporter() override;
06160         static std::string getDescription();
06161
06162         void noMatchingTestCases(std::string const& spec) override;
06163
06164         void reportInvalidArguments(std::string const& arg) override;
06165
06166         void assertionStarting(AssertionInfo const&) override;
06167
06168         bool assertionEnded(AssertionStats const& _assertionStats) override;
06169
06170         void sectionStarting(SectionInfo const& _sectionInfo) override;
06171         void sectionEnded(SectionStats const& _sectionStats) override;
06172
06173         #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
06174         void benchmarkPreparing(std::string const& name) override;
06175         void benchmarkStarting(BenchmarkInfo const& info) override;
06176         void benchmarkEnded(BenchmarkStats const& stats) override;
06177         void benchmarkFailed(std::string const& error) override;
06178         #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
06179
06180         void testCaseEnded(TestCaseStats const& _testCaseStats) override;
06181         void testGroupEnded(TestGroupStats const& _testGroupStats) override;
06182         void testRunEnded(TestRunStats const& _testRunStats) override;
06183         void testRunStarting(TestRunInfo const& _testRunInfo) override;
06184     private:
06185         void lazyPrint();
06186
06187         void lazyPrintWithoutClosingBenchmarkTable();
06188         void lazyPrintRunInfo();
06189         void lazyPrintGroupInfo();
06190         void printTestCaseAndSectionHeader();
06191
06192         void printClosedHeader(std::string const& _name);
06193         void printOpenHeader(std::string const& _name);
06194
06195         // if string has a : in first line will set indent to follow it on
06196         // subsequent lines
06197         void printHeaderString(std::string const& _string, std::size_t indent = 0);
06198
06199         void printTotals(Totals const& totals);
06200         void printSummaryRow(std::string const& label, std::vector<SummaryColumn> const& cols,
06201             std::size_t row);
06202
06203         void printTotalsDivider(Totals const& totals);
06204         void printSummaryDivider();
06205         void printTestFilters();
06206
06207     private:
06208         bool m_headerPrinted = false;
06209     };
06210
06211 } // end namespace Catch
06212
06213 #if defined(_MSC_VER)
06214 #pragma warning(pop)
06215 #endif
06216
06217 // end catch_reporter_console.h
06218 // start catch_reporter_junit.h
06219
06220 // start catch_xmlwriter.h
06221
06222 #include <vector>
06223
06224 namespace Catch {
06225     enum class XmlFormatting {
06226         None = 0x00,
06227         Indent = 0x01,
06228         Newline = 0x02,
06229     };
06230

```

```

06231     XmlFormatting operator | (XmlFormatting lhs, XmlFormatting rhs);
06232     XmlFormatting operator & (XmlFormatting lhs, XmlFormatting rhs);
06233
06234     class XmlEncode {
06235     public:
06236         enum ForWhat { ForTextNodes, ForAttributes };
06237
06238         XmlEncode( std::string const& str, ForWhat forWhat = ForTextNodes );
06239
06240         void encodeTo( std::ostream& os ) const;
06241
06242         friend std::ostream& operator < ( std::ostream& os, XmlEncode const& xmlEncode );
06243
06244     private:
06245         std::string m_str;
06246         ForWhat m_forWhat;
06247     };
06248
06249     class XmlWriter {
06250     public:
06251
06252         class ScopedElement {
06253         public:
06254             ScopedElement( XmlWriter* writer, XmlFormatting fmt );
06255
06256             ScopedElement( ScopedElement&& other ) noexcept;
06257             ScopedElement& operator=( ScopedElement&& other ) noexcept;
06258
06259             ~ScopedElement();
06260
06261             ScopedElement& writeText( std::string const& text, XmlFormatting fmt =
06262             XmlFormatting::Newline | XmlFormatting::Indent );
06263
06264             template<typename T>
06265             ScopedElement& writeAttribute( std::string const& name, T const& attribute ) {
06266                 m_writer->writeAttribute( name, attribute );
06267                 return *this;
06268             }
06269
06270         private:
06271             mutable XmlWriter* m_writer = nullptr;
06272             XmlFormatting m_fmt;
06273         };
06274
06275         XmlWriter( std::ostream& os = Catch::cout() );
06276         ~XmlWriter();
06277
06278         XmlWriter( XmlWriter const& ) = delete;
06279         XmlWriter& operator=( XmlWriter const& ) = delete;
06280
06281         XmlWriter& startElement( std::string const& name, XmlFormatting fmt = XmlFormatting::Newline |
06282         XmlFormatting::Indent);
06283
06284         ScopedElement scopedElement( std::string const& name, XmlFormatting fmt =
06285         XmlFormatting::Newline | XmlFormatting::Indent);
06286
06287         XmlWriter& endElement(XmlFormatting fmt = XmlFormatting::Newline | XmlFormatting::Indent);
06288
06289         XmlWriter& writeAttribute( std::string const& name, std::string const& attribute );
06290
06291         XmlWriter& writeAttribute( std::string const& name, bool attribute );
06292
06293         template<typename T>
06294         XmlWriter& writeAttribute( std::string const& name, T const& attribute ) {
06295             ReusableStringStream rss;
06296             rss << attribute;
06297             return writeAttribute( name, rss.str() );
06298         }
06299
06300         XmlWriter& writeText( std::string const& text, XmlFormatting fmt = XmlFormatting::Newline |
06301         XmlFormatting::Indent);
06302
06303         XmlWriter& writeComment(std::string const& text, XmlFormatting fmt = XmlFormatting::Newline |
06304         XmlFormatting::Indent);
06305
06306         void writeStylesheetRef( std::string const& url );
06307
06308         XmlWriter& writeBlankLine();
06309
06310         void ensureTagClosed();
06311
06312     private:
06313
06314         void applyFormatting(XmlFormatting fmt);
06315
06316         void writeDeclaration();

```

```

06313         void newlineIfNecessary();
06314
06315         bool m_tagIsOpen = false;
06316         bool m_needsNewline = false;
06317         std::vector<std::string> m_tags;
06318         std::string m_indent;
06319         std::ostream& m_os;
06320     };
06321
06322 }
06323
06324 // end catch_xmlwriter.h
06325 namespace Catch {
06326
06327     class JunitReporter : public CumulativeReporterBase<JunitReporter> {
06328     public:
06329         JunitReporter(ReporterConfig const& _config);
06330
06331         ~JunitReporter() override;
06332
06333         static std::string getDescription();
06334
06335         void noMatchingTestCases(std::string const& /*spec*/) override;
06336
06337         void testRunStarting(TestRunInfo const& runInfo) override;
06338
06339         void testGroupStarting(GroupInfo const& groupInfo) override;
06340
06341         void testCaseStarting(TestCaseInfo const& testCaseInfo) override;
06342         bool assertionEnded(AssertionStats const& assertionStats) override;
06343
06344         void testCaseEnded(TestCaseStats const& testCaseStats) override;
06345
06346         void testGroupEnded(TestGroupStats const& testGroupStats) override;
06347
06348         void testRunEndedCumulative() override;
06349
06350         void writeGroup(TestGroupNode const& groupNode, double suiteTime);
06351
06352         void writeTestCase(TestCaseNode const& testCaseNode);
06353
06354         void writeSection( std::string const& className,
06355                           std::string const& rootName,
06356                           SectionNode const& sectionNode,
06357                           bool testOkToFail );
06358
06359         void writeAssertions(SectionNode const& sectionNode);
06360         void writeAssertion(AssertionStats const& stats);
06361
06362         XmlWriter xml;
06363         Timer suiteTimer;
06364         std::string stdOutForSuite;
06365         std::string stdErrForSuite;
06366         unsigned int unexpectedExceptions = 0;
06367         bool m_okToFail = false;
06368     };
06369
06370 } // end namespace Catch
06371
06372 // end catch_reporter_junit.h
06373 // start catch_reporter_xml.h
06374
06375 namespace Catch {
06376     class XmlReporter : public StreamingReporterBase<XmlReporter> {
06377     public:
06378         XmlReporter(ReporterConfig const& _config);
06379
06380         ~XmlReporter() override;
06381
06382         static std::string getDescription();
06383
06384         virtual std::string getStylesheetRef() const;
06385
06386         void writeSourceInfo(SourceLineInfo const& sourceInfo);
06387
06388     public: // StreamingReporterBase
06389
06390         void noMatchingTestCases(std::string const& s) override;
06391
06392         void testRunStarting(TestRunInfo const& testInfo) override;
06393
06394         void testGroupStarting(GroupInfo const& groupInfo) override;
06395
06396         void testCaseStarting(TestCaseInfo const& testInfo) override;
06397
06398         void sectionStarting(SectionInfo const& sectionInfo) override;
06399
06400     };
06401 }

```

```

06400         void assertionStarting(AssertionInfo const&) override;
06401
06402         bool assertionEnded(AssertionStats const& assertionStats) override;
06403
06404         void sectionEnded(SectionStats const& sectionStats) override;
06405
06406         void testCaseEnded(TestCaseStats const& testCaseStats) override;
06407
06408         void testGroupEnded(TestGroupStats const& testGroupStats) override;
06409
06410         void testRunEnded(TestRunStats const& testRunStats) override;
06411
06412 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
06413     void benchmarkPreparing(std::string const& name) override;
06414     void benchmarkStarting(BenchmarkInfo const&) override;
06415     void benchmarkEnded(BenchmarkStats<> const&) override;
06416     void benchmarkFailed(std::string const&) override;
06417 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
06418
06419     private:
06420         Timer m_testCaseTimer;
06421         XmlWriter m_xml;
06422         int m_sectionDepth = 0;
06423     };
06424
06425 } // end namespace Catch
06426
06427 // end catch_reporter_xml.h
06428
06429 // end catch_external_interfaces.h
06430 #endif
06431
06432 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
06433 // start catch_benchmarking_all.hpp
06434
06435 // A proxy header that includes all of the benchmarking headers to allow
06436 // concise include of the benchmarking features. You should prefer the
06437 // individual includes in standard use.
06438
06439 // start catch_benchmark.hpp
06440
06441 // Benchmark
06442
06443 // start catch_chronometer.hpp
06444
06445 // User-facing chronometer
06446
06447
06448 // start catch_clock.hpp
06449
06450 // Clocks
06451
06452 #include <chrono>
06453 #include <ratio>
06454
06455 namespace Catch {
06456     namespace Benchmark {
06457         template <typename Clock>
06458         using ClockDuration = typename Clock::duration;
06459         template <typename Clock>
06460         using FloatDuration = std::chrono::duration<double, typename Clock::period>;
06461
06462         template <typename Clock>
06463         using TimePoint = typename Clock::time_point;
06464
06465         using default_clock = std::chrono::steady_clock;
06466
06467         template <typename Clock>
06468         struct now {
06469             TimePoint<Clock> operator()() const {
06470                 return Clock::now();
06471             }
06472         };
06473     };
06474
06475     using fp_seconds = std::chrono::duration<double, std::ratio<1>;
06476 } // namespace Benchmark
06477 } // namespace Catch
06478
06479 // end catch_clock.hpp
06480 // start catch_optimizer.hpp
06481
06482 // Hinting the optimizer
06483
06484
06485 #if defined(_MSC_VER)
06486 #    include <atomic> // atomic_thread_fence

```

```

06487 #endif
06488
06489 namespace Catch {
06490     namespace Benchmark {
06491         #if defined(__GNUC__) || defined(__clang__)
06492             template <typename T>
06493             inline void keep_memory(T* p) {
06494                 asm volatile("" : : "g"(p) : "memory");
06495             }
06496             inline void keep_memory() {
06497                 asm volatile("" : : : "memory");
06498             }
06499
06500             namespace Detail {
06501                 inline void optimizer_barrier() { keep_memory(); }
06502             } // namespace Detail
06503         #elif defined(_MSC_VER)
06504
06505         #pragma optimize("", off)
06506         template <typename T>
06507         inline void keep_memory(T* p) {
06508             // thanks @milleniumbug
06509             *reinterpret_cast<char volatile*>(p) = *reinterpret_cast<char const volatile*>(p);
06510         }
06511         // TODO equivalent keep_memory()
06512         #pragma optimize("", on)
06513
06514         namespace Detail {
06515             inline void optimizer_barrier() {
06516                 std::atomic_thread_fence(std::memory_order_seq_cst);
06517             }
06518         } // namespace Detail
06519
06520     #endif
06521
06522     template <typename T>
06523     inline void deoptimize_value(T&& x) {
06524         keep_memory(&x);
06525     }
06526
06527     template <typename Fn, typename... Args>
06528     inline auto invoke_deoptimized(Fn&& fn, Args&&... args) -> typename
std::enable_if<!std::is_same<void, decltype(fn(args...))>::value>::type {
06529         deoptimize_value(std::forward<Fn>(fn) (std::forward<Args...>(args...)));
06530     }
06531
06532     template <typename Fn, typename... Args>
06533     inline auto invoke_deoptimized(Fn&& fn, Args&&... args) -> typename
std::enable_if<std::is_same<void, decltype(fn(args...))>::value>::type {
06534         std::forward<Fn>(fn) (std::forward<Args...>(args...));
06535     }
06536 } // namespace Benchmark
06537 } // namespace Catch
06538
06539 // end catch_optimizer.hpp
06540 // start catch_complete_invoke.hpp
06541
06542 // Invoke with a special case for void
06543
06544 #include <type_traits>
06545 #include <utility>
06546
06547 namespace Catch {
06548     namespace Benchmark {
06549         namespace Detail {
06550             template <typename T>
06551             struct CompleteType { using type = T; };
06552             template <>
06553             struct CompleteType<void> { struct type {}; };
06554
06555             template <typename T>
06556             using CompleteType_t = typename CompleteType<T>::type;
06557
06558             template <typename Result>
06559             struct CompleteInvoker {
06560                 template <typename Fun, typename... Args>
06561                 static Result invoke(Fun&& fun, Args&&... args) {
06562                     return std::forward<Fun>(fun) (std::forward<Args>(args)...);
06563                 }
06564             };
06565
06566             template <>
06567             struct CompleteInvoker<void> {
06568                 template <typename Fun, typename... Args>
06569                 static CompleteType_t<void> invoke(Fun&& fun, Args&&... args) {
06570                     std::forward<Fun>(fun) (std::forward<Args>(args)...);
06571                     return {};
06572                 }
06573             };
06574         }
06575     }
06576 }

```

```

06572         }
06573     };
06574
06575     // invoke and not return void :(
06576     template <typename Fun, typename... Args>
06577     CompleteType_t<FunctionReturnType<Fun, Args...> complete_invoke(Fun&& fun, Args&&... args)
06578     {
06579         return CompleteInvoker<FunctionReturnType<Fun,
06580     Args...>::invoke(std::forward<Fun>(fun), std::forward<Args>(args)...);
06581     }
06582
06583     const std::string benchmarkErrorMsg = "a benchmark failed to run successfully";
06584     } // namespace Detail
06585
06586     template <typename Fun>
06587     Detail::CompleteType_t<FunctionReturnType<Fun> user_code(Fun&& fun) {
06588     CATCH_TRY{
06589         return Detail::complete_invoke(std::forward<Fun>(fun));
06590     } CATCH_CATCH_ALL{
06591         getResultCapture().benchmarkFailed(translateActiveException());
06592         CATCH_RUNTIME_ERROR(Detail::benchmarkErrorMsg);
06593     }
06594     } // namespace Benchmark
06595 } // namespace Catch
06596 // end catch_complete_invoke.hpp
06597 namespace Catch {
06598     namespace Benchmark {
06599         namespace Detail {
06600             struct ChronometerConcept {
06601                 virtual void start() = 0;
06602                 virtual void finish() = 0;
06603                 virtual ~ChronometerConcept() = default;
06604             };
06605             template <typename Clock>
06606             struct ChronometerModel final : public ChronometerConcept {
06607                 void start() override { started = Clock::now(); }
06608                 void finish() override { finished = Clock::now(); }
06609
06610                 ClockDuration<Clock> elapsed() const { return finished - started; }
06611
06612                 TimePoint<Clock> started;
06613                 TimePoint<Clock> finished;
06614             };
06615         } // namespace Detail
06616
06617         struct Chronometer {
06618         public:
06619             template <typename Fun>
06620             void measure(Fun&& fun) { measure(std::forward<Fun>(fun), is_callable<Fun(int)>()); }
06621
06622             int runs() const { return k; }
06623
06624             Chronometer(Detail::ChronometerConcept& meter, int k)
06625                 : impl(&meter)
06626                 , k(k) {}
06627
06628         private:
06629             template <typename Fun>
06630             void measure(Fun&& fun, std::false_type) {
06631                 measure([&fun](int) { return fun(); }, std::true_type());
06632             }
06633
06634             template <typename Fun>
06635             void measure(Fun&& fun, std::true_type) {
06636                 Detail::optimizer_barrier();
06637                 impl->start();
06638                 for (int i = 0; i < k; ++i) invoke_deoptimized(fun, i);
06639                 impl->finish();
06640                 Detail::optimizer_barrier();
06641             }
06642
06643             Detail::ChronometerConcept* impl;
06644             int k;
06645         };
06646     } // namespace Benchmark
06647 } // namespace Catch
06648
06649 // end catch_chronometer.hpp
06650 // start catch_environment.hpp
06651
06652 // Environment information
06653
06654 namespace Catch {
06655     namespace Benchmark {

```

```

06657     template <typename Duration>
06658     struct EnvironmentEstimate {
06659         Duration mean;
06660         OutlierClassification outliers;
06661
06662         template <typename Duration2>
06663         operator EnvironmentEstimate<Duration2>() const {
06664             return { mean, outliers };
06665         }
06666     };
06667     template <typename Clock>
06668     struct Environment {
06669         using clock_type = Clock;
06670         EnvironmentEstimate<FloatDuration<Clock> clock_resolution;
06671         EnvironmentEstimate<FloatDuration<Clock> clock_cost;
06672     };
06673 } // namespace Benchmark
06674 } // namespace Catch
06675
06676 // end catch_environment.hpp
06677 // start catch_execution_plan.hpp
06678
06679 // Execution plan
06680
06681
06682 // start catch_benchmark_function.hpp
06683
06684 // Dumb std::function implementation for consistent call overhead
06685
06686
06687 #include <cassert>
06688 #include <type_traits>
06689 #include <utility>
06690 #include <memory>
06691
06692 namespace Catch {
06693     namespace Benchmark {
06694         namespace Detail {
06695             template <typename T>
06696             using Decay = typename std::decay<T>::type;
06697             template <typename T, typename U>
06698             struct is_related
06699                 : std::is_same<Decay<T>, Decay<U>> {};
06700
06701             struct BenchmarkFunction {
06702 private:
06703                 struct callable {
06704                     virtual void call(Chronometer meter) const = 0;
06705                     virtual callable* clone() const = 0;
06706                     virtual ~callable() = default;
06707                 };
06708                 template <typename Fun>
06709                 struct model : public callable {
06710                     model(Fun&& fun) : fun(std::move(fun)) {}
06711                     model(Fun const& fun) : fun(fun) {}
06712
06713                     model<Fun>* clone() const override { return new model<Fun>(*this); }
06714
06715                     void call(Chronometer meter) const override {
06716                         call(meter, is_callable<Fun(Chronometer)>());
06717                     }
06718                     void call(Chronometer meter, std::true_type) const {
06719                         fun(meter);
06720                     }
06721                     void call(Chronometer meter, std::false_type) const {
06722                         meter.measure(fun);
06723                     }
06724
06725                     Fun fun;
06726                 };
06727
06728                 struct do_nothing { void operator()() const {} };
06729
06730                 template <typename T>
06731                 BenchmarkFunction(model<T>* c) : f(c) {}
06732
06733 public:
06734                 BenchmarkFunction()
06735                     : f(new model<do_nothing>{ {} }) {}
06736
06737                 template <typename Fun,
06738                     typename std::enable_if<!is_related<Fun, BenchmarkFunction>::value, int>::type =
06739                     0>
06740                 BenchmarkFunction(Fun&& fun)
06741                     : f(new model<typename std::decay<Fun>::type>(std::forward<Fun>(fun))) {}
06742
06743                 BenchmarkFunction(BenchmarkFunction&& that)

```

```

06750         : f(std::move(that.f)) {}
06751
06752     BenchmarkFunction(BenchmarkFunction const& that)
06753         : f(that.f->clone()) {}
06754
06755     BenchmarkFunction& operator=(BenchmarkFunction&& that) {
06756         f = std::move(that.f);
06757         return *this;
06758     }
06759
06760     BenchmarkFunction& operator=(BenchmarkFunction const& that) {
06761         f.reset(that.f->clone());
06762         return *this;
06763     }
06764
06765     void operator()(Chronometer meter) const { f->call(meter); }
06766
06767     private:
06768         std::unique_ptr<callable> f;
06769     };
06770 } // namespace Detail
06771 } // namespace Benchmark
06772 } // namespace Catch
06773
06774 // end catch_benchmark_function.hpp
06775 // start catch_repeat.hpp
06776
06777 // repeat algorithm
06778
06779 #include <type_traits>
06780 #include <utility>
06781
06782 namespace Catch {
06783     namespace Benchmark {
06784         namespace Detail {
06785             template <typename Fun>
06786             struct repeater {
06787                 void operator()(int k) const {
06788                     for (int i = 0; i < k; ++i) {
06789                         fun();
06790                     }
06791                 }
06792                 Fun fun;
06793             };
06794             template <typename Fun>
06795             repeater<typename std::decay<Fun>::type> repeat(Fun&& fun) {
06796                 return { std::forward<Fun>(fun) };
06797             }
06798         } // namespace Detail
06799     } // namespace Benchmark
06800 } // namespace Catch
06801
06802 // end catch_repeat.hpp
06803 // start catch_run_for_at_least.hpp
06804
06805 // Run a function for a minimum amount of time
06806
06807 // start catch_measure.hpp
06808
06809 // Measure
06810
06811 // start catch_timing.hpp
06812
06813 // Timing
06814
06815 #include <tuple>
06816 #include <type_traits>
06817
06818 namespace Catch {
06819     namespace Benchmark {
06820         template <typename Duration, typename Result>
06821         struct Timing {
06822             Duration elapsed;
06823             Result result;
06824             int iterations;
06825         };
06826         template <typename Clock, typename Func, typename... Args>
06827         using TimingOf = Timing<ClockDuration<Clock>, Detail::CompleteType_t<FunctionReturnType<Func,
06828             Args...>>>;
06829     } // namespace Benchmark
06830 } // namespace Catch
06831
06832 // end catch_timing.hpp

```



```

06836 #include <utility>
06837
06838 namespace Catch {
06839     namespace Benchmark {
06840         namespace Detail {
06841             template <typename Clock, typename Fun, typename... Args>
06842             TimingOf<Clock, Fun, Args...> measure(Fun&& fun, Args&&... args) {
06843                 auto start = Clock::now();
06844                 auto&& r = Detail::complete_invoke(fun, std::forward<Args>(args)...);
06845                 auto end = Clock::now();
06846                 auto delta = end - start;
06847                 return { delta, std::forward<decltype(r)>(r), 1 };
06848             }
06849         } // namespace Detail
06850     } // namespace Benchmark
06851 } // namespace Catch
06852
06853 // end catch_measure.hpp
06854 #include <utility>
06855 #include <type_traits>
06856
06857 namespace Catch {
06858     namespace Benchmark {
06859         namespace Detail {
06860             template <typename Clock, typename Fun>
06861             TimingOf<Clock, Fun, int> measure_one(Fun&& fun, int iters, std::false_type) {
06862                 return Detail::measure<Clock>(fun, iters);
06863             }
06864             template <typename Clock, typename Fun>
06865             TimingOf<Clock, Fun, Chronometer> measure_one(Fun&& fun, int iters, std::true_type) {
06866                 Detail::ChronometerModel<Clock> meter;
06867                 auto&& result = Detail::complete_invoke(fun, Chronometer(meter, iters));
06868                 return { meter.elapsed(), std::move(result), iters };
06869             }
06870         }
06871
06872         template <typename Clock, typename Fun>
06873         using run_for_at_least_argument_t = typename
06874             std::conditional<is_callable<Fun(Chronometer)>::value, Chronometer, int>::type;
06875
06876         struct optimized_away_error : std::exception {
06877             const char* what() const noexcept override {
06878                 return "could not measure benchmark, maybe it was optimized away";
06879             }
06880         };
06881
06882         template <typename Clock, typename Fun>
06883         TimingOf<Clock, Fun, run_for_at_least_argument_t<Clock, Fun>
06884         run_for_at_least(ClockDuration<Clock> how_long, int seed, Fun&& fun) {
06885             auto iters = seed;
06886             while (iters < (1 << 30)) {
06887                 auto&& Timing = measure_one<Clock>(fun, iters, is_callable<Fun(Chronometer)>());
06888                 if (Timing.elapsed >= how_long) {
06889                     return { Timing.elapsed, std::move(Timing.result), iters };
06890                 }
06891                 iters *= 2;
06892             }
06893             Catch::throw_exception(optimized_away_error{});
06894         } // namespace Detail
06895     } // namespace Benchmark
06896 } // namespace Catch
06897
06898 // end catch_run_for_at_least.hpp
06899 #include <algorithm>
06900 #include <iterator>
06901
06902 namespace Catch {
06903     namespace Benchmark {
06904         template <typename Duration>
06905         struct ExecutionPlan {
06906             int iterations_per_sample;
06907             Duration estimated_duration;
06908             Detail::BenchmarkFunction benchmark;
06909             Duration warmup_time;
06910             int warmup_iterations;
06911
06912             template <typename Duration2>
06913             operator ExecutionPlan<Duration2>() const {
06914                 return { iterations_per_sample, estimated_duration, benchmark, warmup_time,
06915                     warmup_iterations };
06916             }
06917
06918             template <typename Clock>
06919             std::vector<FloatDuration<Clock>> run(const IConfig &cfg, Environment<FloatDuration<Clock>>
06920             env) const {

```

```

06919         // warmup a bit
06920
06921         Detail::run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(warmup_time),
06922         warmup_iterations, Detail::repeat(now<Clock>{}));
06923
06924         std::vector<FloatDuration<Clock>> times;
06925         times.reserve(cfg.benchmarkSamples());
06926         std::generate_n(std::back_inserter(times), cfg.benchmarkSamples(), [this, env] {
06927             Detail::ChronometerModel<Clock> model;
06928             this->benchmark(Chronometer(model, iterations_per_sample));
06929             auto sample_time = model.elapsed() - env.clock_cost.mean();
06930             if (sample_time < FloatDuration<Clock>::zero()) sample_time =
06931             FloatDuration<Clock>::zero();
06932             return sample_time / iterations_per_sample;
06933         });
06934         return times;
06935     }
06936 };
06937 } // namespace Benchmark
06938 } // namespace Catch
06939
06940 // end catch_execution_plan.hpp
06941 // start catch_estimate_clock.hpp
06942
06943 // Environment measurement
06944
06945 // start catch_stats.hpp
06946
06947 // Statistical analysis tools
06948
06949 #include <algorithm>
06950 #include <functional>
06951 #include <vector>
06952 #include <iterator>
06953 #include <numeric>
06954 #include <tuple>
06955 #include <cmath>
06956 #include <utility>
06957 #include <cstdint>
06958 #include <random>
06959
06960 namespace Catch {
06961     namespace Benchmark {
06962         namespace Detail {
06963             using sample = std::vector<double>;
06964
06965             double weighted_average_quantile(int k, int q, std::vector<double>::iterator first,
06966             std::vector<double>::iterator last);
06967
06968             template <typename Iterator>
06969             OutlierClassification classify_outliers(Iterator first, Iterator last) {
06970                 std::vector<double> copy(first, last);
06971
06972                 auto q1 = weighted_average_quantile(1, 4, copy.begin(), copy.end());
06973                 auto q3 = weighted_average_quantile(3, 4, copy.begin(), copy.end());
06974                 auto iqr = q3 - q1;
06975                 auto los = q1 - (iqr * 3.);
06976                 auto lom = q1 - (iqr * 1.5);
06977                 auto him = q3 + (iqr * 1.5);
06978                 auto his = q3 + (iqr * 3.);
06979
06980                 OutlierClassification o;
06981                 for (; first != last; ++first) {
06982                     auto&& t = *first;
06983                     if (t < los) ++o.low_severe;
06984                     else if (t < lom) ++o.low_mild;
06985                     else if (t > his) ++o.high_severe;
06986                     else if (t > him) ++o.high_mild;
06987                     ++o.samples_seen;
06988                 }
06989                 return o;
06990             }
06991
06992             template <typename Iterator>
06993             double mean(Iterator first, Iterator last) {
06994                 double count = last - first;
06995                 double sum = std::accumulate(first, last, 0.);
06996                 return sum / count;
06997             }
06998
06999             template <typename URng, typename Iterator, typename Estimator>
07000             sample resample(URng& rng, int resamples, Iterator first, Iterator last, Estimator&
07001             estimator) {
07002                 auto n = last - first;
07003                 std::uniform_int_distribution<decltype(n)> dist(0, n - 1);

```

```

07001
07002         sample out;
07003         out.reserve(resamples);
07004         std::generate_n(std::back_inserter(out), resamples, [n, first, &estimator, &dist,
&rng] {
07005             std::vector<double> resampled;
07006             resampled.reserve(n);
07007             std::generate_n(std::back_inserter(resampled), n, [first, &dist, &rng] { return
first[dist(rng)]; });
07008             return estimator(resampled.begin(), resampled.end());
07009         });
07010         std::sort(out.begin(), out.end());
07011         return out;
07012     }
07013
07014     template <typename Estimator, typename Iterator>
07015     sample jackknife(Estimator&& estimator, Iterator first, Iterator last) {
07016         auto n = last - first;
07017         auto second = std::next(first);
07018         sample results;
07019         results.reserve(n);
07020
07021         for (auto it = first; it != last; ++it) {
07022             std::iter_swap(it, first);
07023             results.push_back(estimator(second, last));
07024         }
07025
07026         return results;
07027     }
07028
07029     inline double normal_cdf(double x) {
07030         return std::erfc(-x / std::sqrt(2.0)) / 2.0;
07031     }
07032
07033     double erfc_inv(double x);
07034
07035     double normal_quantile(double p);
07036
07037     template <typename Iterator, typename Estimator>
07038     Estimate<double> bootstrap(double confidence_level, Iterator first, Iterator last, sample
const& resample, Estimator&& estimator) {
07039         auto n_samples = last - first;
07040
07041         double point = estimator(first, last);
07042         // Degenerate case with a single sample
07043         if (n_samples == 1) return { point, point, point, confidence_level };
07044
07045         sample jack = jackknife(estimator, first, last);
07046         double jack_mean = mean(jack.begin(), jack.end());
07047         double sum_squares, sum_cubes;
07048         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> {
07049             auto d = jack_mean - x;
07050             auto d2 = d * d;
07051             auto d3 = d2 * d;
07052             return { sqcb.first + d2, sqcb.second + d3 };
07053         });
07054
07055         double accel = sum_cubes / (6 * std::pow(sum_squares, 1.5));
07056         int n = static_cast<int>(resample.size());
07057         double prob_n = std::count_if(resample.begin(), resample.end(), [point](double x) {
return x < point; }) / (double)n;
07058         // degenerate case with uniform samples
07059         if (prob_n == 0) return { point, point, point, confidence_level };
07060
07061         double bias = normal_quantile(prob_n);
07062         double z1 = normal_quantile((1. - confidence_level) / 2.);
07063
07064         auto cumn = [n](double x) -> int {
07065             return std::lround(normal_cdf(x) * n); };
07066         auto a = [bias, accel](double b) { return bias + b / (1. - accel * b); };
07067         double b1 = bias + z1;
07068         double b2 = bias - z1;
07069         double a1 = a(b1);
07070         double a2 = a(b2);
07071         auto lo = (std::max)(cumn(a1), 0);
07072         auto hi = (std::min)(cumn(a2), n - 1);
07073
07074         return { point, resample[lo], resample[hi], confidence_level };
07075     }
07076
07077     double outlier_variance(Estimate<double> mean, Estimate<double> stddev, int n);
07078
07079     struct bootstrap_analysis {
07080         Estimate<double> mean;
07081         Estimate<double> standard_deviation;

```

```

07082         double outlier_variance;
07083     };
07084
07085     bootstrap_analysis analyse_samples(double confidence_level, int n_resamples,
std::vector<double>::iterator first, std::vector<double>::iterator last);
07086     } // namespace Detail
07087     } // namespace Benchmark
07088 } // namespace Catch
07089
07090 // end catch_stats.hpp
07091 #include <algorithm>
07092 #include <iterator>
07093 #include <tuple>
07094 #include <vector>
07095 #include <cmath>
07096
07097 namespace Catch {
07098     namespace Benchmark {
07099         namespace Detail {
07100             template <typename Clock>
07101             std::vector<double> resolution(int k) {
07102                 std::vector<TimePoint<Clock>> times;
07103                 times.reserve(k + 1);
07104                 std::generate_n(std::back_inserter(times), k + 1, now<Clock>{});
07105
07106                 std::vector<double> deltas;
07107                 deltas.reserve(k);
07108                 std::transform(std::next(times.begin()), times.end(), times.begin(),
07109                     std::back_inserter(deltas),
07110                     [](TimePoint<Clock> a, TimePoint<Clock> b) { return static_cast<double>((a -
07111 b).count()); });
07112
07113                 return deltas;
07114             }
07115
07116             const auto warmup_iterations = 10000;
07117             const auto warmup_time = std::chrono::milliseconds(100);
07118             const auto minimum_ticks = 1000;
07119             const auto warmup_seed = 10000;
07120             const auto clock_resolution_estimation_time = std::chrono::milliseconds(500);
07121             const auto clock_cost_estimation_time_limit = std::chrono::seconds(1);
07122             const auto clock_cost_estimation_tick_limit = 100000;
07123             const auto clock_cost_estimation_time = std::chrono::milliseconds(10);
07124             const auto clock_cost_estimation_iterations = 10000;
07125
07126             template <typename Clock>
07127             int warmup() {
07128                 return
run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(warmup_time), warmup_seed,
&resolution<Clock>)
07129                     .iterations;
07130             }
07131             template <typename Clock>
07132             EnvironmentEstimate<FloatDuration<Clock>> estimate_clock_resolution(int iterations) {
07133                 auto r =
run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(clock_resolution_estimation_time),
iterations, &resolution<Clock>)
07134                     .result;
07135                 return {
07136                     FloatDuration<Clock>(mean(r.begin(), r.end())),
07137                     classify_outliers(r.begin(), r.end()),
07138                 };
07139             }
07140             template <typename Clock>
07141             EnvironmentEstimate<FloatDuration<Clock>> estimate_clock_cost(FloatDuration<Clock>
resolution) {
07142                 auto time_limit = (std::min)(
07143                     resolution * clock_cost_estimation_tick_limit,
07144                     FloatDuration<Clock>(clock_cost_estimation_time_limit));
07145                 auto time_clock = [](int k) {
07146                     return Detail::measure<Clock>([k] {
07147                         for (int i = 0; i < k; ++i) {
07148                             volatile auto ignored = Clock::now();
07149                             (void)ignored;
07150                         }
07151                     }).elapsed;
07152                 };
07153                 time_clock(1);
07154                 int iters = clock_cost_estimation_iterations;
07155                 auto&& r =
run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(clock_cost_estimation_time),
iters, time_clock);
07156                 std::vector<double> times;
07157                 int nsamples = static_cast<int>(std::ceil(time_limit / r.elapsed));
07158                 times.reserve(nsamples);
07159                 std::generate_n(std::back_inserter(times), nsamples, [time_clock, &r] {
07160                     return static_cast<double>((time_clock(r.iterations) / r.iterations).count());
07161                 });
07162             }
07163         }
07164     }
07165 }

```

```

07160         });
07161         return {
07162             FloatDuration<Clock>(mean(times.begin(), times.end())),
07163             classify_outliers(times.begin(), times.end()),
07164         };
07165     }
07166
07167     template <typename Clock>
07168     Environment<FloatDuration<Clock>> measure_environment() {
07169         static Environment<FloatDuration<Clock>>* env = nullptr;
07170         if (env) {
07171             return *env;
07172         }
07173
07174         auto iters = Detail::warmup<Clock>();
07175         auto resolution = Detail::estimate_clock_resolution<Clock>(iters);
07176         auto cost = Detail::estimate_clock_cost<Clock>(resolution.mean);
07177
07178         env = new Environment<FloatDuration<Clock>>(resolution, cost);
07179         return *env;
07180     }
07181 } // namespace Detail
07182 } // namespace Benchmark
07183 } // namespace Catch
07184
07185 // end catch_estimate_clock.hpp
07186 // start catch_analyse.hpp
07187
07188 // Run and analyse one benchmark
07189
07190
07191 // start catch_sample_analysis.hpp
07192
07193 // Benchmark results
07194
07195
07196 #include <algorithm>
07197 #include <vector>
07198 #include <string>
07199 #include <iterator>
07200
07201 namespace Catch {
07202     namespace Benchmark {
07203         template <typename Duration>
07204         struct SampleAnalysis {
07205             std::vector<Duration> samples;
07206             Estimate<Duration> mean;
07207             Estimate<Duration> standard_deviation;
07208             OutlierClassification outliers;
07209             double outlier_variance;
07210
07211             template <typename Duration2>
07212             operator SampleAnalysis<Duration2>() const {
07213                 std::vector<Duration2> samples2;
07214                 samples2.reserve(samples.size());
07215                 std::transform(samples.begin(), samples.end(), std::back_inserter(samples2),
07216                     [](Duration d) { return Duration2(d); });
07217                 return {
07218                     std::move(samples2),
07219                     mean,
07220                     standard_deviation,
07221                     outliers,
07222                     outlier_variance,
07223                 };
07224             }
07225         }; // namespace Benchmark
07226     } // namespace Catch
07227
07228 // end catch_sample_analysis.hpp
07229 #include <algorithm>
07230 #include <iterator>
07231 #include <vector>
07232
07233 namespace Catch {
07234     namespace Benchmark {
07235         namespace Detail {
07236             template <typename Duration, typename Iterator>
07237             SampleAnalysis<Duration> analyse(const IConfig &cfg, Environment<Duration>, Iterator
07238                 first, Iterator last) {
07239                 if (!cfg.benchmarkNoAnalysis()) {
07240                     std::vector<double> samples;
07241                     samples.reserve(last - first);
07242                     std::transform(first, last, std::back_inserter(samples), [](Duration d) { return
07243                         d.count(); });
07244                     auto analysis =

```

```

Catch::Benchmark::Detail::analyse_samples(cfg.benchmarkConfidenceInterval(), cfg.benchmarkResamples(),
samples.begin(), samples.end());
07244     auto outliers = Catch::Benchmark::Detail::classify_outliers(samples.begin(),
samples.end());
07245
07246     auto wrap_estimate = [](Estimate<double> e) {
07247         return Estimate<Duration> {
07248             Duration(e.point),
07249             Duration(e.lower_bound),
07250             Duration(e.upper_bound),
07251             e.confidence_interval,
07252         };
07253     };
07254     std::vector<Duration> samples2;
07255     samples2.reserve(samples.size());
07256     std::transform(samples.begin(), samples.end(), std::back_inserter(samples2),
[] (double d) { return Duration(d); });
07257     return {
07258         std::move(samples2),
07259         wrap_estimate(analysis.mean),
07260         wrap_estimate(analysis.standard_deviation),
07261         outliers,
07262         analysis.outlier_variance,
07263     };
07264 } else {
07265     std::vector<Duration> samples;
07266     samples.reserve(last - first);
07267
07268     Duration mean = Duration(0);
07269     int i = 0;
07270     for (auto it = first; it < last; ++it, ++i) {
07271         samples.push_back(Duration(*it));
07272         mean += Duration(*it);
07273     }
07274     mean /= i;
07275
07276     return {
07277         std::move(samples),
07278         Estimate<Duration>{mean, mean, mean, 0.0},
07279         Estimate<Duration>{Duration(0), Duration(0), Duration(0), 0.0},
07280         OutlierClassification{},
07281         0.0
07282     };
07283 }
07284 }
07285 } // namespace Detail
07286 } // namespace Benchmark
07287 } // namespace Catch
07288
07289 // end catch_analyse.hpp
07290 #include <algorithm>
07291 #include <functional>
07292 #include <string>
07293 #include <vector>
07294 #include <cmath>
07295
07296 namespace Catch {
07297     namespace Benchmark {
07298         struct Benchmark {
07299             Benchmark(std::string &&name)
07300                 : name(std::move(name)) {}
07301
07302             template <class FUN>
07303             Benchmark(std::string &&name, FUN &&func)
07304                 : fun(std::move(func)), name(std::move(name)) {}
07305
07306             template <typename Clock>
07307             ExecutionPlan<FloatDuration<Clock>> prepare(const IConfig &cfg,
Environment<FloatDuration<Clock>> env) const {
07308                 auto min_time = env.clock_resolution.mean * Detail::minimum_ticks;
07309                 auto run_time = std::max(min_time,
std::chrono::duration_cast<decltype(min_time)>(cfg.benchmarkWarmupTime()));
07310                 auto&& test =
Detail::run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(run_time), 1, fun);
07311                 int new_iters = static_cast<int>(std::ceil(min_time * test.iterations /
test.elapsed));
07312                 return { new_iters, test.elapsed / test.iterations * new_iters *
cfg.benchmarkSamples(), fun,
std::chrono::duration_cast<FloatDuration<Clock>>(cfg.benchmarkWarmupTime()), Detail::warmup_iterations
};
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             BenchmarkInfo info {
07328                 name,
07329                 plan.estimated_duration.count(),
07330                 plan.iterations_per_sample,
07331                 cfg->benchmarkSamples(),
07332                 cfg->benchmarkResamples(),
07333                 env.clock_resolution.mean.count(),
07334                 env.clock_cost.mean.count()
07335             };
07336
07337             getResultCapture().benchmarkStarting(info);
07338
07339             auto samples = user_code([&] {
07340                 return plan.template run<Clock>(*cfg, env);
07341             });
07342
07343             auto analysis = Detail::analyse(*cfg, env, samples.begin(), samples.end());
07344             BenchmarkStats<FloatDuration<Clock>> stats{ info, analysis.samples, analysis.mean,
analysis.standard_deviation, analysis.outliers, analysis.outlier_variance };
07345             getResultCapture().benchmarkEnded(stats);
07346
07347             } CATCH_CATCH_ALL{
07348                 if (translateActiveException() != Detail::benchmarkErrorMsg) // benchmark errors
have been reported, otherwise rethrow.
07349                 std::rethrow_exception(std::current_exception());
07350             }
07351         }
07352
07353         // sets lambda to be used in fun *and* executes benchmark!
07354         template <typename Fun,
07355             typename std::enable_if<!Detail::is_related<Fun, Benchmark>::value, int>::type = 0>
07356             Benchmark & operator=(Fun func) {
07357             fun = Detail::BenchmarkFunction(func);
07358             run();
07359             return *this;
07360         }
07361
07362         explicit operator bool() {
07363             return true;
07364         }
07365
07366     private:
07367         Detail::BenchmarkFunction fun;
07368         std::string name;
07369     };
07370 }
07371 } // namespace Catch
07372
07373 #define INTERNAL_CATCH_GET_1_ARG(arg1, arg2, ...) arg1
07374 #define INTERNAL_CATCH_GET_2_ARG(arg1, arg2, ...) arg2
07375
07376 #define INTERNAL_CATCH_BENCHMARK(BenchmarkName, name, benchmarkIndex)\
07377     if( Catch::Benchmark::Benchmark(BenchmarkName{name}) ) \
07378         BenchmarkName = [&](int benchmarkIndex)
07379
07380 #define INTERNAL_CATCH_BENCHMARK_ADVANCED(BenchmarkName, name)\
07381     if( Catch::Benchmark::Benchmark(BenchmarkName{name}) ) \
07382         BenchmarkName = [&]
07383
07384 // end catch_benchmark.hpp
07385 // start catch_constructor.hpp
07386
07387 // Constructor and destructor helpers
07388
07389 #include <type_traits>
07390
07391 namespace Catch {
07392     namespace Benchmark {
07393         namespace Detail {
07394             template <typename T, bool Destruct>
07395             struct ObjectStorage
07396             {
07397                 ObjectStorage() : data() {}
07398
07399                 ObjectStorage(const ObjectStorage& other)
07400                 {
07401                     new(&data) T(other.stored_object());
07402                 }
07403             }
07404

```

```

07405         ObjectStorage(ObjectStorage&& other)
07406         {
07407             new(&data) T(std::move(other.stored_object()));
07408         }
07409
07410         ~ObjectStorage() { destruct_on_exit<T>(); }
07411
07412         template <typename... Args>
07413         void construct(Args&&... args)
07414         {
07415             new (&data) T(std::forward<Args>(args)...);
07416         }
07417
07418         template <bool AllowManualDestruction = !Destruct>
07419         typename std::enable_if<AllowManualDestruction>::type destruct()
07420         {
07421             stored_object().~T();
07422         }
07423
07424     private:
07425         // If this is a constructor benchmark, destruct the underlying object
07426         template <typename U>
07427         void destruct_on_exit(typename std::enable_if<Destruct, U>::type* = 0) {
07428     destruct<true>(); }
07429         // Otherwise, don't
07430         template <typename U>
07431         void destruct_on_exit(typename std::enable_if<!Destruct, U>::type* = 0) { }
07432
07433         T& stored_object() {
07434             return *static_cast<T*>(static_cast<void*>(&data));
07435         }
07436
07437         T const& stored_object() const {
07438             return *static_cast<T*>(static_cast<void*>(&data));
07439         }
07440
07441         struct { alignas(T) unsigned char data[sizeof(T)]; } data;
07442     };
07443
07444     template <typename T>
07445     using storage_for = Detail::ObjectStorage<T, true>;
07446
07447     template <typename T>
07448     using destructable_object = Detail::ObjectStorage<T, false>;
07449 }
07450
07451
07452 // end catch_constructor.hpp
07453 // end catch_benchmarking_all.hpp
07454 #endif
07455
07456 #endif // ! CATCH_CONFIG_IMPL_ONLY
07457
07458 #ifdef CATCH_IMPL
07459 // start catch_impl.hpp
07460
07461 #ifdef __clang__
07462 #pragma clang diagnostic push
07463 #pragma clang diagnostic ignored "-Wweak-vtables"
07464 #endif
07465
07466 // Keep these here for external reporters
07467 // start catch_test_case_tracker.h
07468
07469 #include <string>
07470 #include <vector>
07471 #include <memory>
07472
07473 namespace Catch {
07474     namespace TestCaseTracking {
07475
07476         struct NameAndLocation {
07477             std::string name;
07478             SourceLineInfo location;
07479
07480             NameAndLocation( std::string const& _name, SourceLineInfo const& _location );
07481             friend bool operator==(NameAndLocation const& lhs, NameAndLocation const& rhs) {
07482                 return lhs.name == rhs.name
07483                     && lhs.location == rhs.location;
07484             }
07485         };
07486
07487         class ITracker;
07488
07489         using ITrackerPtr = std::shared_ptr<ITracker>;
07490

```



```

07491     class ITracker {
07492         NameAndLocation m_nameAndLocation;
07493     public:
07494         ITracker(NameAndLocation const& nameAndLoc) :
07495             m_nameAndLocation(nameAndLoc)
07496         {}
07497
07498         // static queries
07499         NameAndLocation const& nameAndLocation() const {
07500             return m_nameAndLocation;
07501         }
07502
07503         virtual ~ITracker();
07504
07505         // dynamic queries
07506         virtual bool isComplete() const = 0; // Successfully completed or failed
07507         virtual bool isSuccessfullyCompleted() const = 0;
07508         virtual bool isOpen() const = 0; // Started but not complete
07509         virtual bool hasChildren() const = 0;
07510         virtual bool hasStarted() const = 0;
07511
07512         virtual ITracker& parent() = 0;
07513
07514         // actions
07515         virtual void close() = 0; // Successfully complete
07516         virtual void fail() = 0;
07517         virtual void markAsNeedingAnotherRun() = 0;
07518
07519         virtual void addChild( ITrackerPtr const& child ) = 0;
07520         virtual ITrackerPtr findChild( NameAndLocation const& nameAndLocation ) = 0;
07521         virtual void openChild() = 0;
07522
07523         // Debug/ checking
07524         virtual bool isSectionTracker() const = 0;
07525         virtual bool isGeneratorTracker() const = 0;
07526     };
07527
07528     class TrackerContext {
07529     public:
07530         enum RunState {
07531             NotStarted,
07532             Executing,
07533             CompletedCycle
07534         };
07535
07536         ITrackerPtr m_rootTracker;
07537         ITracker* m_currentTracker = nullptr;
07538         RunState m_runState = NotStarted;
07539
07540     public:
07541         ITracker& startRun();
07542         void endRun();
07543
07544         void startCycle();
07545         void completeCycle();
07546
07547         bool completedCycle() const;
07548         ITracker& currentTracker();
07549         void setCurrentTracker( ITracker* tracker );
07550     };
07551
07552     class TrackerBase : public ITracker {
07553     protected:
07554         enum CycleState {
07555             NotStarted,
07556             Executing,
07557             ExecutingChildren,
07558             NeedsAnotherRun,
07559             CompletedSuccessfully,
07560             Failed
07561         };
07562
07563         using Children = std::vector<ITrackerPtr>;
07564         TrackerContext& m_ctx;
07565         ITracker* m_parent;
07566         Children m_children;
07567         CycleState m_runState = NotStarted;
07568
07569     public:
07570         TrackerBase( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker* parent
07571     );
07572
07573         bool isComplete() const override;
07574         bool isSuccessfullyCompleted() const override;
07575         bool isOpen() const override;

```

```

07577         bool hasChildren() const override;
07578         bool hasStarted() const override {
07579             return m_runState != NotStarted;
07580         }
07581
07582         void addChild( ITrackerPtr const& child ) override;
07583
07584         ITrackerPtr findChild( NameAndLocation const& nameAndLocation ) override;
07585         ITracker& parent() override;
07586
07587         void openChild() override;
07588
07589         bool isSectionTracker() const override;
07590         bool isGeneratorTracker() const override;
07591
07592         void open();
07593
07594         void close() override;
07595         void fail() override;
07596         void markAsNeedingAnotherRun() override;
07597
07598     private:
07599         void moveToParent();
07600         void moveToThis();
07601     };
07602
07603     class SectionTracker : public TrackerBase {
07604     public:
07605         std::vector<std::string> m_filters;
07606         std::string m_trimmed_name;
07607     public:
07608         SectionTracker( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker*
parent );
07609
07610         bool isSectionTracker() const override;
07611
07612         bool isComplete() const override;
07613
07614         static SectionTracker& acquire( TrackerContext& ctx, NameAndLocation const&
nameAndLocation );
07615
07616         void tryOpen();
07617
07618         void addInitialFilters( std::vector<std::string> const& filters );
07619         void addNextFilters( std::vector<std::string> const& filters );
07620         std::vector<std::string> const& getFilters() const;
07621         std::string const& trimmedName() const;
07622     };
07623 };
07624
07625 } // namespace TestCaseTracking
07626
07627 using TestCaseTracking::ITracker;
07628 using TestCaseTracking::TrackerContext;
07629 using TestCaseTracking::SectionTracker;
07630
07631 } // namespace Catch
07632
07633 // end catch_test_case_tracker.h
07634
07635 // start catch_leak_detector.h
07636
07637 namespace Catch {
07638
07639     struct LeakDetector {
07640         LeakDetector();
07641         ~LeakDetector();
07642     };
07643
07644 }
07645 // end catch_leak_detector.h
07646 // Cpp files will be included in the single-header file here
07647 // start catch_stats.cpp
07648
07649 // Statistical analysis tools
07650
07651 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
07652
07653 #include <cassert>
07654 #include <random>
07655
07656 #if defined(CATCH_CONFIG_USE_ASYNC)
07657 #include <future>
07658 #endif
07659
07660 namespace {
07661     double erf_inv(double x) {
07662         // Code accompanying the article "Approximating the erfinv function" in GPU Computing Gems,
07663         Volume 2

```

```

07663     double w, p;
07664
07665     w = -log((1.0 - x) * (1.0 + x));
07666
07667     if (w < 6.250000) {
07668         w = w - 3.125000;
07669         p = -3.6444120640178196996e-21;
07670         p = -1.685059138182016589e-19 + p * w;
07671         p = 1.2858480715256400167e-18 + p * w;
07672         p = 1.115787767802518096e-17 + p * w;
07673         p = -1.333171662854620906e-16 + p * w;
07674         p = 2.0972767875968561637e-17 + p * w;
07675         p = 6.6376381343583238325e-15 + p * w;
07676         p = -4.0545662729752068639e-14 + p * w;
07677         p = -8.1519341976054721522e-14 + p * w;
07678         p = 2.6335093153082322977e-12 + p * w;
07679         p = -1.2975133253453532498e-11 + p * w;
07680         p = -5.4154120542946279317e-11 + p * w;
07681         p = 1.051212273321532285e-09 + p * w;
07682         p = -4.1126339803469836976e-09 + p * w;
07683         p = -2.9070369957882005086e-08 + p * w;
07684         p = 4.2347877827932403518e-07 + p * w;
07685         p = -1.3654692000834678645e-06 + p * w;
07686         p = -1.3882523362786468719e-05 + p * w;
07687         p = 0.0001867342080340571352 + p * w;
07688         p = -0.00074070253416626697512 + p * w;
07689         p = -0.0060336708714301490533 + p * w;
07690         p = 0.24015818242558961693 + p * w;
07691         p = 1.6536545626831027356 + p * w;
07692     } else if (w < 16.000000) {
07693         w = sqrt(w) - 3.250000;
07694         p = 2.2137376921775787049e-09;
07695         p = 9.0756561938885390979e-08 + p * w;
07696         p = -2.7517406297064545428e-07 + p * w;
07697         p = 1.8239629214389227755e-08 + p * w;
07698         p = 1.5027403968909827627e-06 + p * w;
07699         p = -4.013867526981545969e-06 + p * w;
07700         p = 2.9234449089955446044e-06 + p * w;
07701         p = 1.2475304481671778723e-05 + p * w;
07702         p = -4.7318229009055733981e-05 + p * w;
07703         p = 6.8284851459573175448e-05 + p * w;
07704         p = 2.4031110387097893999e-05 + p * w;
07705         p = -0.0003550375203628474796 + p * w;
07706         p = 0.00095328937973738049703 + p * w;
07707         p = -0.0016882755560235047313 + p * w;
07708         p = 0.0024914420961078508066 + p * w;
07709         p = -0.0037512085075692412107 + p * w;
07710         p = 0.005370914553590063617 + p * w;
07711         p = 1.0052589676941592334 + p * w;
07712         p = 3.0838856104922207635 + p * w;
07713     } else {
07714         w = sqrt(w) - 5.000000;
07715         p = -2.7109920616438573243e-11;
07716         p = -2.5556418169965252055e-10 + p * w;
07717         p = 1.5076572693500548083e-09 + p * w;
07718         p = -3.7894654401267369937e-09 + p * w;
07719         p = 7.6157012080783393804e-09 + p * w;
07720         p = -1.4960026627149240478e-08 + p * w;
07721         p = 2.9147953450901080826e-08 + p * w;
07722         p = -6.7711997758452339498e-08 + p * w;
07723         p = 2.2900482228026654717e-07 + p * w;
07724         p = -9.9298272942317002539e-07 + p * w;
07725         p = 4.5260625972231537039e-06 + p * w;
07726         p = -1.9681778105531670567e-05 + p * w;
07727         p = 7.5995277030017761139e-05 + p * w;
07728         p = -0.00021503011930044477347 + p * w;
07729         p = -0.00013871931833623122026 + p * w;
07730         p = 1.0103004648645343977 + p * w;
07731         p = 4.8499064014085844221 + p * w;
07732     }
07733     return p * x;
07734 }
07735
07736 double standard_deviation(std::vector<double>::iterator first, std::vector<double>::iterator last)
07737 {
07738     auto m = Catch::Benchmark::Detail::mean(first, last);
07739     double variance = std::accumulate(first, last, 0., [m](double a, double b) {
07740         double diff = b - m;
07741         return a + diff * diff;
07742     }) / (last - first);
07743     return std::sqrt(variance);
07744 }
07745 }
07746
07747 namespace Catch {
07748     namespace Benchmark {

```

```

07749     namespace Detail {
07750
07751         double weighted_average_quantile(int k, int q, std::vector<double>::iterator first,
std::vector<double>::iterator last) {
07752             auto count = last - first;
07753             double idx = (count - 1) * k / static_cast<double>(q);
07754             int j = static_cast<int>(idx);
07755             double g = idx - j;
07756             std::nth_element(first, first + j, last);
07757             auto xj = first[j];
07758             if (g == 0) return xj;
07759
07760             auto xj1 = *std::min_element(first + (j + 1), last);
07761             return xj + g * (xj1 - xj);
07762         }
07763
07764         double erfc_inv(double x) {
07765             return erf_inv(1.0 - x);
07766         }
07767
07768         double normal_quantile(double p) {
07769             static const double ROOT_TWO = std::sqrt(2.0);
07770
07771             double result = 0.0;
07772             assert(p >= 0 && p <= 1);
07773             if (p < 0 || p > 1) {
07774                 return result;
07775             }
07776
07777             result = -erfc_inv(2.0 * p);
07778             // result *= normal distribution standard deviation (1.0) * sqrt(2)
07779             result *= /*sd */ ROOT_TWO;
07780             // result += normal distribution mean (0)
07781             return result;
07782         }
07783
07784         double outlier_variance(Estimate<double> mean, Estimate<double> stddev, int n) {
07785             double sb = stddev.point;
07786             double mn = mean.point / n;
07787             double mg_min = mn / 2.;
07788             double sg = (std::min)(mg_min / 4., sb / std::sqrt(n));
07789             double sg2 = sg * sg;
07790             double sb2 = sb * sb;
07791
07792             auto c_max = [n, mn, sb2, sg2](double x) -> double {
07793                 double k = mn - x;
07794                 double d = k * k;
07795                 double nd = n * d;
07796                 double k0 = -n * nd;
07797                 double k1 = sb2 - n * sg2 + nd;
07798                 double det = k1 * k1 - 4 * sg2 * k0;
07799                 return (int)(-2. * k0 / (k1 + std::sqrt(det)));
07800             };
07801
07802             auto var_out = [n, sb2, sg2](double c) {
07803                 double nc = n - c;
07804                 return (nc / n) * (sb2 - nc * sg2);
07805             };
07806
07807             return (std::min)(var_out(1), var_out((std::min)(c_max(0.), c_max(mg_min)))) / sb2;
07808         }
07809
07810         bootstrap_analysis analyse_samples(double confidence_level, int n_resamples,
std::vector<double>::iterator first, std::vector<double>::iterator last) {
07811             CATCH_INTERNAL_START_WARNINGS_SUPPRESSION
07812             CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS
07813             static std::random_device entropy;
07814             CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
07815
07816             auto n = static_cast<int>(last - first); // seriously, one can't use integral types
without hell in C++
07817
07818             auto mean = &Detail::mean<std::vector<double>::iterator>;
07819             auto stddev = &standard_deviation;
07820
07821             #if defined(CATCH_CONFIG_USE_ASYNC)
07822             auto Estimate = [=](double(*f)(std::vector<double>::iterator,
std::vector<double>::iterator)) {
07823                 auto seed = entropy();
07824                 return std::async(std::launch::async, [=] {
07825                     std::mt19937 rng(seed);
07826                     auto resampled = resample(rng, n_resamples, first, last, f);
07827                     return bootstrap(confidence_level, first, last, resampled, f);
07828                 });
07829             };
07830
07831             auto mean_future = Estimate(mean);

```

```

07832         auto stddev_future = Estimate(stddev);
07833
07834         auto mean_estimate = mean_future.get();
07835         auto stddev_estimate = stddev_future.get();
07836     #else
07837         auto Estimate = [=](double(*f)(std::vector<double>::iterator,
std::vector<double>::iterator)) {
07838             auto seed = entropy();
07839             std::mt19937 rng(seed);
07840             auto resampled = resample(rng, n_resamples, first, last, f);
07841             return bootstrap(confidence_level, first, last, resampled, f);
07842         };
07843
07844         auto mean_estimate = Estimate(mean);
07845         auto stddev_estimate = Estimate(stddev);
07846     #endif // CATCH_USE_ASYNC
07847
07848     double outlier_variance = Detail::outlier_variance(mean_estimate, stddev_estimate, n);
07849
07850     return { mean_estimate, stddev_estimate, outlier_variance };
07851 }
07852 } // namespace Detail
07853 } // namespace Benchmark
07854 } // namespace Catch
07855
07856 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
07857 // end catch_stats.cpp
07858 // start catch_approx.cpp
07859
07860 #include <cmath>
07861 #include <limits>
07862
07863 namespace {
07864
07865     // Performs equivalent check of std::fabs(lhs - rhs) <= margin
07866     // But without the subtraction to allow for INFINITY in comparison
07867     bool marginComparison(double lhs, double rhs, double margin) {
07868         return (lhs + margin >= rhs) && (rhs + margin >= lhs);
07869     }
07870
07871 }
07872
07873 namespace Catch {
07874     namespace Detail {
07875
07876         Approx::Approx ( double value )
07877             :   m_epsilon( std::numeric_limits<float>::epsilon()*100 ),
07878                 m_margin( 0.0 ),
07879                 m_scale( 0.0 ),
07880                 m_value( value )
07881         {}
07882
07883         Approx Approx::custom() {
07884             return Approx( 0 );
07885         }
07886
07887         Approx Approx::operator-() const {
07888             auto temp(*this);
07889             temp.m_value = -temp.m_value;
07890             return temp;
07891         }
07892
07893         std::string Approx::toString() const {
07894             ReusableStringStream rss;
07895             rss << "Approx( " << ::Catch::Detail::stringify( m_value ) << " )";
07896             return rss.str();
07897         }
07898
07899         bool Approx::equalityComparisonImpl(const double other) const {
07900             // First try with fixed margin, then compute margin based on epsilon, scale and Approx's
07901             value
07902             // Thanks to Richard Harris for his help refining the scaled margin value
07903             return marginComparison(m_value, other, m_margin)
07904                 || marginComparison(m_value, other, m_epsilon * (m_scale +
std::fabs(std::isinf(m_value)? 0 : m_value)));
07905         }
07906
07907         void Approx::setMargin(double newMargin) {
07908             CATCH_ENFORCE(newMargin >= 0,
07909                 "Invalid Approx::margin: " << newMargin << ".'"
07909                 << " Approx::Margin has to be non-negative.");
07910             m_margin = newMargin;
07911         }
07912
07913         void Approx::setEpsilon(double newEpsilon) {
07914             CATCH_ENFORCE(newEpsilon >= 0 && newEpsilon <= 1.0,
07915                 "Invalid Approx::epsilon: " << newEpsilon << ".')

```

```

07916                                     « " Approx::epsilon has to be in [0, 1]");
07917         m_epsilon = newEpsilon;
07918     }
07919 } // end namespace Detail
07920
07921 namespace literals {
07922     Detail::Approx operator "" _a(long double val) {
07923         return Detail::Approx(val);
07924     }
07925     Detail::Approx operator "" _a(unsigned long long val) {
07926         return Detail::Approx(val);
07927     }
07928 } // end namespace literals
07929
07930 std::string StringMaker<Catch::Detail::Approx>::convert(Catch::Detail::Approx const& value) {
07931     return value.toString();
07932 }
07933
07934 } // end namespace Catch
07935 // end catch_approx.cpp
07936 // start catch_assertionhandler.cpp
07937
07938 // start catch_debugger.h
07939
07940 namespace Catch {
07941     bool isDebuggerActive();
07942 }
07943
07944 #ifdef CATCH_PLATFORM_MAC
07945 #if defined(__i386__) || defined(__x86_64__)
07946 #define CATCH_TRAP() __asm__("int $3\n" : : ) /* NOLINT */
07947 #elif defined(__aarch64__)
07948 #define CATCH_TRAP() __asm__(".inst 0xd43e0000")
07949 #endif
07950 #endif
07951
07952 #elif defined(CATCH_PLATFORM_IPHONE)
07953 // use inline assembler
07954 #if defined(__i386__) || defined(__x86_64__)
07955 #define CATCH_TRAP() __asm__("int $3")
07956 #elif defined(__aarch64__)
07957 #define CATCH_TRAP() __asm__(".inst 0xd4200000")
07958 #elif defined(__arm__) && !defined(__thumb__)
07959 #define CATCH_TRAP() __asm__(".inst 0xe7f001f0")
07960 #elif defined(__arm__) && defined(__thumb__)
07961 #define CATCH_TRAP() __asm__(".inst 0xde01")
07962 #endif
07963
07964 #elif defined(CATCH_PLATFORM_LINUX)
07965 // If we can use inline assembler, do it because this allows us to break
07966 // directly at the location of the failing check instead of breaking inside
07967 // raise() called from it, i.e. one stack frame below.
07968 #if defined(__GNUC__) && (defined(__i386__) || defined(__x86_64__))
07969 #define CATCH_TRAP() asm volatile ("int $3") /* NOLINT */
07970 #else // Fall back to the generic way.
07971 #include <signal.h>
07972
07973 #define CATCH_TRAP() raise(SIGTRAP)
07974 #endif
07975
07976 #elif defined(_MSC_VER)
07977 #define CATCH_TRAP() __debugbreak()
07978 #elif defined(__MINGW32__)
07979 extern "C" __declspec(dllimport) void __stdcall DebugBreak();
07980 #define CATCH_TRAP() DebugBreak()
07981 #endif
07982
07983 #ifndef CATCH_BREAK_INTO_DEBUGGER
07984 #define CATCH_BREAK_INTO_DEBUGGER() []{ if( Catch::isDebuggerActive() ) { CATCH_TRAP(); } }()
07985 #else
07986 #define CATCH_BREAK_INTO_DEBUGGER() []{}()
07987 #endif
07988
07989 #include <cassert>
07990
07991 namespace Catch {
07992     // Wrapper for platform-specific fatal error (signals/SEH) handlers
07993     //

```

```

08003 // Tries to be cooperative with other handlers, and not step over
08004 // other handlers. This means that unknown structured exceptions
08005 // are passed on, previous signal handlers are called, and so on.
08006 //
08007 // Can only be instantiated once, and assumes that once a signal
08008 // is caught, the binary will end up terminating. Thus, there
08009 class FatalConditionHandler {
08010     bool m_started = false;
08011
08012     // Install/disengage implementation for specific platform.
08013     // Should be if-defed to work on current platform, can assume
08014     // engage-disengage 1:1 pairing.
08015     void engage_platform();
08016     void disengage_platform();
08017 public:
08018     // Should also have platform-specific implementations as needed
08019     FatalConditionHandler();
08020     ~FatalConditionHandler();
08021
08022     void engage() {
08023         assert(!m_started && "Handler cannot be installed twice.");
08024         m_started = true;
08025         engage_platform();
08026     }
08027
08028     void disengage() {
08029         assert(m_started && "Handler cannot be uninstalled without being installed first");
08030         m_started = false;
08031         disengage_platform();
08032     }
08033 };
08034
08035 class FatalConditionHandlerGuard {
08036     FatalConditionHandler* m_handler;
08037 public:
08038     FatalConditionHandlerGuard(FatalConditionHandler* handler):
08039         m_handler(handler) {
08040         m_handler->engage();
08041     }
08042     ~FatalConditionHandlerGuard() {
08043         m_handler->disengage();
08044     }
08045 };
08046 };
08047
08048 } // end namespace Catch
08049
08050 // end catch_fatal_condition.h
08051 #include <string>
08052
08053 namespace Catch {
08054
08055     struct IMutableContext;
08056
08057
08058     class RunContext : public IResultCapture, public IRunner {
08059 public:
08060         RunContext( RunContext const& ) = delete;
08061         RunContext& operator =( RunContext const& ) = delete;
08062
08063         explicit RunContext( IConfigPtr const& _config, IStreamingReporterPtr&& reporter );
08064
08065         ~RunContext() override;
08066
08067         void testGroupStarting( std::string const& testSpec, std::size_t groupIndex, std::size_t
groupsCount );
08070         void testGroupEnded( std::string const& testSpec, Totals const& totals, std::size_t
groupIndex, std::size_t groupsCount );
08071
08072         Totals runTest(TestCase const& testCase);
08073
08074         IConfigPtr config() const;
08075         IStreamingReporter& reporter() const;
08076
08077     public: // IResultCapture
08078
08079         // Assertion handlers
08080         void handleExpr
08081             ( AssertionInfo const& info,
08082               ITransientExpression const& expr,
08083               AssertionReaction& reaction ) override;
08084         void handleMessage
08085             ( AssertionInfo const& info,
08086               ResultWas::OfType resultType,
08087               StringRef const& message,
08088               AssertionReaction& reaction ) override;
08089         void handleUnexpectedExceptionNotThrown

```

```

08090         ( AssertionInfo const& info,
08091           AssertionReaction& reaction ) override;
08092 void handleUnexpectedInflightException
08093     ( AssertionInfo const& info,
08094       std::string const& message,
08095       AssertionReaction& reaction ) override;
08096 void handleIncomplete
08097     ( AssertionInfo const& info ) override;
08098 void handleNonExpr
08099     ( AssertionInfo const& info,
08100       ResultWas::OfType resultType,
08101       AssertionReaction& reaction ) override;
08102
08103 bool sectionStarted( SectionInfo const& sectionInfo, Counts& assertions ) override;
08104
08105 void sectionEnded( SectionEndInfo const& endInfo ) override;
08106 void sectionEndedEarly( SectionEndInfo const& endInfo ) override;
08107
08108 auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
    IGeneratorTracker& override;
08109
08110 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
08111 void benchmarkPreparing( std::string const& name ) override;
08112 void benchmarkStarting( BenchmarkInfo const& info ) override;
08113 void benchmarkEnded( BenchmarkStats<> const& stats ) override;
08114 void benchmarkFailed( std::string const& error ) override;
08115 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
08116
08117 void pushScopedMessage( MessageInfo const& message ) override;
08118 void popScopedMessage( MessageInfo const& message ) override;
08119
08120 void emplaceUnscopedMessage( MessageBuilder const& builder ) override;
08121
08122 std::string getCurrentTestName() const override;
08123
08124 const AssertionResult* getLastResult() const override;
08125
08126 void exceptionEarlyReported() override;
08127
08128 void handleFatalErrorCondition( StringRef message ) override;
08129
08130 bool lastAssertionPassed() override;
08131
08132 void assertionPassed() override;
08133
08134 public:
08135     // !TBD We need to do this another way!
08136     bool aborting() const final;
08137
08138 private:
08139
08140     void runCurrentTest( std::string& redirectedCout, std::string& redirectedCerr );
08141     void invokeActiveTestCase();
08142
08143     void resetAssertionInfo();
08144     bool testForMissingAssertions( Counts& assertions );
08145
08146     void assertionEnded( AssertionResult const& result );
08147     void reportExpr
08148         ( AssertionInfo const& info,
08149           ResultWas::OfType resultType,
08150           ITransientExpression const* expr,
08151           bool negated );
08152
08153     void populateReaction( AssertionReaction& reaction );
08154
08155 private:
08156
08157     void handleUnfinishedSections();
08158
08159     TestRunInfo m_runInfo;
08160     IMutableContext& m_context;
08161     TestCase const* m_activeTestCase = nullptr;
08162     ITracker* m_testCaseTracker = nullptr;
08163     Option<AssertionResult> m_lastResult;
08164
08165     IConfigPtr m_config;
08166     Totals m_totals;
08167     IStreamingReporterPtr m_reporter;
08168     std::vector<MessageInfo> m_messages;
08169     std::vector<ScopedMessage> m_messageScopes; /* Keeps owners of so-called unscoped messages. */
08170     AssertionInfo m_lastAssertionInfo;
08171     std::vector<SectionEndInfo> m_unfinishedSections;
08172     std::vector<ITracker*> m_activeSections;
08173     TrackerContext m_trackerContext;
08174     FatalConditionHandler m_fatalConditionhandler;
08175     bool m_lastAssertionPassed = false;

```



```

08176         bool m_shouldReportUnexpected = true;
08177         bool m_includeSuccessfulResults;
08178     };
08179
08180     void seedRng(IConfig const& config);
08181     unsigned int rngSeed();
08182 } // end namespace Catch
08183
08184 // end catch_run_context.h
08185 namespace Catch {
08186
08187     namespace {
08188         auto operator «( std::ostream& os, ITransientExpression const& expr ) -> std::ostream& {
08189             expr.streamReconstructedExpression( os );
08190             return os;
08191         }
08192     }
08193
08194     LazyExpression::LazyExpression( bool isNegated )
08195         :   m_isNegated( isNegated )
08196     {}
08197
08198     LazyExpression::LazyExpression( LazyExpression const& other ) : m_isNegated( other.m_isNegated )
08199 {}
08200
08201     LazyExpression::operator bool() const {
08202         return m_transientExpression != nullptr;
08203     }
08204
08205     auto operator « ( std::ostream& os, LazyExpression const& lazyExpr ) -> std::ostream& {
08206         if( lazyExpr.m_isNegated )
08207             os << "!";
08208
08209         if( lazyExpr ) {
08210             if( lazyExpr.m_isNegated && lazyExpr.m_transientExpression->isBinaryExpression() )
08211                 os << "(" << *lazyExpr.m_transientExpression << ")";
08212             else
08213                 os << *lazyExpr.m_transientExpression;
08214         }
08215         else {
08216             os << "{** error - unchecked empty expression requested **}";
08217         }
08218         return os;
08219     }
08220
08221     AssertionHandler::AssertionHandler
08222     (   StringRef const& macroName,
08223         SourceLineInfo const& lineInfo,
08224         StringRef capturedExpression,
08225         ResultDisposition::Flags resultDisposition )
08226     :   m_assertionInfo{ macroName, lineInfo, capturedExpression, resultDisposition },
08227         m_resultCapture( getResultCapture() )
08228     {}
08229
08230     void AssertionHandler::handleExpr( ITransientExpression const& expr ) {
08231         m_resultCapture.handleExpr( m_assertionInfo, expr, m_reaction );
08232     }
08233
08234     void AssertionHandler::handleMessage(ResultWas::OfType resultType, StringRef const& message) {
08235         m_resultCapture.handleMessage( m_assertionInfo, resultType, message, m_reaction );
08236     }
08237
08238     auto AssertionHandler::allowThrows() const -> bool {
08239         return getCurrentContext().getConfig()->allowThrows();
08240     }
08241
08242     void AssertionHandler::complete() {
08243         setCompleted();
08244         if( m_reaction.shouldDebugBreak ) {
08245             // If you find your debugger stopping you here then go one level up on the
08246             // call-stack for the code that caused it (typically a failed assertion)
08247
08248             // (To go back to the test and change execution, jump over the throw, next)
08249             CATCH_BREAK_INTO_DEBUGGER();
08250         }
08251         if (m_reaction.shouldThrow) {
08252             #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
08253             throw Catch::TestFailureException();
08254             #else
08255             CATCH_ERROR( "Test failure requires aborting test!" );
08256             #endif
08257         }
08258     }
08259
08260     void AssertionHandler::setCompleted() {
08261         m_completed = true;
08262     }

```

```

08262     void AssertionHandler::handleUnexpectedInflightException() {
08263         m_resultCapture.handleUnexpectedInflightException( m_assertionInfo,
Catch::translateActiveException(), m_reaction );
08264     }
08265
08266     void AssertionHandler::handleExceptionThrownAsExpected() {
08267         m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
08268     }
08269     void AssertionHandler::handleExceptionNotThrownAsExpected() {
08270         m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
08271     }
08272
08273     void AssertionHandler::handleUnexpectedExceptionNotThrown() {
08274         m_resultCapture.handleUnexpectedExceptionNotThrown( m_assertionInfo, m_reaction );
08275     }
08276
08277     void AssertionHandler::handleThrowingCallSkipped() {
08278         m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
08279     }
08280
08281     // This is the overload that takes a string and infers the Equals matcher from it
08282     // The more general overload, that takes any string matcher, is in catch_capture_matchers.cpp
08283     void handleExceptionMatchExpr( AssertionHandler& handler, std::string const& str, StringRef const&
matcherString ) {
08284         handleExceptionMatchExpr( handler, Matchers::Equals( str ), matcherString );
08285     }
08286
08287 } // namespace Catch
08288 // end catch_assertionhandler.cpp
08289 // start catch_assertionresult.cpp
08290
08291 namespace Catch {
08292     AssertionResultData::AssertionResultData(ResultWas::OfType _resultType, LazyExpression const &
_lazyExpression):
08293         lazyExpression(_lazyExpression),
08294         resultType(_resultType) {}
08295
08296     std::string AssertionResultData::reconstructExpression() const {
08297
08298         if( reconstructedExpression.empty() ) {
08299             if( lazyExpression ) {
08300                 ReusableStringStream rss;
08301                 rss << lazyExpression;
08302                 reconstructedExpression = rss.str();
08303             }
08304         }
08305         return reconstructedExpression;
08306     }
08307
08308     AssertionResult::AssertionResult( AssertionInfo const& info, AssertionResultData const& data )
08309         :   m_info( info ),
08310             m_resultData( data )
08311     {}
08312
08313     // Result was a success
08314     bool AssertionResult::succeeded() const {
08315         return Catch::isOk( m_resultData.resultType );
08316     }
08317
08318     // Result was a success, or failure is suppressed
08319     bool AssertionResult::isOk() const {
08320         return Catch::isOk( m_resultData.resultType ) || shouldSuppressFailure(
m_info.resultDisposition );
08321     }
08322
08323     ResultWas::OfType AssertionResult::getResultType() const {
08324         return m_resultData.resultType;
08325     }
08326
08327     bool AssertionResult::hasExpression() const {
08328         return !m_info.capturedExpression.empty();
08329     }
08330
08331     bool AssertionResult::hasMessage() const {
08332         return !m_resultData.message.empty();
08333     }
08334
08335     std::string AssertionResult::getExpression() const {
08336         // Possibly overallocating by 3 characters should be basically free
08337         std::string expr; expr.reserve(m_info.capturedExpression.size() + 3);
08338         if (isFalseTest(m_info.resultDisposition)) {
08339             expr += "!(";
08340         }
08341         expr += m_info.capturedExpression;
08342         if (isFalseTest(m_info.resultDisposition)) {
08343             expr += ')';
08344         }
08345     }

```

```

08345         return expr;
08346     }
08347
08348     std::string AssertionResult::getExpressionInMacro() const {
08349         std::string expr;
08350         if( m_info.macroName.empty() )
08351             expr = static_cast<std::string>(m_info.capturedExpression);
08352         else {
08353             expr.reserve( m_info.macroName.size() + m_info.capturedExpression.size() + 4 );
08354             expr += m_info.macroName;
08355             expr += " ( ";
08356             expr += m_info.capturedExpression;
08357             expr += " ) ";
08358         }
08359         return expr;
08360     }
08361
08362     bool AssertionResult::hasExpandedExpression() const {
08363         return hasExpression() && getExpandedExpression() != getExpression();
08364     }
08365
08366     std::string AssertionResult::getExpandedExpression() const {
08367         std::string expr = m_resultData.reconstructExpression();
08368         return expr.empty()
08369             ? getExpression()
08370             : expr;
08371     }
08372
08373     std::string AssertionResult::getMessage() const {
08374         return m_resultData.message;
08375     }
08376
08377     SourceLineInfo AssertionResult::getSourceInfo() const {
08378         return m_info.lineInfo;
08379     }
08380
08381     StringRef AssertionResult::getTestMacroName() const {
08382         return m_info.macroName;
08383     }
08384 } // end namespace Catch
08385 // end catch_assertionresult.cpp
08386 // start catch_capture_matchers.cpp
08387
08388 namespace Catch {
08389
08390     using StringMatcher = Matchers::Impl::MatcherBase<std::string>;
08391
08392     // This is the general overload that takes a any string matcher
08393     // There is another overload, in catch_assertionhandler.h/.cpp, that only takes a string and
08394     // infers
08395     // the Equals matcher (so the header does not mention matchers)
08396     void handleExceptionMatchExpr( AssertionHandler& handler, StringMatcher const& matcher, StringRef
08397     const& matcherString ) {
08398         std::string exceptionMessage = Catch::translateActiveException();
08399         MatchExpr<std::string, StringMatcher const&> expr( exceptionMessage, matcher, matcherString );
08400         handler.handleExpr( expr );
08401     }
08402 } // namespace Catch
08403 // end catch_capture_matchers.cpp
08404 // start catch_commandline.cpp
08405
08406 // start catch_commandline.h
08407
08408 // Use Catch's value for console width (store Clara's off to the side, if present)
08409 #ifdef CLARA_CONFIG_CONSOLE_WIDTH
08410 #define CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
08411 #undef CATCH_CLARA_CONFIG_CONSOLE_WIDTH
08412 #endif
08413 #define CATCH_CLARA_CONFIG_CONSOLE_WIDTH CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
08414 #define CATCH_CLARA_CONFIG_CONSOLE_WIDTH CATCH_CONFIG_CONSOLE_WIDTH-1
08415
08416 #ifdef __clang__
08417 #pragma clang diagnostic push
08418 #pragma clang diagnostic ignored "-Wweak-vtables"
08419 #pragma clang diagnostic ignored "-Wexit-time-destructors"
08420 #pragma clang diagnostic ignored "-Wshadow"
08421 #endif
08422
08423 // start clara.hpp
08424 // Copyright 2017 Two Blue Cubes Ltd. All rights reserved.
08425 //
08426 // Distributed under the Boost Software License, Version 1.0. (See accompanying
08427 // file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
08428 //
08429 // See https://github.com/philsquared/Clara for more details

```

```

08430
08431 // Clara v1.1.5
08432
08433
08434 #ifndef CATCH_CLARA_CONFIG_CONSOLE_WIDTH
08435 #define CATCH_CLARA_CONFIG_CONSOLE_WIDTH 80
08436 #endif
08437
08438 #ifndef CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
08439 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH CATCH_CLARA_CONFIG_CONSOLE_WIDTH
08440 #endif
08441
08442 #ifndef CLARA_CONFIG_OPTIONAL_TYPE
08443 #ifdef __has_include
08444 #if __has_include(<optional>) && __cplusplus >= 201703L
08445 #include <optional>
08446 #define CLARA_CONFIG_OPTIONAL_TYPE std::optional
08447 #endif
08448 #endif
08449 #endif
08450
08451 // ----- #included from clara_textflow.hpp -----
08452
08453 // TextFlowCpp
08454 //
08455 // A single-header library for wrapping and laying out basic text, by Phil Nash
08456 //
08457 // Distributed under the Boost Software License, Version 1.0. (See accompanying
08458 // file LICENSE.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
08459 //
08460 // This project is hosted at https://github.com/philsquared/textflowcpp
08461
08462
08463 #include <cassert>
08464 #include <ostream>
08465 #include <sstream>
08466 #include <vector>
08467
08468 #ifndef CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
08469 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH 80
08470 #endif
08471
08472 namespace Catch {
08473     namespace clara {
08474         namespace TextFlow {
08475
08476             inline auto isWhitespace(char c) -> bool {
08477                 static std::string chars = " \t\n\r";
08478                 return chars.find(c) != std::string::npos;
08479             }
08480             inline auto isBreakableBefore(char c) -> bool {
08481                 static std::string chars = "[{(<|";
08482                 return chars.find(c) != std::string::npos;
08483             }
08484             inline auto isBreakableAfter(char c) -> bool {
08485                 static std::string chars = "])>.,;+*=&/\\\"";
08486                 return chars.find(c) != std::string::npos;
08487             }
08488
08489             class Columns;
08490
08491             class Column {
08492             public:
08493                 std::vector<std::string> m_strings;
08494                 size_t m_width = CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH;
08495                 size_t m_indent = 0;
08496                 size_t m_initialIndent = std::string::npos;
08497
08498                 class iterator {
08499                 public:
08500                     Column const& m_column;
08501                     size_t m_stringIndex = 0;
08502                     size_t m_pos = 0;
08503
08504                     size_t m_len = 0;
08505                     size_t m_end = 0;
08506                     bool m_suffix = false;
08507
08508                     iterator(Column const& column, size_t stringIndex)
08509                         : m_column(column),
08510                         m_stringIndex(stringIndex) {}
08511
08512                     auto line() const -> std::string const& { return
08513 m_column.m_strings[m_stringIndex]; }
08514
08515                     auto isBoundary(size_t at) const -> bool {

```

```

08516         assert(at > 0);
08517         assert(at <= line().size());
08518
08519         return at == line().size() ||
08520                (isWhitespace(line()[at]) && !isWhitespace(line()[at - 1])) ||
08521                isBreakableBefore(line()[at]) ||
08522                isBreakableAfter(line()[at - 1]);
08523     }
08524
08525     void calcLength() {
08526         assert(m_stringIndex < m_column.m_strings.size());
08527
08528         m_suffix = false;
08529         auto width = m_column.m_width - indent();
08530         m_end = m_pos;
08531         if (line()[m_pos] == '\n') {
08532             ++m_end;
08533         }
08534         while (m_end < line().size() && line()[m_end] != '\n')
08535             ++m_end;
08536
08537         if (m_end < m_pos + width) {
08538             m_len = m_end - m_pos;
08539         } else {
08540             size_t len = width;
08541             while (len > 0 && !isBoundary(m_pos + len))
08542                 --len;
08543             while (len > 0 && isWhitespace(line()[m_pos + len - 1]))
08544                 --len;
08545
08546             if (len > 0) {
08547                 m_len = len;
08548             } else {
08549                 m_suffix = true;
08550                 m_len = width - 1;
08551             }
08552         }
08553     }
08554
08555     auto indent() const -> size_t {
08556         auto initial = m_pos == 0 && m_stringIndex == 0 ? m_column.m_initialIndent :
std::string::npos;
08557         return initial == std::string::npos ? m_column.m_indent : initial;
08558     }
08559
08560     auto addIndentAndSuffix(std::string const &plain) const -> std::string {
08561         return std::string(indent(), ' ') + (m_suffix ? plain + "-" : plain);
08562     }
08563
08564     public:
08565         using difference_type = std::ptrdiff_t;
08566         using value_type = std::string;
08567         using pointer = value_type *;
08568         using reference = value_type &;
08569         using iterator_category = std::forward_iterator_tag;
08570
08571     explicit iterator(Column const& column) : m_column(column) {
08572         assert(m_column.m_width > m_column.m_indent);
08573         assert(m_column.m_initialIndent == std::string::npos || m_column.m_width >
m_column.m_initialIndent);
08574         calcLength();
08575         if (m_len == 0)
08576             m_stringIndex++; // Empty string
08577     }
08578
08579     auto operator*() const -> std::string {
08580         assert(m_stringIndex < m_column.m_strings.size());
08581         assert(m_pos <= m_end);
08582         return addIndentAndSuffix(line().substr(m_pos, m_len));
08583     }
08584
08585     auto operator++() -> iterator& {
08586         m_pos += m_len;
08587         if (m_pos < line().size() && line()[m_pos] == '\n')
08588             m_pos += 1;
08589         else
08590             while (m_pos < line().size() && isWhitespace(line()[m_pos]))
08591                 ++m_pos;
08592
08593         if (m_pos == line().size()) {
08594             m_pos = 0;
08595             ++m_stringIndex;
08596         }
08597         if (m_stringIndex < m_column.m_strings.size())
08598             calcLength();
08599         return *this;
08600     }

```

```

08601         auto operator ++(int) -> iterator {
08602             iterator prev(*this);
08603             operator++();
08604             return prev;
08605         }
08606
08607         auto operator ==(iterator const& other) const -> bool {
08608             return
08609                 m_pos == other.m_pos &&
08610                 m_stringIndex == other.m_stringIndex &&
08611                 &m_column == &other.m_column;
08612         }
08613         auto operator !=(iterator const& other) const -> bool {
08614             return !operator==(other);
08615         }
08616     };
08617     using const_iterator = iterator;
08618
08619     explicit Column(std::string const& text) { m_strings.push_back(text); }
08620
08621     auto width(size_t newWidth) -> Column& {
08622         assert(newWidth > 0);
08623         m_width = newWidth;
08624         return *this;
08625     }
08626     auto indent(size_t newIndent) -> Column& {
08627         m_indent = newIndent;
08628         return *this;
08629     }
08630     auto initialIndent(size_t newIndent) -> Column& {
08631         m_initialIndent = newIndent;
08632         return *this;
08633     }
08634
08635     auto width() const -> size_t { return m_width; }
08636     auto begin() const -> iterator { return iterator(*this); }
08637     auto end() const -> iterator { return { *this, m_strings.size() }; }
08638
08639     inline friend std::ostream& operator << (std::ostream& os, Column const& col) {
08640         bool first = true;
08641         for (auto line : col) {
08642             if (first)
08643                 first = false;
08644             else
08645                 os << "\n";
08646             os << line;
08647         }
08648         return os;
08649     }
08650
08651     auto operator + (Column const& other)->Columns;
08652
08653     auto toString() const -> std::string {
08654         std::ostringstream oss;
08655         oss << *this;
08656         return oss.str();
08657     }
08658 };
08659
08660 class Spacer : public Column {
08661 public:
08662     explicit Spacer(size_t spaceWidth) : Column("") {
08663         width(spaceWidth);
08664     }
08665 };
08666
08667 class Columns {
08668     std::vector<Column> m_columns;
08669 public:
08670
08671     class iterator {
08672     friend Columns;
08673     struct EndTag {};
08674
08675     std::vector<Column> const& m_columns;
08676     std::vector<Column::iterator> m_iterators;
08677     size_t m_activeIterators;
08678
08679     iterator(Columns const& columns, EndTag)
08680         : m_columns(columns.m_columns),
08681           m_activeIterators(0) {
08682         m_iterators.reserve(m_columns.size());
08683         for (auto const& col : m_columns)
08684             m_iterators.push_back(col.end());
08685     }

```

```

08688     }
08689
08690 public:
08691     using difference_type = std::ptrdiff_t;
08692     using value_type = std::string;
08693     using pointer = value_type * ;
08694     using reference = value_type & ;
08695     using iterator_category = std::forward_iterator_tag;
08696
08697     explicit iterator(Columns const& columns)
08698         : m_columns(columns.m_columns),
08699           m_activeIterators(m_columns.size()) {
08700         m_iterators.reserve(m_columns.size());
08701
08702         for (auto const& col : m_columns)
08703             m_iterators.push_back(col.begin());
08704     }
08705
08706     auto operator==(iterator const& other) const -> bool {
08707         return m_iterators == other.m_iterators;
08708     }
08709     auto operator!=(iterator const& other) const -> bool {
08710         return m_iterators != other.m_iterators;
08711     }
08712     auto operator*() const -> std::string {
08713         std::string row, padding;
08714
08715         for (size_t i = 0; i < m_columns.size(); ++i) {
08716             auto width = m_columns[i].width();
08717             if (m_iterators[i] != m_columns[i].end()) {
08718                 std::string col = *m_iterators[i];
08719                 row += padding + col;
08720                 if (col.size() < width)
08721                     padding = std::string(width - col.size(), ' ');
08722             }
08723             else
08724                 padding = "";
08725             padding += std::string(width, ' ');
08726         }
08727         return row;
08728     }
08729
08730     auto operator++() -> iterator& {
08731         for (size_t i = 0; i < m_columns.size(); ++i) {
08732             if (m_iterators[i] != m_columns[i].end())
08733                 ++m_iterators[i];
08734         }
08735         return *this;
08736     }
08737     auto operator++(int) -> iterator {
08738         iterator prev(*this);
08739         operator++;
08740         return prev;
08741     }
08742 };
08743 using const_iterator = iterator;
08744
08745 auto begin() const -> iterator { return iterator(*this); }
08746 auto end() const -> iterator { return { *this, iterator::EndTag() }; }
08747
08748 auto operator+=(Column const& col) -> Columns& {
08749     m_columns.push_back(col);
08750     return *this;
08751 }
08752 auto operator+(Column const& col) -> Columns {
08753     Columns combined = *this;
08754     combined += col;
08755     return combined;
08756 }
08757
08758 inline friend std::ostream& operator<< (std::ostream& os, Columns const& cols) {
08759
08760     bool first = true;
08761     for (auto line : cols) {
08762         if (first)
08763             first = false;
08764         else
08765             os << "\n";
08766         os << line;
08767     }
08768     return os;
08769 }
08770
08771 auto toString() const -> std::string {
08772     std::ostringstream oss;
08773     oss << *this;
08774     return oss.str();

```

```

08775         }
08776     };
08777
08778     inline auto Column::operator + (Column const& other) -> Columns {
08779         Columns cols;
08780         cols += *this;
08781         cols += other;
08782         return cols;
08783     }
08784 }
08785
08786 }
08787 }
08788
08789 // ----- end of #include from clara_textflow.hpp -----
08790 // ..... back in clara.hpp
08791
08792 #include <cctype>
08793 #include <string>
08794 #include <memory>
08795 #include <set>
08796 #include <algorithm>
08797
08798 #if !defined(CATCH_PLATFORM_WINDOWS) && ( defined(WIN32) || defined(__WIN32__) || defined(_WIN32) ||
    defined(_MSC_VER) )
08799 #define CATCH_PLATFORM_WINDOWS
08800 #endif
08801
08802 namespace Catch { namespace clara {
08803     namespace detail {
08804
08805         // Traits for extracting arg and return type of lambdas (for single argument lambdas)
08806         template<typename L>
08807         struct UnaryLambdaTraits : UnaryLambdaTraits<decltype( &L::operator() )> {};
08808
08809         template<typename ClassT, typename ReturnT, typename... Args>
08810         struct UnaryLambdaTraits<ReturnT( ClassT::* )( Args... ) const> {
08811             static const bool isValid = false;
08812         };
08813
08814         template<typename ClassT, typename ReturnT, typename ArgT>
08815         struct UnaryLambdaTraits<ReturnT( ClassT::* )( ArgT ) const> {
08816             static const bool isValid = true;
08817             using ArgType = typename std::remove_const<typename
std::remove_reference<ArgT>::type>::type;
08818             using ReturnT = ReturnT;
08819         };
08820
08821         class TokenStream;
08822
08823         // Transport for raw args (copied from main args, or supplied via init list for testing)
08824         class Args {
08825             friend TokenStream;
08826             std::string m_exeName;
08827             std::vector<std::string> m_args;
08828
08829         public:
08830             Args( int argc, char const* const* argv )
08831                 : m_exeName(argv[0]),
08832                   m_args(argv + 1, argv + argc) {}
08833
08834             Args( std::initializer_list<std::string> args )
08835                 : m_exeName( *args.begin() ),
08836                   m_args( args.begin()+1, args.end() )
08837             {}
08838
08839             auto exeName() const -> std::string {
08840                 return m_exeName;
08841             }
08842         };
08843
08844         // Wraps a token coming from a token stream. These may not directly correspond to strings
08845         as a single string
08846         // may encode an option + its argument if the : or = form is used
08847         enum class TokenType {
08848             Option, Argument
08849         };
08850         struct Token {
08851             TokenType type;
08852             std::string token;
08853         };
08854
08855         inline auto isOptPrefix( char c ) -> bool {
08856             return c == '-' ||
08857 #ifndef CATCH_PLATFORM_WINDOWS
08858                 || c == '/'
08859 #endif
08860         }
08861     }
08862 }

```



```

08859             ;
08860         }
08861
08862         // Abstracts iterators into args as a stream of tokens, with option arguments uniformly
08863         handled
08864         class TokenStream {
08865             using Iterator = std::vector<std::string>::const_iterator;
08866             Iterator it;
08867             Iterator itEnd;
08868             std::vector<Token> m_tokenBuffer;
08869
08870             void loadBuffer() {
08871                 m_tokenBuffer.resize( 0 );
08872
08873                 // Skip any empty strings
08874                 while( it != itEnd && it->empty() )
08875                     ++it;
08876
08877                 if( it != itEnd ) {
08878                     auto const &next = *it;
08879                     if( isOptPrefix( next[0] ) ) {
08880                         auto delimiterPos = next.find_first_of( " :=" );
08881                         if( delimiterPos != std::string::npos ) {
08882                             m_tokenBuffer.push_back( { TokenType::Option, next.substr( 0,
08883                                     delimiterPos ) } );
08884                             m_tokenBuffer.push_back( { TokenType::Argument, next.substr(
08885                                     delimiterPos + 1 ) } );
08886                         } else {
08887                             if( next[1] != '-' && next.size() > 2 ) {
08888                                 std::string opt = "- ";
08889                                 for( size_t i = 1; i < next.size(); ++i ) {
08890                                     opt[i] = next[i];
08891                                     m_tokenBuffer.push_back( { TokenType::Option, opt } );
08892                                 }
08893                             } else {
08894                                 m_tokenBuffer.push_back( { TokenType::Option, next } );
08895                             }
08896                         } else {
08897                             m_tokenBuffer.push_back( { TokenType::Argument, next } );
08898                         }
08899                     }
08900                 }
08901             public:
08902             explicit TokenStream( Args const &args ) : TokenStream( args.m_args.begin(),
08903                     args.m_args.end() ) {}
08904
08905             TokenStream( Iterator it, Iterator itEnd ) : it( it ), itEnd( itEnd ) {
08906                 loadBuffer();
08907             }
08908
08909             explicit operator bool() const {
08910                 return !m_tokenBuffer.empty() || it != itEnd;
08911             }
08912
08913             auto count() const -> size_t { return m_tokenBuffer.size() + (itEnd - it); }
08914
08915             auto operator*() const -> Token {
08916                 assert( !m_tokenBuffer.empty() );
08917                 return m_tokenBuffer.front();
08918             }
08919
08920             auto operator->() const -> Token const * {
08921                 assert( !m_tokenBuffer.empty() );
08922                 return &m_tokenBuffer.front();
08923             }
08924
08925             auto operator++() -> TokenStream & {
08926                 if( m_tokenBuffer.size() >= 2 ) {
08927                     m_tokenBuffer.erase( m_tokenBuffer.begin() );
08928                 } else {
08929                     if( it != itEnd )
08930                         ++it;
08931                     loadBuffer();
08932                 }
08933                 return *this;
08934             }
08935
08936             };
08937
08938             class ResultBase {
08939             public:
08940                 enum Type {
08941                     Ok, LogicError, RuntimeError
08942                 };
08943
08944             protected:

```

```

08942         ResultBase( Type type ) : m_type( type ) {}
08943         virtual ~ResultBase() = default;
08944
08945         virtual void enforceOk() const = 0;
08946
08947         Type m_type;
08948     };
08949
08950     template<typename T>
08951     class ResultValueBase : public ResultBase {
08952     public:
08953         auto value() const -> T const & {
08954             enforceOk();
08955             return m_value;
08956         }
08957
08958     protected:
08959         ResultValueBase( Type type ) : ResultBase( type ) {}
08960
08961         ResultValueBase( ResultValueBase const &other ) : ResultBase( other ) {
08962             if( m_type == ResultBase::Ok )
08963                 new( &m_value ) T( other.m_value );
08964         }
08965
08966         ResultValueBase( Type, T const &value ) : ResultBase( Ok ) {
08967             new( &m_value ) T( value );
08968         }
08969
08970         auto operator=( ResultValueBase const &other ) -> ResultValueBase & {
08971             if( m_type == ResultBase::Ok )
08972                 m_value.~T();
08973             ResultBase::operator=(other);
08974             if( m_type == ResultBase::Ok )
08975                 new( &m_value ) T( other.m_value );
08976             return *this;
08977         }
08978
08979         ~ResultValueBase() override {
08980             if( m_type == Ok )
08981                 m_value.~T();
08982         }
08983
08984         union {
08985             T m_value;
08986         };
08987     };
08988
08989     template<>
08990     class ResultValueBase<void> : public ResultBase {
08991     protected:
08992         using ResultBase::ResultBase;
08993     };
08994
08995     template<typename T = void>
08996     class BasicResult : public ResultValueBase<T> {
08997     public:
08998         template<typename U>
08999         explicit BasicResult( BasicResult<U> const &other )
09000             : ResultValueBase<T>( other.type() ),
09001               m_errorMessage( other.errorMessage() )
09002         {
09003             assert( type() != ResultBase::Ok );
09004         }
09005
09006         template<typename U>
09007         static auto ok( U const &value ) -> BasicResult { return { ResultBase::Ok, value }; }
09008         static auto ok() -> BasicResult { return { ResultBase::Ok }; }
09009         static auto logicError( std::string const &message ) -> BasicResult { return {
09010             ResultBase::LogicError, message }; }
09011         static auto runtimeError( std::string const &message ) -> BasicResult { return {
09012             ResultBase::RuntimeError, message }; }
09013
09014         explicit operator bool() const { return m_type == ResultBase::Ok; }
09015         auto type() const -> ResultBase::Type { return m_type; }
09016         auto errorMessage() const -> std::string { return m_errorMessage; }
09017
09018     protected:
09019         void enforceOk() const override {
09020             // Errors shouldn't reach this point, but if they do
09021             // the actual error message will be in m_errorMessage
09022             assert( m_type != ResultBase::LogicError );
09023             assert( m_type != ResultBase::RuntimeError );
09024             if( m_type != ResultBase::Ok )
09025                 std::abort();
09026         }

```

```

09027         std::string m_errorMessage; // Only populated if resultType is an error
09028
09029         BasicResult( ResultBase::Type type, std::string const &message )
09030             :   ResultValueBase<T>(type),
09031                 m_errorMessage(message)
09032         {
09033             assert( m_type != ResultBase::Ok );
09034         }
09035
09036         using ResultValueBase<T>::ResultValueBase;
09037         using ResultBase::m_type;
09038     };
09039
09040     enum class ParseResultType {
09041         Matched, NoMatch, ShortCircuitAll, ShortCircuitSame
09042     };
09043
09044     class ParseState {
09045     public:
09046
09047         ParseState( ParseResultType type, TokenStream const &remainingTokens )
09048             : m_type(type),
09049               m_remainingTokens( remainingTokens )
09050         {}
09051
09052         auto type() const -> ParseResultType { return m_type; }
09053         auto remainingTokens() const -> TokenStream { return m_remainingTokens; }
09054
09055     private:
09056         ParseResultType m_type;
09057         TokenStream m_remainingTokens;
09058     };
09059
09060     using Result = BasicResult<void>;
09061     using ParserResult = BasicResult<ParseResultType>;
09062     using InternalParserResult = BasicResult<ParseState>;
09063
09064     struct HelpColumns {
09065         std::string left;
09066         std::string right;
09067     };
09068
09069     template<typename T>
09070     inline auto convertInto( std::string const &source, T& target ) -> ParserResult {
09071         std::stringstream ss;
09072         ss << source;
09073         ss >> target;
09074         if( ss.fail() )
09075             return ParserResult::runtimeError( "Unable to convert '" + source + "' to
destination type" );
09076         else
09077             return ParserResult::ok( ParseResultType::Matched );
09078     }
09079     inline auto convertInto( std::string const &source, std::string& target ) -> ParserResult
{
09080         target = source;
09081         return ParserResult::ok( ParseResultType::Matched );
09082     }
09083     inline auto convertInto( std::string const &source, bool &target ) -> ParserResult {
09084         std::string srcLC = source;
09085         std::transform( srcLC.begin(), srcLC.end(), srcLC.begin(), []( unsigned char c ) {
return static_cast<char>( std::tolower(c) ); } );
09086         if (srcLC == "y" || srcLC == "1" || srcLC == "true" || srcLC == "yes" || srcLC ==
"on")
09087             target = true;
09088         else if (srcLC == "n" || srcLC == "0" || srcLC == "false" || srcLC == "no" || srcLC ==
"off")
09089             target = false;
09090         else
09091             return ParserResult::runtimeError( "Expected a boolean value but did not
recognise: '" + source + "'" );
09092         return ParserResult::ok( ParseResultType::Matched );
09093     }
09094 #ifndef CLARA_CONFIG_OPTIONAL_TYPE
09095     template<typename T>
09096     inline auto convertInto( std::string const &source, CLARA_CONFIG_OPTIONAL_TYPE<T>& target
) -> ParserResult {
09097         T temp;
09098         auto result = convertInto( source, temp );
09099         if( result )
09100             target = std::move(temp);
09101         return result;
09102     }
09103 #endif // CLARA_CONFIG_OPTIONAL_TYPE
09104
09105     struct NonCopyable {
09106         NonCopyable() = default;

```

```

09107         NonCopyable( NonCopyable const & ) = delete;
09108         NonCopyable( NonCopyable && ) = delete;
09109         NonCopyable &operator=( NonCopyable const & ) = delete;
09110         NonCopyable &operator=( NonCopyable && ) = delete;
09111     };
09112
09113     struct BoundRef : NonCopyable {
09114         virtual ~BoundRef() = default;
09115         virtual auto isContainer() const -> bool { return false; }
09116         virtual auto isFlag() const -> bool { return false; }
09117     };
09118     struct BoundValueRefBase : BoundRef {
09119         virtual auto setValue( std::string const &arg ) -> ParserResult = 0;
09120     };
09121     struct BoundFlagRefBase : BoundRef {
09122         virtual auto setFlag( bool flag ) -> ParserResult = 0;
09123         virtual auto isFlag() const -> bool { return true; }
09124     };
09125
09126     template<typename T>
09127     struct BoundValueRef : BoundValueRefBase {
09128         T &m_ref;
09129
09130         explicit BoundValueRef( T &ref ) : m_ref( ref ) {}
09131
09132         auto setValue( std::string const &arg ) -> ParserResult override {
09133             return convertInto( arg, m_ref );
09134         }
09135     };
09136
09137     template<typename T>
09138     struct BoundValueRef<std::vector<T> > : BoundValueRefBase {
09139         std::vector<T> &m_ref;
09140
09141         explicit BoundValueRef( std::vector<T> &ref ) : m_ref( ref ) {}
09142
09143         auto isContainer() const -> bool override { return true; }
09144
09145         auto setValue( std::string const &arg ) -> ParserResult override {
09146             T temp;
09147             auto result = convertInto( arg, temp );
09148             if( result )
09149                 m_ref.push_back( temp );
09150             return result;
09151         }
09152     };
09153
09154     struct BoundFlagRef : BoundFlagRefBase {
09155         bool &m_ref;
09156
09157         explicit BoundFlagRef( bool &ref ) : m_ref( ref ) {}
09158
09159         auto setFlag( bool flag ) -> ParserResult override {
09160             m_ref = flag;
09161             return ParserResult::ok( ParseResultType::Matched );
09162         }
09163     };
09164
09165     template<typename ReturnType>
09166     struct LambdaInvoker {
09167         static_assert( std::is_same<ReturnType, ParserResult>::value, "Lambda must return void
or clara::ParserResult" );
09168
09169         template<typename L, typename ArgType>
09170         static auto invoke( L const &lambda, ArgType const &arg ) -> ParserResult {
09171             return lambda( arg );
09172         }
09173     };
09174
09175     template<>
09176     struct LambdaInvoker<void> {
09177         template<typename L, typename ArgType>
09178         static auto invoke( L const &lambda, ArgType const &arg ) -> ParserResult {
09179             lambda( arg );
09180             return ParserResult::ok( ParseResultType::Matched );
09181         }
09182     };
09183
09184     template<typename ArgType, typename L>
09185     inline auto invokeLambda( L const &lambda, std::string const &arg ) -> ParserResult {
09186         ArgType temp{};
09187         auto result = convertInto( arg, temp );
09188         return !result
09189             ? result
09190             : LambdaInvoker<typename UnaryLambdaTraits<L>::ReturnType>::invoke( lambda,
temp );
09191     }

```

```

09192
09193     template<typename L>
09194     struct BoundLambda : BoundValueRefBase {
09195         L m_lambda;
09196
09197         static_assert( UnaryLambdaTraits<L>::isValid, "Supplied lambda must take exactly one
argument" );
09198         explicit BoundLambda( L const &lambda ) : m_lambda( lambda ) {}
09199
09200         auto setValue( std::string const &arg ) -> ParserResult override {
09201             return invokeLambda<typename UnaryLambdaTraits<L>::ArgType>( m_lambda, arg );
09202         }
09203     };
09204
09205     template<typename L>
09206     struct BoundFlagLambda : BoundFlagRefBase {
09207         L m_lambda;
09208
09209         static_assert( UnaryLambdaTraits<L>::isValid, "Supplied lambda must take exactly one
argument" );
09210         static_assert( std::is_same<typename UnaryLambdaTraits<L>::ArgType, bool>::value,
"flags must be boolean" );
09211
09212         explicit BoundFlagLambda( L const &lambda ) : m_lambda( lambda ) {}
09213
09214         auto setFlag( bool flag ) -> ParserResult override {
09215             return LambdaInvoker<typename UnaryLambdaTraits<L>::ReturnType>::invoke( m_lambda,
flag );
09216         }
09217     };
09218
09219     enum class Optionality { Optional, Required };
09220
09221     struct Parser;
09222
09223     class ParserBase {
09224     public:
09225         virtual ~ParserBase() = default;
09226         virtual auto validate() const -> Result { return Result::ok(); }
09227         virtual auto parse( std::string const& exeName, TokenStream const &tokens) const ->
InternalParseResult = 0;
09228         virtual auto cardinality() const -> size_t { return 1; }
09229
09230         auto parse( Args const &args ) const -> InternalParseResult {
09231             return parse( args.exeName(), TokenStream( args ) );
09232         }
09233     };
09234
09235     template<typename DerivedT>
09236     class ComposableParserImpl : public ParserBase {
09237     public:
09238         template<typename T>
09239         auto operator|( T const &other ) const -> Parser;
09240
09241         template<typename T>
09242         auto operator+( T const &other ) const -> Parser;
09243     };
09244
09245     // Common code and state for Args and Opts
09246     template<typename DerivedT>
09247     class ParserRefImpl : public ComposableParserImpl<DerivedT> {
09248     protected:
09249         Optionality m_optionality = Optionality::Optional;
09250         std::shared_ptr<BoundRef> m_ref;
09251         std::string m_hint;
09252         std::string m_description;
09253
09254         explicit ParserRefImpl( std::shared_ptr<BoundRef> const &ref ) : m_ref( ref ) {}
09255
09256     public:
09257         template<typename T>
09258         ParserRefImpl( T &ref, std::string const &hint )
09259             : m_ref( std::make_shared<BoundValueRef<T>( ref ) ),
09260               m_hint( hint )
09261         {}
09262
09263         template<typename LambdaT>
09264         ParserRefImpl( LambdaT const &ref, std::string const &hint )
09265             : m_ref( std::make_shared<BoundLambda<LambdaT>( ref ) ),
09266               m_hint( hint )
09267         {}
09268
09269         auto operator()( std::string const &description ) -> DerivedT & {
09270             m_description = description;
09271             return static_cast<DerivedT &>( *this );
09272         }
09273     }

```

```

09274         auto optional() -> DerivedT & {
09275             m_optionality = Optionality::Optional;
09276             return static_cast<DerivedT &>( *this );
09277         };
09278
09279         auto required() -> DerivedT & {
09280             m_optionality = Optionality::Required;
09281             return static_cast<DerivedT &>( *this );
09282         };
09283
09284         auto isOptional() const -> bool {
09285             return m_optionality == Optionality::Optional;
09286         }
09287
09288         auto cardinality() const -> size_t override {
09289             if( m_ref->isContainer() )
09290                 return 0;
09291             else
09292                 return 1;
09293         }
09294
09295         auto hint() const -> std::string { return m_hint; }
09296     };
09297
09298     class ExeName : public ComposableParserImpl<ExeName> {
09299     public:
09300         std::shared_ptr<std::string> m_name;
09301         std::shared_ptr<BoundValueRefBase> m_ref;
09302
09303         template<typename LambdaT>
09304         static auto makeRef( LambdaT const& lambda ) -> std::shared_ptr<BoundValueRefBase> {
09305             return std::make_shared<BoundLambda<LambdaT>>( lambda );
09306         }
09307
09308     public:
09309         ExeName() : m_name( std::make_shared<std::string>( "<executable>" ) ) {}
09310
09311         explicit ExeName( std::string& ref ) : ExeName() {
09312             m_ref = std::make_shared<BoundValueRef<std::string>>( ref );
09313         }
09314
09315         template<typename LambdaT>
09316         explicit ExeName( LambdaT const& lambda ) : ExeName() {
09317             m_ref = std::make_shared<BoundLambda<LambdaT>>( lambda );
09318         }
09319
09320         // The exe name is not parsed out of the normal tokens, but is handled specially
09321         auto parse( std::string const&, TokenStream const& tokens ) const ->
09322             InternalParseResult override {
09323             return InternalParseResult::ok( ParseState( ParseResultType::NoMatch, tokens ) );
09324         }
09325
09326         auto name() const -> std::string { return *m_name; }
09327         auto set( std::string const& newName ) -> ParserResult {
09328             auto lastSlash = newName.find_last_of( "\\/" );
09329             auto filename = ( lastSlash == std::string::npos
09330                             ? newName
09331                             : newName.substr( lastSlash+1 ) );
09332
09333             *m_name = filename;
09334             if( m_ref )
09335                 return m_ref->setValue( filename );
09336             else
09337                 return ParserResult::ok( ParseResultType::Matched );
09338         }
09339     };
09340
09341     class Arg : public ParserRefImpl<Arg> {
09342     public:
09343         using ParserRefImpl::ParserRefImpl;
09344
09345         auto parse( std::string const&, TokenStream const& tokens ) const ->
09346             InternalParseResult override {
09347             auto validationResult = validate();
09348             if( !validationResult )
09349                 return InternalParseResult( validationResult );
09350
09351             auto remainingTokens = tokens;
09352             auto const& token = *remainingTokens;
09353             if( token.type != TokenType::Argument )
09354                 return InternalParseResult::ok( ParseState( ParseResultType::NoMatch,
09355 remainingTokens ) );
09356
09357             assert( !m_ref->isFlag() );
09358             auto valueRef = static_cast<detail::BoundValueRefBase*>( m_ref.get() );
09359             auto result = valueRef->setValue( remainingTokens->token );

```

```

09358             if( !result )
09359                 return InternalParseResult( result );
09360             else
09361                 return InternalParseResult::ok( ParseState( ParseResultType::Matched,
++remainingTokens ) );
09362         }
09363     };
09364
09365     inline auto normaliseOpt( std::string const &optName ) -> std::string {
09366 #ifdef CATCH_PLATFORM_WINDOWS
09367         if( optName[0] == '/' )
09368             return "-" + optName.substr( 1 );
09369         else
09370             return optName;
09371 #endif
09372     }
09373
09374     class Opt : public ParserRefImpl<Opt> {
09375     protected:
09376         std::vector<std::string> m_optNames;
09377
09378     public:
09379         template<typename LambdaT>
09380         explicit Opt( LambdaT const &ref ) : ParserRefImpl(
std::make_shared<BoundFlagLambda<LambdaT>>( ref ) ) {}
09381
09382         explicit Opt( bool &ref ) : ParserRefImpl( std::make_shared<BoundFlagRef>( ref ) ) {}
09383
09384         template<typename LambdaT>
09385         Opt( LambdaT const &ref, std::string const &hint ) : ParserRefImpl( ref, hint ) {}
09386
09387         template<typename T>
09388         Opt( T &ref, std::string const &hint ) : ParserRefImpl( ref, hint ) {}
09389
09390         auto operator[]( std::string const &optName ) -> Opt & {
09391             m_optNames.push_back( optName );
09392             return *this;
09393         }
09394
09395         auto getHelpColumns() const -> std::vector<HelpColumns> {
09396             std::ostringstream oss;
09397             bool first = true;
09398             for( auto const &opt : m_optNames ) {
09399                 if( first )
09400                     first = false;
09401                 else
09402                     oss << ", ";
09403                 oss << opt;
09404             }
09405             if( !m_hint.empty() )
09406                 oss << " <" << m_hint << ">";
09407             return { { oss.str(), m_description } };
09408         }
09409
09410         auto isMatch( std::string const &optToken ) const -> bool {
09411             auto normalisedToken = normaliseOpt( optToken );
09412             for( auto const &name : m_optNames ) {
09413                 if( normaliseOpt( name ) == normalisedToken )
09414                     return true;
09415             }
09416             return false;
09417         }
09418
09419         using ParserBase::parse;
09420
09421         auto parse( std::string const&, TokenStream const &tokens ) const ->
InternalParseResult override {
09422             auto validationResult = validate();
09423             if( !validationResult )
09424                 return InternalParseResult( validationResult );
09425
09426             auto remainingTokens = tokens;
09427             if( remainingTokens && remainingTokens->type == TokenType::Option ) {
09428                 auto const &token = *remainingTokens;
09429                 if( isMatch( token.token ) ) {
09430                     if( m_ref->isFlag() ) {
09431                         auto flagRef = static_cast<detail::BoundFlagRefBase*>( m_ref.get() );
09432                         auto result = flagRef->setFlag( true );
09433                         if( !result )
09434                             return InternalParseResult( result );
09435                         if( result.value() == ParseResultType::ShortCircuitAll )
09436                             return InternalParseResult::ok( ParseState( result.value(),
remainingTokens ) );
09437                     } else {
09438                         auto valueRef = static_cast<detail::BoundValueRefBase*>( m_ref.get() );
09439

```

```

09440                                     if( !remainingTokens )
09441                                     return InternalParseResult::runtimeError( "Expected argument
following " + token.token );
09442                                     auto const &argToken = *remainingTokens;
09443                                     if( argToken.type != TokenType::Argument )
09444                                     return InternalParseResult::runtimeError( "Expected argument
following " + 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(),
remainingTokens ) );
09450                                     }
09451                                     return InternalParseResult::ok( ParseState( ParseResultType::Matched,
++remainingTokens ) );
09452                                     }
09453                                     }
09454                                     return InternalParseResult::ok( ParseState( ParseResultType::NoMatch,
remainingTokens ) );
09455                                     }
09456
09457                                     auto validate() const -> Result override {
09458                                     if( m_optNames.empty() )
09459                                     return Result::logicError( "No options supplied to Opt" );
09460                                     for( auto const &name : m_optNames ) {
09461                                     if( name.empty() )
09462                                     return Result::logicError( "Option name cannot be empty" );
09463                                     #ifdef CATCH_PLATFORM_WINDOWS
09464                                     if( name[0] != '-' && name[0] != '/' )
09465                                     return Result::logicError( "Option name must begin with '-' or '/'" );
09466                                     #else
09467                                     if( name[0] != '-' )
09468                                     return Result::logicError( "Option name must begin with '-'" );
09469                                     #endif
09470                                     }
09471                                     return ParserRefImpl::validate();
09472                                     }
09473                                     };
09474
09475                                     struct Help : Opt {
09476                                     Help( bool &showHelpFlag )
09477                                     : Opt([&]( bool flag ) {
09478                                     showHelpFlag = flag;
09479                                     return ParserResult::ok( ParseResultType::ShortCircuitAll );
09480                                     })
09481                                     {
09482                                     static_cast<Opt &>( *this )
09483                                     ("display usage information")
09484                                     ["-?"]["-h"]["--help"]
09485                                     .optional();
09486                                     }
09487                                     };
09488
09489                                     struct Parser : ParserBase {
09490
09491                                     mutable ExeName m_exeName;
09492                                     std::vector<Opt> m_options;
09493                                     std::vector<Arg> m_args;
09494
09495                                     auto operator|=( ExeName const &exeName ) -> Parser & {
09496                                     m_exeName = exeName;
09497                                     return *this;
09498                                     }
09499
09500                                     auto operator|=( Arg const &arg ) -> Parser & {
09501                                     m_args.push_back(arg);
09502                                     return *this;
09503                                     }
09504
09505                                     auto operator|=( Opt const &opt ) -> Parser & {
09506                                     m_options.push_back(opt);
09507                                     return *this;
09508                                     }
09509
09510                                     auto operator|=( Parser const &other ) -> Parser & {
09511                                     m_options.insert(m_options.end(), other.m_options.begin(), other.m_options.end());
09512                                     m_args.insert(m_args.end(), other.m_args.begin(), other.m_args.end());
09513                                     return *this;
09514                                     }
09515
09516                                     template<typename T>
09517                                     auto operator|( T const &other ) const -> Parser {
09518                                     return Parser( *this ) |= other;
09519                                     }
09520
09521                                     // Forward deprecated interface with '+' instead of '|'

```



```

09522     template<typename T>
09523     auto operator+=( T const &other ) -> Parser & { return operator|=( other ); }
09524     template<typename T>
09525     auto operator+( T const &other ) const -> Parser { return operator|( other ); }
09526
09527     auto getHelpColumns() const -> std::vector<HelpColumns> {
09528         std::vector<HelpColumns> cols;
09529         for (auto const &o : m_options) {
09530             auto childCols = o.getHelpColumns();
09531             cols.insert( cols.end(), childCols.begin(), childCols.end() );
09532         }
09533         return cols;
09534     }
09535
09536     void writeToStream( std::ostream &os ) const {
09537         if ( !m_exeName.name().empty() ) {
09538             os << "usage:\n" << " " << m_exeName.name() << " ";
09539             bool required = true, first = true;
09540             for( auto const &arg : m_args ) {
09541                 if (first)
09542                     first = false;
09543                 else
09544                     os << " ";
09545                 if( arg.isOptional() && required ) {
09546                     os << "[";
09547                     required = false;
09548                 }
09549                 os << "<" << arg.hint() << ">";
09550                 if( arg.cardinality() == 0 )
09551                     os << " ... ";
09552             }
09553             if( !required )
09554                 os << "]";
09555             if( !m_options.empty() )
09556                 os << " options";
09557             os << "\n\nwhere options are:" << std::endl;
09558         }
09559
09560         auto rows = getHelpColumns();
09561         size_t consoleWidth = CATCH_CLARA_CONFIG_CONSOLE_WIDTH;
09562         size_t optWidth = 0;
09563         for( auto const &cols : rows )
09564             optWidth = (std::max)(optWidth, cols.left.size() + 2);
09565
09566         optWidth = (std::min)(optWidth, consoleWidth/2);
09567
09568         for( auto const &cols : rows ) {
09569             auto row =
09570                 TextFlow::Column( cols.left ).width( optWidth ).indent( 2 ) +
09571                 TextFlow::Spacer(4) +
09572                 TextFlow::Column( cols.right ).width( consoleWidth - 7 - optWidth );
09573             os << row << std::endl;
09574         }
09575     }
09576
09577     friend auto operator<<( std::ostream &os, Parser const &parser ) -> std::ostream& {
09578         parser.writeToStream( os );
09579         return os;
09580     }
09581
09582     auto validate() const -> Result override {
09583         for( auto const &opt : m_options ) {
09584             auto result = opt.validate();
09585             if( !result )
09586                 return result;
09587         }
09588         for( auto const &arg : m_args ) {
09589             auto result = arg.validate();
09590             if( !result )
09591                 return result;
09592         }
09593         return Result::ok();
09594     }
09595
09596     using ParserBase::parse;
09597
09598     auto parse( std::string const& exeName, TokenStream const &tokens ) const ->
InternalParseResult override {
09599
09600         struct ParserInfo {
09601             ParserBase const* parser = nullptr;
09602             size_t count = 0;
09603         };
09604         const size_t totalParsers = m_options.size() + m_args.size();
09605         assert( totalParsers < 512 );
09606         // ParserInfo parseInfos[totalParsers]; // <-- this is what we really want to do
09607         ParserInfo parseInfos[512];

```

```

09608
09609
09610         {
09611             size_t i = 0;
09612             for (auto const &opt : m_options) parseInfos[i++].parser = &opt;
09613             for (auto const &arg : m_args) parseInfos[i++].parser = &arg;
09614         }
09615         m_exeName.set( exeName );
09616
09617         auto result = InternalParseResult::ok( ParseState( ParseResultType::NoMatch,
tokens ) );
09618         while( result.value().remainingTokens() ) {
09619             bool tokenParsed = false;
09620
09621             for( size_t i = 0; i < totalParsers; ++i ) {
09622                 auto& parseInfo = parseInfos[i];
09623                 if( parseInfo.parser->cardinality() == 0 || parseInfo.count <
parseInfo.parser->cardinality() ) {
09624                     result = parseInfo.parser->parse( exeName,
result.value().remainingTokens() );
09625                     if (!result)
09626                         return result;
09627                     if (result.value().type() != ParseResultType::NoMatch) {
09628                         tokenParsed = true;
09629                         ++parseInfo.count;
09630                         break;
09631                     }
09632                 }
09633             }
09634
09635             if( result.value().type() == ParseResultType::ShortCircuitAll )
09636                 return result;
09637             if( !tokenParsed )
09638                 return InternalParseResult::runtimeError( "Unrecognised token: " +
result.value().remainingTokens()->token );
09639         }
09640         // !TBD Check missing required options
09641         return result;
09642     }
09643 };
09644
09645     template<typename DerivedT>
09646     template<typename T>
09647     auto ComposableParserImpl<DerivedT>::operator|( T const &other ) const -> Parser {
09648         return Parser() | static_cast<DerivedT const &>( *this ) | other;
09649     }
09650 } // namespace detail
09651
09652 // A Combined parser
09653 using detail::Parser;
09654
09655 // A parser for options
09656 using detail::Opt;
09657
09658 // A parser for arguments
09659 using detail::Arg;
09660
09661 // Wrapper for argc, argv from main()
09662 using detail::Args;
09663
09664 // Specifies the name of the executable
09665 using detail::ExeName;
09666
09667 // Convenience wrapper for option parser that specifies the help option
09668 using detail::Help;
09669
09670 // enum of result types from a parse
09671 using detail::ParseResultType;
09672
09673 // Result type for parser operation
09674 using detail::ParserResult;
09675
09676 } // namespace Catch::clara
09677
09678 // end clara.hpp
09679 #ifdef __clang__
09680 #pragma clang diagnostic pop
09681 #endif
09682
09683 // Restore Clara's value for console width, if present
09684 #ifdef CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
09685 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
09686 #undef CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
09687 #endif
09688
09689 // end catch_clara.h
09690 namespace Catch {

```

```

09691
09692     clara::Parser makeCommandLineParser( ConfigData& config );
09693
09694 } // end namespace Catch
09695
09696 // end catch_commandline.h
09697 #include <fstream>
09698 #include <ctime>
09699
09700 namespace Catch {
09701
09702     clara::Parser makeCommandLineParser( ConfigData& config ) {
09703
09704         using namespace clara;
09705
09706         auto const setWarning = [&]( std::string const& warning ) {
09707             auto warningSet = [&]() {
09708                 if( warning == "NoAssertions" )
09709                     return WarnAbout::NoAssertions;
09710
09711                 if ( warning == "NoTests" )
09712                     return WarnAbout::NoTests;
09713
09714                 return WarnAbout::Nothing;
09715             }();
09716
09717             if (warningSet == WarnAbout::Nothing)
09718                 return ParserResult::runtimeError( "Unrecognised warning: '" + warning + "'" );
09719             config.warnings = static_cast<WarnAbout::What>( config.warnings | warningSet );
09720             return ParserResult::ok( ParseResultType::Matched );
09721         };
09722         auto const loadTestNamesFromFile = [&]( std::string const& filename ) {
09723             std::ifstream f( filename.c_str() );
09724             if( !f.is_open() )
09725                 return ParserResult::runtimeError( "Unable to load input file: '" + filename + "'" );
09726
09727             std::string line;
09728             while( std::getline( f, line ) ) {
09729                 line = trim(line);
09730                 if( !line.empty() && !startsWith( line, '#' ) ) {
09731                     if( !startsWith( line, '"' ) )
09732                         line = '"' + line + '"';
09733                     config.testsOrTags.push_back( line );
09734                     config.testsOrTags.emplace_back( "," );
09735                 }
09736             }
09737             //Remove comma in the end
09738             if(!config.testsOrTags.empty())
09739                 config.testsOrTags.erase( config.testsOrTags.end()-1 );
09740
09741             return ParserResult::ok( ParseResultType::Matched );
09742         };
09743         auto const setTestOrder = [&]( std::string const& order ) {
09744             if( startsWith( "declared", order ) )
09745                 config.runOrder = RunTests::InDeclarationOrder;
09746             else if( startsWith( "lexical", order ) )
09747                 config.runOrder = RunTests::InLexicographicalOrder;
09748             else if( startsWith( "random", order ) )
09749                 config.runOrder = RunTests::InRandomOrder;
09750             else
09751                 return clara::ParserResult::runtimeError( "Unrecognised ordering: '" + order + "'" );
09752             return ParserResult::ok( ParseResultType::Matched );
09753         };
09754         auto const setRngSeed = [&]( std::string const& seed ) {
09755             if( seed != "time" )
09756                 return clara::detail::convertInto( seed, config.rngSeed );
09757             config.rngSeed = static_cast<unsigned int>( std::time(nullptr) );
09758             return ParserResult::ok( ParseResultType::Matched );
09759         };
09760         auto const setColourUsage = [&]( std::string const& useColour ) {
09761             auto mode = toLower( useColour );
09762
09763             if( mode == "yes" )
09764                 config.useColour = UseColour::Yes;
09765             else if( mode == "no" )
09766                 config.useColour = UseColour::No;
09767             else if( mode == "auto" )
09768                 config.useColour = UseColour::Auto;
09769             else
09770                 return ParserResult::runtimeError( "colour mode must be one of: auto, yes or no. '" +
09771 useColour + "' not recognised" );
09772             return ParserResult::ok( ParseResultType::Matched );
09773         };
09774         auto const setWaitForKeypress = [&]( std::string const& keypress ) {
09775             auto keypressLc = toLower( keypress );
09776             if( keypressLc == "never" )
09777                 config.waitForKeypress = WaitForKeypress::Never;

```

```

09777         else if( keypressLc == "start" )
09778             config.waitForKeypress = WaitForKeypress::BeforeStart;
09779         else if( keypressLc == "exit" )
09780             config.waitForKeypress = WaitForKeypress::BeforeExit;
09781         else if( keypressLc == "both" )
09782             config.waitForKeypress = WaitForKeypress::BeforeStartAndExit;
09783         else
09784             return ParserResult::runtimeError( "keypress argument must be one of: never, start,
exit or both. '" + keypress + "' not recognised" );
09785         return ParserResult::ok( ParseResultType::Matched );
09786     };
09787     auto const setVerbosity = [&]( std::string const& verbosity ) {
09788         auto lcVerbosity = toLower( verbosity );
09789         if( lcVerbosity == "quiet" )
09790             config.verbosity = Verbosity::Quiet;
09791         else if( lcVerbosity == "normal" )
09792             config.verbosity = Verbosity::Normal;
09793         else if( lcVerbosity == "high" )
09794             config.verbosity = Verbosity::High;
09795         else
09796             return ParserResult::runtimeError( "Unrecognised verbosity, '" + verbosity + "'" );
09797         return ParserResult::ok( ParseResultType::Matched );
09798     };
09799     auto const setReporter = [&]( std::string const& reporter ) {
09800         IReporterRegistry::FactoryMap const& factories =
getRegistryHub().getReporterRegistry().getFactories();
09801
09802         auto lcReporter = toLower( reporter );
09803         auto result = factories.find( lcReporter );
09804
09805         if( factories.end() != result )
09806             config.reporterName = lcReporter;
09807         else
09808             return ParserResult::runtimeError( "Unrecognized reporter, '" + reporter + "'. Check
available with --list-reporters" );
09809         return ParserResult::ok( ParseResultType::Matched );
09810     };
09811
09812     auto cli
09813         = ExeName( config.processName )
09814         | Help( config.showHelp )
09815         | Opt( config.listTests )
09816         [ "-l" ][ "--list-tests" ]
09817             ( "list all/matching test cases" )
09818         | Opt( config.listTags )
09819         [ "-t" ][ "--list-tags" ]
09820             ( "list all/matching tags" )
09821         | Opt( config.showSuccessfulTests )
09822         [ "-s" ][ "--success" ]
09823             ( "include successful tests in output" )
09824         | Opt( config.shouldDebugBreak )
09825         [ "-b" ][ "--break" ]
09826             ( "break into debugger on failure" )
09827         | Opt( config.noThrow )
09828         [ "-e" ][ "--nothrow" ]
09829             ( "skip exception tests" )
09830         | Opt( config.showInvisibles )
09831         [ "-i" ][ "--invisibles" ]
09832             ( "show invisibles (tabs, newlines)" )
09833         | Opt( config.outputFilename, "filename" )
09834         [ "-o" ][ "--out" ]
09835             ( "output filename" )
09836         | Opt( setReporter, "name" )
09837         [ "-r" ][ "--reporter" ]
09838             ( "reporter to use (defaults to console)" )
09839         | Opt( config.name, "name" )
09840         [ "-n" ][ "--name" ]
09841             ( "suite name" )
09842         | Opt( [&]( bool ){ config.abortAfter = 1; } )
09843         [ "-a" ][ "--abort" ]
09844             ( "abort at first failure" )
09845         | Opt( [&]( int x ){ config.abortAfter = x; }, "no. failures" )
09846         [ "-x" ][ "--abortx" ]
09847             ( "abort after x failures" )
09848         | Opt( setWarning, "warning name" )
09849         [ "-w" ][ "--warn" ]
09850             ( "enable warnings" )
09851         | Opt( [&]( bool flag ) { config.showDurations = flag ? ShowDurations::Always :
ShowDurations::Never; }, "yes|no" )
09852         [ "-d" ][ "--durations" ]
09853             ( "show test durations" )
09854         | Opt( config.minDuration, "seconds" )
09855         [ "-D" ][ "--min-duration" ]
09856             ( "show test durations for tests taking at least the given number of
seconds" )
09857         | Opt( loadTestNamesFromFile, "filename" )
09858         [ "-f" ][ "--input-file" ]

```

```

09859         ( "load test names to run from a file" )
09860     | Opt( config.filenameAsTags )
09861     [ "--filenames-as-tags" ]
09862         ( "adds a tag for the filename" )
09863     | Opt( config.sectionsToRun, "section name" )
09864     [ "--section" ]
09865         ( "specify section to run" )
09866     | Opt( setVerbosity, "quiet|normal|high" )
09867     [ "--v" ][ "--verbosity" ]
09868         ( "set output verbosity" )
09869     | Opt( config.listTestNamesOnly )
09870     [ "--list-test-names-only" ]
09871         ( "list all/matching test cases names only" )
09872     | Opt( config.listReporters )
09873     [ "--list-reporters" ]
09874         ( "list all reporters" )
09875     | Opt( setTestOrder, "decl|lex|rand" )
09876     [ "--order" ]
09877         ( "test case order (defaults to decl)" )
09878     | Opt( setRngSeed, "'time'|number" )
09879     [ "--rng-seed" ]
09880         ( "set a specific seed for random numbers" )
09881     | Opt( setColourUsage, "yes|no" )
09882     [ "--use-colour" ]
09883         ( "should output be colourised" )
09884     | Opt( config.libIdentify )
09885     [ "--libidentify" ]
09886         ( "report name and version according to libidentify standard" )
09887     | Opt( setWaitForKeypress, "never|start|exit|both" )
09888     [ "--wait-for-keypress" ]
09889         ( "waits for a keypress before exiting" )
09890     | Opt( config.benchmarkSamples, "samples" )
09891     [ "--benchmark-samples" ]
09892         ( "number of samples to collect (default: 100)" )
09893     | Opt( config.benchmarkResamples, "resamples" )
09894     [ "--benchmark-resamples" ]
09895         ( "number of resamples for the bootstrap (default: 100000)" )
09896     | Opt( config.benchmarkConfidenceInterval, "confidence interval" )
09897     [ "--benchmark-confidence-interval" ]
09898         ( "confidence interval for the bootstrap (between 0 and 1, default: 0.95)" )
09899     | Opt( config.benchmarkNoAnalysis )
09900     [ "--benchmark-no-analysis" ]
09901         ( "perform only measurements; do not perform any analysis" )
09902     | Opt( config.benchmarkWarmupTime, "benchmarkWarmupTime" )
09903     [ "--benchmark-warmup-time" ]
09904         ( "amount of time in milliseconds spent on warming up each test (default:
09905 100)" )
09906     | Arg( config.testsOrTags, "test name|pattern|tags" )
09907         ( "which test or tests to use" );
09908     return cli;
09909 }
09910
09911 } // end namespace Catch
09912 // end catch_commandline.cpp
09913 // start catch_common.cpp
09914
09915 #include <cstring>
09916 #include <ostream>
09917
09918 namespace Catch {
09919
09920     bool SourceLineInfo::operator == ( SourceLineInfo const& other ) const noexcept {
09921         return line == other.line && ( file == other.file || std::strcmp(file, other.file) == 0 );
09922     }
09923     bool SourceLineInfo::operator < ( SourceLineInfo const& other ) const noexcept {
09924         // We can assume that the same file will usually have the same pointer.
09925         // Thus, if the pointers are the same, there is no point in calling the strcmp
09926         return line < other.line || ( line == other.line && file != other.file && (std::strcmp(file,
09927 other.file) < 0) );
09928     }
09929     std::ostream& operator << ( std::ostream& os, SourceLineInfo const& info ) {
09930 #ifndef __GNUG__
09931         os << info.file << '(' << info.line << ')';
09932     #else
09933         os << info.file << ':' << info.line;
09934     #endif
09935         return os;
09936     }
09937
09938     std::string StreamEndStop::operator+() const {
09939         return std::string();
09940     }
09941
09942     NonCopyable::NonCopyable() = default;
09943     NonCopyable::~NonCopyable() = default;

```

```

09944
09945 }
09946 // end catch_common.cpp
09947 // start catch_config.cpp
09948
09949 namespace Catch {
09950
09951     Config::Config( ConfigData const& data )
09952         :   m_data( data ),
09953             m_stream( openStream() )
09954     {
09955         // We need to trim filter specs to avoid trouble with superfluous
09956         // whitespace (esp. important for bdd macros, as those are manually
09957         // aligned with whitespace).
09958
09959         for (auto& elem : m_data.testsOrTags) {
09960             elem = trim(elem);
09961         }
09962         for (auto& elem : m_data.sectionsToRun) {
09963             elem = trim(elem);
09964         }
09965
09966         TestSpecParser parser(ITagAliasRegistry::get());
09967         if (!m_data.testsOrTags.empty()) {
09968             m_hasTestFilters = true;
09969             for (auto const& testOrTags : m_data.testsOrTags) {
09970                 parser.parse(testOrTags);
09971             }
09972         }
09973         m_testSpec = parser.testSpec();
09974     }
09975
09976     std::string const& Config::getFilename() const {
09977         return m_data.outputFilename ;
09978     }
09979
09980     bool Config::listTests() const { return m_data.listTests; }
09981     bool Config::listTestNamesOnly() const { return m_data.listTestNamesOnly; }
09982     bool Config::listTags() const { return m_data.listTags; }
09983     bool Config::listReporters() const { return m_data.listReporters; }
09984
09985     std::string Config::getProcessName() const { return m_data.processName; }
09986     std::string const& Config::getReporterName() const { return m_data.reporterName; }
09987
09988     std::vector<std::string> const& Config::getTestsOrTags() const { return m_data.testsOrTags; }
09989     std::vector<std::string> const& Config::getSectionsToRun() const { return m_data.sectionsToRun; }
09990
09991     TestSpec const& Config::testSpec() const { return m_testSpec; }
09992     bool Config::hasTestFilters() const { return m_hasTestFilters; }
09993
09994     bool Config::showHelp() const { return m_data.showHelp; }
09995
09996     // IConfig interface
09997     bool Config::allowThrows() const { return !m_data.noThrow; }
09998     std::ostream& Config::stream() const { return m_stream->stream(); }
09999     std::string Config::name() const { return m_data.name.empty() ?
m_data.processName : m_data.name; }
10000     bool Config::includeSuccessfulResults() const { return m_data.showSuccessfulTests; }
10001     bool Config::warnAboutMissingAssertions() const { return !(m_data.warnings &
WarnAbout::NoAssertions); }
10002     bool Config::warnAboutNoTests() const { return !(m_data.warnings &
WarnAbout::NoTests); }
10003     ShowDurations::OrNot Config::showDurations() const { return m_data.showDurations; }
10004     double Config::minDuration() const { return m_data.minDuration; }
10005     RunTests::InWhatOrder Config::runOrder() const { return m_data.runOrder; }
10006     unsigned int Config::rngSeed() const { return m_data.rngSeed; }
10007     UseColour::YesOrNo Config::useColour() const { return m_data.useColour; }
10008     bool Config::shouldDebugBreak() const { return m_data.shouldDebugBreak; }
10009     int Config::abortAfter() const { return m_data.abortAfter; }
10010     bool Config::showInvisibles() const { return m_data.showInvisibles; }
10011     Verbosity Config::verbosity() const { return m_data.verbosity; }
10012
10013     bool Config::benchmarkNoAnalysis() const { return m_data.benchmarkNoAnalysis; }
10014     int Config::benchmarkSamples() const { return m_data.benchmarkSamples; }
10015     double Config::benchmarkConfidenceInterval() const { return m_data.benchmarkConfidenceInterval; }
10016     unsigned int Config::benchmarkResamples() const { return m_data.benchmarkResamples; }
10017     std::chrono::milliseconds Config::benchmarkWarmupTime() const { return
std::chrono::milliseconds(m_data.benchmarkWarmupTime); }
10018
10019     IStream const* Config::openStream() {
10020         return Catch::makeStream(m_data.outputFilename);
10021     }
10022
10023 } // end namespace Catch

```

```

10024 // end catch_config.cpp
10025 // start catch_console_colour.cpp
10026
10027 #if defined(__clang__)
10028 #   pragma clang diagnostic push
10029 #   pragma clang diagnostic ignored "-Wexit-time-destructors"
10030 #endif
10031
10032 // start catch_errno_guard.h
10033
10034 namespace Catch {
10035
10036     class ErrnoGuard {
10037     public:
10038         ErrnoGuard();
10039         ~ErrnoGuard();
10040     private:
10041         int m_oldErrno;
10042     };
10043
10044 }
10045
10046 // end catch_errno_guard.h
10047 // start catch_windows_h_proxy.h
10048
10049
10050 #if defined(CATCH_PLATFORM_WINDOWS)
10051
10052 #if !defined(NOMINMAX) && !defined(CATCH_CONFIG_NO_NOMINMAX)
10053 #   define CATCH_DEFINED_NOMINMAX
10054 #   define NOMINMAX
10055 #endif
10056 #if !defined(WIN32_LEAN_AND_MEAN) && !defined(CATCH_CONFIG_NO_WIN32_LEAN_AND_MEAN)
10057 #   define CATCH_DEFINED_WIN32_LEAN_AND_MEAN
10058 #   define WIN32_LEAN_AND_MEAN
10059 #endif
10060
10061 #ifdef __AFXDLL
10062 #include <AfxWin.h>
10063 #else
10064 #include <windows.h>
10065 #endif
10066
10067 #ifdef CATCH_DEFINED_NOMINMAX
10068 #   undef NOMINMAX
10069 #endif
10070 #ifdef CATCH_DEFINED_WIN32_LEAN_AND_MEAN
10071 #   undef WIN32_LEAN_AND_MEAN
10072 #endif
10073
10074 #endif // defined(CATCH_PLATFORM_WINDOWS)
10075
10076 // end catch_windows_h_proxy.h
10077 #include <sstream>
10078
10079 namespace Catch {
10080     namespace {
10081
10082         struct IColourImpl {
10083             virtual ~IColourImpl() = default;
10084             virtual void use( Colour::Code _colourCode ) = 0;
10085         };
10086
10087         struct NoColourImpl : IColourImpl {
10088             void use( Colour::Code ) override {}
10089
10090             static IColourImpl* instance() {
10091                 static NoColourImpl s_instance;
10092                 return &s_instance;
10093             }
10094         };
10095
10096     } // anon namespace
10097 } // namespace Catch
10098
10099 #if !defined( CATCH_CONFIG_COLOUR_NONE ) && !defined( CATCH_CONFIG_COLOUR_WINDOWS ) && !defined(
CATCH_CONFIG_COLOUR_ANSI )
10100 #   ifdef CATCH_PLATFORM_WINDOWS
10101 #       define CATCH_CONFIG_COLOUR_WINDOWS
10102 #   else
10103 #       define CATCH_CONFIG_COLOUR_ANSI
10104 #   endif
10105 #endif
10106
10107 #if defined( CATCH_CONFIG_COLOUR_WINDOWS )
10108
10109 namespace Catch {

```

```

10110 namespace {
10111
10112     class Win32ColourImpl : public IColourImpl {
10113     public:
10114         Win32ColourImpl() : stdoutHandle( GetStdHandle(STD_OUTPUT_HANDLE) )
10115         {
10116             CONSOLE_SCREEN_BUFFER_INFO csbiInfo;
10117             GetConsoleScreenBufferInfo( stdoutHandle, &csbiInfo );
10118             originalForegroundAttributes = csbiInfo.wAttributes & ~( BACKGROUND_GREEN | BACKGROUND_RED
| BACKGROUND_BLUE | BACKGROUND_INTENSITY );
10119             originalBackgroundAttributes = csbiInfo.wAttributes & ~( FOREGROUND_GREEN | FOREGROUND_RED
| FOREGROUND_BLUE | FOREGROUND_INTENSITY );
10120         }
10121
10122         void use( Colour::Code _colourCode ) override {
10123             switch( _colourCode ) {
10124                 case Colour::None:         return setTextAttribute( originalForegroundAttributes );
10125                 case Colour::White:        return setTextAttribute( FOREGROUND_GREEN | FOREGROUND_RED |
FOREGROUND_BLUE );
10126                 case Colour::Red:          return setTextAttribute( FOREGROUND_RED );
10127                 case Colour::Green:        return setTextAttribute( FOREGROUND_GREEN );
10128                 case Colour::Blue:         return setTextAttribute( FOREGROUND_BLUE );
10129                 case Colour::Cyan:         return setTextAttribute( FOREGROUND_BLUE | FOREGROUND_GREEN );
10130                 case Colour::Yellow:       return setTextAttribute( FOREGROUND_RED | FOREGROUND_GREEN );
10131                 case Colour::Grey:         return setTextAttribute( 0 );
10132
10133                 case Colour::LightGrey:    return setTextAttribute( FOREGROUND_INTENSITY );
10134                 case Colour::BrightRed:    return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_RED );
10135                 case Colour::BrightGreen:  return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_GREEN );
10136                 case Colour::BrightWhite:  return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_GREEN | FOREGROUND_RED | FOREGROUND_BLUE );
10137                 case Colour::BrightYellow: return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_RED | FOREGROUND_GREEN );
10138
10139                 case Colour::Bright:      CATCH_INTERNAL_ERROR( "not a colour" );
10140
10141                 default:
10142                     CATCH_ERROR( "Unknown colour requested" );
10143             }
10144         }
10145
10146     private:
10147         void setTextAttribute( WORD _textAttribute ) {
10148             SetConsoleTextAttribute( stdoutHandle, _textAttribute | originalBackgroundAttributes );
10149         }
10150         HANDLE stdoutHandle;
10151         WORD originalForegroundAttributes;
10152         WORD originalBackgroundAttributes;
10153     };
10154
10155     IColourImpl* platformColourInstance() {
10156         static Win32ColourImpl s_instance;
10157
10158         IConfigPtr config = getCurrentContext().getConfig();
10159         UseColour::YesOrNo colourMode = config
? config->useColour()
: UseColour::Auto;
10160         if( colourMode == UseColour::Auto )
10161             colourMode = UseColour::Yes;
10162         return colourMode == UseColour::Yes
? &s_instance
: NoColourImpl::instance();
10163     }
10164 }
10165
10166 } // end anon namespace
10167 } // end namespace Catch
10168
10169 #elif defined( CATCH_CONFIG_COLOUR_ANSI )
10170
10171 #include <unistd.h>
10172
10173 namespace Catch {
10174     namespace {
10175         // use POSIX/ ANSI console terminal codes
10176         // Thanks to Adam Strzelecki for original contribution
10177         // (http://github.com/nanoant)
10178         // https://github.com/philsquared/Catch/pull/131
10179         class PosixColourImpl : public IColourImpl {
10180     public:
10181         void use( Colour::Code _colourCode ) override {
10182             switch( _colourCode ) {
10183                 case Colour::None:
10184                 case Colour::White:        return setColour( "[0m" );
10185                 case Colour::Red:          return setColour( "[0;31m" );

```



```

10190         case Colour::Green:      return setColour( "[0;32m" );
10191         case Colour::Blue:       return setColour( "[0;34m" );
10192         case Colour::Cyan:       return setColour( "[0;36m" );
10193         case Colour::Yellow:     return setColour( "[0;33m" );
10194         case Colour::Grey:       return setColour( "[1;30m" );
10195
10196         case Colour::LightGrey:   return setColour( "[0;37m" );
10197         case Colour::BrightRed:   return setColour( "[1;31m" );
10198         case Colour::BrightGreen: return setColour( "[1;32m" );
10199         case Colour::BrightWhite: return setColour( "[1;37m" );
10200         case Colour::BrightYellow: return setColour( "[1;33m" );
10201
10202         case Colour::Bright: CATCH_INTERNAL_ERROR( "not a colour" );
10203         default: CATCH_INTERNAL_ERROR( "Unknown colour requested" );
10204     }
10205 }
10206 static IColourImpl* instance() {
10207     static PosixColourImpl s_instance;
10208     return &s_instance;
10209 }
10210
10211 private:
10212     void setColour( const char* _escapeCode ) {
10213         getCurrentContext().getConfig()->stream()
10214             << '\033' << _escapeCode;
10215     }
10216 };
10217
10218     bool useColourOnPlatform() {
10219         return
10220 #if defined(CATCH_PLATFORM_MAC) || defined(CATCH_PLATFORM_IPHONE)
10221             !isDebuggerActive() &&
10222             #endif
10223             #if !(defined(__DJGPP__) && defined(__STRICT_ANSI__))
10224                 isatty(STDOUT_FILENO)
10225             #else
10226                 false
10227             #endif
10228         ;
10229     }
10230     IColourImpl* platformColourInstance() {
10231         ErrnoGuard guard;
10232         IConfigPtr config = getCurrentContext().getConfig();
10233         UseColour::YesOrNo colourMode = config
10234             ? config->useColour()
10235             : UseColour::Auto;
10236         if( colourMode == UseColour::Auto )
10237             colourMode = useColourOnPlatform()
10238                 ? UseColour::Yes
10239                 : UseColour::No;
10240         return colourMode == UseColour::Yes
10241             ? PosixColourImpl::instance()
10242             : NoColourImpl::instance();
10243     }
10244 } // end anon namespace
10245 } // end namespace Catch
10246
10247 #else // not Windows or ANSI //////////////////////////////////////
10248
10249 namespace Catch {
10250     static IColourImpl* platformColourInstance() { return NoColourImpl::instance(); }
10251 } // end namespace Catch
10252
10253 #endif // Windows/ ANSI/ None
10254
10255 namespace Catch {
10256     Colour::Colour( Code _colourCode ) { use( _colourCode ); }
10257     Colour::Colour( Colour&& other ) noexcept {
10258         m_moved = other.m_moved;
10259         other.m_moved = true;
10260     }
10261     Colour& Colour::operator=( Colour&& other ) noexcept {
10262         m_moved = other.m_moved;
10263         other.m_moved = true;
10264         return *this;
10265     }
10266     Colour::~~Colour() { if( !m_moved ) use( None ); }
10267
10268     void Colour::use( Code _colourCode ) {
10269         static IColourImpl* impl = platformColourInstance();
10270         // Strictly speaking, this cannot possibly happen.
10271         // However, under some conditions it does happen (see #1626),

```

```

10277         // and this change is small enough that we can let practicality
10278         // triumph over purity in this case.
10279         if (impl != nullptr) {
10280             impl->use( _colourCode );
10281         }
10282     }
10283
10284     std::ostream& operator << ( std::ostream& os, Colour const& ) {
10285         return os;
10286     }
10287
10288 } // end namespace Catch
10289
10290 #if defined(__clang__)
10291 #    pragma clang diagnostic pop
10292 #endif
10293
10294 // end catch_console_colour.cpp
10295 // start catch_context.cpp
10296
10297 namespace Catch {
10298
10299     class Context : public IMutableContext, NonCopyable {
10300     public: // IContext
10301         IResultCapture* getResultCapture() override {
10302             return m_resultCapture;
10303         }
10304         IRunner* getRunner() override {
10305             return m_runner;
10306         }
10307         IConfigPtr const& getConfig() const override {
10308             return m_config;
10309         }
10310         ~Context() override;
10311
10312     public: // IMutableContext
10313         void setResultCapture( IResultCapture* resultCapture ) override {
10314             m_resultCapture = resultCapture;
10315         }
10316         void setRunner( IRunner* runner ) override {
10317             m_runner = runner;
10318         }
10319         void setConfig( IConfigPtr const& config ) override {
10320             m_config = config;
10321         }
10322
10323         friend IMutableContext& getCurrentMutableContext();
10324
10325     private:
10326         IConfigPtr m_config;
10327         IRunner* m_runner = nullptr;
10328         IResultCapture* m_resultCapture = nullptr;
10329     };
10330
10331     IMutableContext* IMutableContext::currentContext = nullptr;
10332
10333     void IMutableContext::createContext() {
10334         currentContext = new Context();
10335     }
10336
10337     void cleanUpContext() {
10338         delete IMutableContext::currentContext;
10339         IMutableContext::currentContext = nullptr;
10340     }
10341
10342     IContext::~IContext() = default;
10343     IMutableContext::~IMutableContext() = default;
10344     Context::~Context() = default;
10345
10346     SimplePcg32& rng() {
10347         static SimplePcg32 s_rng;
10348         return s_rng;
10349     }
10350
10351 }
10352
10353 // end catch_context.cpp
10354 // start catch_debug_console.cpp
10355 // start catch_debug_console.h
10356
10357 #include <string>
10358
10359 namespace Catch {
10360     void writeToDebugConsole( std::string const& text );
10361 }

```

```

10364 }
10365
10366 // end catch_debug_console.h
10367 #if defined(CATCH_CONFIG_ANDROID_LOGWRITE)
10368 #include <android/log.h>
10369
10370 namespace Catch {
10371     void writeToDebugConsole( std::string const& text ) {
10372         __android_log_write( ANDROID_LOG_DEBUG, "Catch", text.c_str() );
10373     }
10374 }
10375
10376 #elif defined(CATCH_PLATFORM_WINDOWS)
10377
10378 namespace Catch {
10379     void writeToDebugConsole( std::string const& text ) {
10380         ::OutputDebugStringA( text.c_str() );
10381     }
10382 }
10383
10384 #else
10385
10386 namespace Catch {
10387     void writeToDebugConsole( std::string const& text ) {
10388         // !TBD: Need a version for Mac/ XCode and other IDEs
10389         Catch::cout() << text;
10390     }
10391 }
10392
10393 #endif // Platform
10394 // end catch_debug_console.cpp
10395 // start catch_debugger.cpp
10396
10397 #if defined(CATCH_PLATFORM_MAC) || defined(CATCH_PLATFORM_IPHONE)
10398
10399 # include <cassert>
10400 # include <sys/types.h>
10401 # include <unistd.h>
10402 # include <sys/sysctl.h>
10403 # include <ostream>
10404
10405 #ifdef __apple_build_version__
10406 // These headers will only compile with AppleClang (XCode)
10407 // For other compilers (Clang, GCC, ... ) we need to exclude them
10408 # include <sys/sysctl.h>
10409 #endif
10410
10411 namespace Catch {
10412 #ifdef __apple_build_version__
10413     // The following function is taken directly from the following technical note:
10414     // https://developer.apple.com/library/archive/qa/qa1361/_index.html
10415     // Returns true if the current process is being debugged (either
10416     // running under the debugger or has a debugger attached post facto).
10417     bool isDebuggerActive() {
10418         int mib[4];
10419         struct kinfo_proc info;
10420         std::size_t size;
10421
10422         // Initialize the flags so that, if sysctl fails for some bizarre
10423         // reason, we get a predictable result.
10424         info.kp_proc.p_flag = 0;
10425
10426         // Initialize mib, which tells sysctl the info we want, in this case
10427         // we're looking for information about a specific process ID.
10428         mib[0] = CTL_KERN;
10429         mib[1] = KERN_PROC;
10430         mib[2] = KERN_PROC_PID;
10431         mib[3] = getpid();
10432
10433         // Call sysctl.
10434         size = sizeof(info);
10435         if( sysctl(mib, sizeof(mib) / sizeof(*mib), &info, &size, nullptr, 0) != 0 ) {
10436             Catch::cerr() << "\n** Call to sysctl failed - unable to determine if debugger is active\n";
10437             return false;
10438         }
10439
10440         // We're being debugged if the P_TRACED flag is set.
10441         return ( info.kp_proc.p_flag & P_TRACED ) != 0 ;
10442     }
10443 #else
10444     bool isDebuggerActive() {
10445
10446 
```

```

10450         // We need to find another way to determine this for non-appleclang compilers on macOS
10451         return false;
10452     }
10453 #endif
10454 } // namespace Catch
10455
10456 #elif defined(CATCH_PLATFORM_LINUX)
10457 #include <fstream>
10458 #include <string>
10459
10460 namespace Catch{
10461     // The standard POSIX way of detecting a debugger is to attempt to
10462     // ptrace() the process, but this needs to be done from a child and not
10463     // this process itself to still allow attaching to this process later
10464     // if wanted, so is rather heavy. Under Linux we have the PID of the
10465     // "debugger" (which doesn't need to be gdb, of course, it could also
10466     // be strace, for example) in /proc/$PID/status, so just get it from
10467     // there instead.
10468     bool isDebuggerActive(){
10469         // Libstdc++ has a bug, where std::ifstream sets errno to 0
10470         // This way our users can properly assert over errno values
10471         ErrnoGuard guard;
10472         std::ifstream in("/proc/self/status");
10473         for( std::string line; std::getline(in, line); ) {
10474             static const int PREFIX_LEN = 11;
10475             if( line.compare(0, PREFIX_LEN, "TracerPid:") == 0 ) {
10476                 // We're traced if the PID is not 0 and no other PID starts
10477                 // with 0 digit, so it's enough to check for just a single
10478                 // character.
10479                 return line.length() > PREFIX_LEN && line[PREFIX_LEN] != '0';
10480             }
10481         }
10482     }
10483
10484     return false;
10485 } // namespace Catch
10486 #elif defined(_MSC_VER)
10487 extern "C" __declspec(dllimport) int __stdcall IsDebuggerPresent();
10488 namespace Catch {
10489     bool isDebuggerActive() {
10490         return IsDebuggerPresent() != 0;
10491     }
10492 }
10493 #elif defined(__MINGW32__)
10494 extern "C" __declspec(dllimport) int __stdcall IsDebuggerPresent();
10495 namespace Catch {
10496     bool isDebuggerActive() {
10497         return IsDebuggerPresent() != 0;
10498     }
10499 }
10500 #else
10501 namespace Catch {
10502     bool isDebuggerActive() { return false; }
10503 }
10504 #endif // Platform
10505 // end catch_debugger.cpp
10506 // start catch_decomposer.cpp
10507
10508 namespace Catch {
10509     ITransientExpression::~ITransientExpression() = default;
10510
10511     void formatReconstructedExpression( std::ostream &os, std::string const& lhs, StringRef op,
10512         std::string const& rhs ) {
10513         if( lhs.size() + rhs.size() < 40 &&
10514             lhs.find('\n') == std::string::npos &&
10515             rhs.find('\n') == std::string::npos )
10516             os << lhs << " " << op << " " << rhs;
10517         else
10518             os << lhs << "\n" << op << "\n" << rhs;
10519     }
10520 }
10521 // end catch_decomposer.cpp
10522 // start catch_enforce.cpp
10523
10524 #include <stdexcept>
10525
10526 namespace Catch {
10527     #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS) &&
10528         !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS_CUSTOM_HANDLER)
10529     [[noreturn]]
10530     void throw_exception(std::exception const& e) {
10531         Catch::cerr() << "Catch will terminate because it needed to throw an exception.\n"
10532             << "The message was: " << e.what() << '\n';
10533         std::terminate();
10534     }
10535 #endif
10536 #endif

```

```

10535
10536     [[noreturn]]
10537     void throw_logic_error(std::string const& msg) {
10538         throw_exception(std::logic_error(msg));
10539     }
10540
10541     [[noreturn]]
10542     void throw_domain_error(std::string const& msg) {
10543         throw_exception(std::domain_error(msg));
10544     }
10545
10546     [[noreturn]]
10547     void throw_runtime_error(std::string const& msg) {
10548         throw_exception(std::runtime_error(msg));
10549     }
10550
10551 } // namespace Catch;
10552 // end catch_enforce.cpp
10553 // start catch_enum_values_registry.cpp
10554 // start catch_enum_values_registry.h
10555
10556 #include <vector>
10557 #include <memory>
10558
10559 namespace Catch {
10560     namespace Detail {
10561         std::unique_ptr<EnumInfo> makeEnumInfo( StringRef enumName, StringRef allValueNames,
10562             std::vector<int> const& values );
10563
10564         class EnumValuesRegistry : public IMutableEnumValuesRegistry {
10565             std::vector<std::unique_ptr<EnumInfo>> m_enumInfos;
10566
10567             EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums, std::vector<int>
10568                 const& values) override;
10569         };
10570
10571         std::vector<StringRef> parseEnums( StringRef enums );
10572     } // Detail
10573 } // Catch
10574
10575 // end catch_enum_values_registry.h
10576
10577 #include <map>
10578 #include <cassert>
10579
10580 namespace Catch {
10581     IMutableEnumValuesRegistry::~IMutableEnumValuesRegistry() {}
10582
10583     namespace Detail {
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             std::vector<StringRef> parseEnums( StringRef enums ) {
10597                 auto enumValues = splitStringRef( enums, ',' );
10598                 std::vector<StringRef> parsed;
10599                 parsed.reserve( enumValues.size() );
10600                 for( auto const& enumValue : enumValues ) {
10601                     parsed.push_back(trim(extractInstanceName(enumValue)));
10602                 }
10603                 return parsed;
10604             }
10605
10606             EnumInfo::~EnumInfo() {}
10607
10608             StringRef EnumInfo::lookup( int value ) const {
10609                 for( auto const& valueToName : m_values ) {
10610                     if( valueToName.first == value )
10611                         return valueToName.second;
10612                 }
10613                 return "{** unexpected enum value **}"_sr;
10614             }
10615
10616             StringRef EnumInfo::lookup( int value ) const {
10617                 for( auto const& valueToName : m_values ) {
10618                     if( valueToName.first == value )
10619                         return valueToName.second;
10620                 }
10621                 return "{** unexpected enum value **}"_sr;
10622             }
10623         }
10624     }
10625 }

```

```

10620     }
10621
10622     std::unique_ptr<EnumInfo> makeEnumInfo( StringRef enumName, StringRef allValueNames,
std::vector<int> const& values ) {
10623         std::unique_ptr<EnumInfo> enumInfo( new EnumInfo );
10624         enumInfo->m_name = enumName;
10625         enumInfo->m_values.reserve( values.size() );
10626
10627         const auto valueNames = Catch::Detail::parseEnums( allValueNames );
10628         assert( valueNames.size() == values.size() );
10629         std::size_t i = 0;
10630         for( auto value : values )
10631             enumInfo->m_values.emplace_back(value, valueNames[i++]);
10632
10633         return enumInfo;
10634     }
10635
10636     EnumInfo const& EnumValuesRegistry::registerEnum( StringRef enumName, StringRef allValueNames,
std::vector<int> const& values ) {
10637         m_enumInfos.push_back(makeEnumInfo(enumName, allValueNames, values));
10638         return *m_enumInfos.back();
10639     }
10640
10641     } // Detail
10642 } // Catch
10643
10644 // end catch_enum_values_registry.cpp
10645 // start catch_errno_guard.cpp
10646
10647 #include <cerrno>
10648
10649 namespace Catch {
10650     ErrnoGuard::ErrnoGuard():m_oldErrno(errno){}
10651     ErrnoGuard::~~ErrnoGuard() { errno = m_oldErrno; }
10652 }
10653 // end catch_errno_guard.cpp
10654 // start catch_exception_translator_registry.cpp
10655
10656 // start catch_exception_translator_registry.h
10657
10658 #include <vector>
10659 #include <string>
10660 #include <memory>
10661
10662 namespace Catch {
10663
10664     class ExceptionTranslatorRegistry : public IExceptionTranslatorRegistry {
10665     public:
10666         ~ExceptionTranslatorRegistry();
10667         virtual void registerTranslator( const IExceptionTranslator* translator );
10668         std::string translateActiveException() const override;
10669         std::string tryTranslators() const;
10670
10671     private:
10672         std::vector<std::unique_ptr<IExceptionTranslator const> m_translators;
10673     };
10674 }
10675
10676 // end catch_exception_translator_registry.h
10677 #ifdef __OBJC__
10678 #import "Foundation/Foundation.h"
10679 #endif
10680
10681 namespace Catch {
10682
10683     ExceptionTranslatorRegistry::~ExceptionTranslatorRegistry() {
10684     }
10685
10686     void ExceptionTranslatorRegistry::registerTranslator( const IExceptionTranslator* translator ) {
10687         m_translators.push_back( std::unique_ptr<const IExceptionTranslator>( translator ) );
10688     }
10689
10690 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
10691     std::string ExceptionTranslatorRegistry::translateActiveException() const {
10692         try {
10693 #ifdef __OBJC__
10694             // In Objective-C try objective-c exceptions first
10695             @try {
10696                 return tryTranslators();
10697             }
10698             @catch (NSException *exception) {
10699                 return Catch::Detail::stringify( [exception description] );
10700             }
10701 #else
10702             // Compiling a mixed mode project with MSVC means that CLR
10703             // exceptions will be caught in (...) as well. However, these
10704             // do not fill-in std::current_exception and thus lead to crash

```

```

10705         // when attempting rethrow.
10706         // /EHa switch also causes structured exceptions to be caught
10707         // here, but they fill-in current_exception properly, so
10708         // at worst the output should be a little weird, instead of
10709         // causing a crash.
10710         if (std::current_exception() == nullptr) {
10711             return "Non C++ exception. Possibly a CLR exception.";
10712         }
10713         return tryTranslators();
10714 #endif
10715     }
10716     catch( TestFailureException& ) {
10717         std::rethrow_exception(std::current_exception());
10718     }
10719     catch( std::exception& ex ) {
10720         return ex.what();
10721     }
10722     catch( std::string& msg ) {
10723         return msg;
10724     }
10725     catch( const char* msg ) {
10726         return msg;
10727     }
10728     catch(...) {
10729         return "Unknown exception";
10730     }
10731 }
10732
10733 std::string ExceptionTranslatorRegistry::tryTranslators() const {
10734     if (m_translators.empty()) {
10735         std::rethrow_exception(std::current_exception());
10736     } else {
10737         return m_translators[0]->translate(m_translators.begin() + 1, m_translators.end());
10738     }
10739 }
10740
10741 #else // ^^ Exceptions are enabled // Exceptions are disabled vv
10742     std::string ExceptionTranslatorRegistry::translateActiveException() const {
10743         CATCH_INTERNAL_ERROR("Attempted to translate active exception under
10744         CATCH_CONFIG_DISABLE_EXCEPTIONS!");
10745     }
10746     std::string ExceptionTranslatorRegistry::tryTranslators() const {
10747         CATCH_INTERNAL_ERROR("Attempted to use exception translators under
10748         CATCH_CONFIG_DISABLE_EXCEPTIONS!");
10749     }
10750 #endif
10751 }
10752 // end catch_exception_translator_registry.cpp
10753 // start catch_fatal_condition.cpp
10754
10755 #include <algorithm>
10756
10757 #if !defined( CATCH_CONFIG_WINDOWS_SEH ) && !defined( CATCH_CONFIG_POSIX_SIGNALS )
10758 namespace Catch {
10759     // If neither SEH nor signal handling is required, the handler impls
10760     // do not have to do anything, and can be empty.
10761     void FatalConditionHandler::engage_platform() {}
10762     void FatalConditionHandler::disengage_platform() {}
10763     FatalConditionHandler::FatalConditionHandler() = default;
10764     FatalConditionHandler::~FatalConditionHandler() = default;
10765 } // end namespace Catch
10766
10767 #endif // !CATCH_CONFIG_WINDOWS_SEH && !CATCH_CONFIG_POSIX_SIGNALS
10768
10769 #if defined( CATCH_CONFIG_WINDOWS_SEH ) && defined( CATCH_CONFIG_POSIX_SIGNALS )
10770 #error "Inconsistent configuration: Windows' SEH handling and POSIX signals cannot be enabled at the same time"
10771 #endif // CATCH_CONFIG_WINDOWS_SEH && CATCH_CONFIG_POSIX_SIGNALS
10772
10773 #if defined( CATCH_CONFIG_WINDOWS_SEH ) || defined( CATCH_CONFIG_POSIX_SIGNALS )
10774 namespace {
10775     namespace {
10776         void reportFatal( char const * const message ) {
10777             Catch::getCurrentContext().getResultCapture()->handleFatalErrorCondition( message );
10778         }
10779     }
10780     constexpr std::size_t minStackSizeForErrors = 32 * 1024;
10781 } // end unnamed namespace
10782
10783 #endif // CATCH_CONFIG_WINDOWS_SEH || CATCH_CONFIG_POSIX_SIGNALS
10784
10785 #if defined( CATCH_CONFIG_WINDOWS_SEH )
10786

```

```

10793
10794 namespace Catch {
10795
10796     struct SignalDefs { DWORD id; const char* name; };
10797
10798     // There is no 1-1 mapping between signals and windows exceptions.
10799     // Windows can easily distinguish between SO and SigSegV,
10800     // but SigInt, SigTerm, etc are handled differently.
10801     static SignalDefs signalDefs[] = {
10802         { static_cast<DWORD>(EXCEPTION_ILLEGAL_INSTRUCTION), "SIGILL - Illegal instruction signal" },
10803         { static_cast<DWORD>(EXCEPTION_STACK_OVERFLOW), "SIGSEGV - Stack overflow" },
10804         { static_cast<DWORD>(EXCEPTION_ACCESS_VIOLATION), "SIGSEGV - Segmentation violation signal" },
10805         { static_cast<DWORD>(EXCEPTION_INT_DIVIDE_BY_ZERO), "Divide by zero error" },
10806     };
10807
10808     static LONG CALLBACK handleVectoredException(PEXCEPTION_POINTERS ExceptionInfo) {
10809         for (auto const& def : signalDefs) {
10810             if (ExceptionInfo->ExceptionRecord->ExceptionCode == def.id) {
10811                 reportFatal(def.name);
10812             }
10813         }
10814         // If its not an exception we care about, pass it along.
10815         // This stops us from eating debugger breaks etc.
10816         return EXCEPTION_CONTINUE_SEARCH;
10817     }
10818
10819     // Since we do not support multiple instantiations, we put these
10820     // into global variables and rely on cleaning them up in outlined
10821     // constructors/destructors
10822     static PVOID exceptionHandlerHandle = nullptr;
10823
10824     // For MSVC, we reserve part of the stack memory for handling
10825     // memory overflow structured exception.
10826     FatalConditionHandler::FatalConditionHandler() {
10827         ULONG guaranteeSize = static_cast<ULONG>(minStackSizeForErrors);
10828         if (!SetThreadStackGuarantee(&guaranteeSize)) {
10829             // We do not want to fully error out, because needing
10830             // the stack reserve should be rare enough anyway.
10831             Catch::cerr()
10832                 << "Failed to reserve piece of stack."
10833                 << " Stack overflows will not be reported successfully.";
10834         }
10835     }
10836
10837     // We do not attempt to unset the stack guarantee, because
10838     // Windows does not support lowering the stack size guarantee.
10839     FatalConditionHandler::~FatalConditionHandler() = default;
10840
10841     void FatalConditionHandler::engage_platform() {
10842         // Register as first handler in current chain
10843         exceptionHandlerHandle = AddVectoredExceptionHandler(1, handleVectoredException);
10844         if (!exceptionHandlerHandle) {
10845             CATCH_RUNTIME_ERROR("Could not register vectored exception handler");
10846         }
10847     }
10848
10849     void FatalConditionHandler::disengage_platform() {
10850         if (!RemoveVectoredExceptionHandler(exceptionHandlerHandle)) {
10851             CATCH_RUNTIME_ERROR("Could not unregister vectored exception handler");
10852         }
10853         exceptionHandlerHandle = nullptr;
10854     }
10855 } // end namespace Catch
10856
10857 #endif // CATCH_CONFIG_WINDOWS_SEH
10858
10859 #if defined(CATCH_CONFIG_POSIX_SIGNALS)
10860
10861 #include <signal.h>
10862
10863 namespace Catch {
10864
10865     struct SignalDefs {
10866         int id;
10867         const char* name;
10868     };
10869
10870     static SignalDefs signalDefs[] = {
10871         { SIGINT, "SIGINT - Terminal interrupt signal" },
10872         { SIGILL, "SIGILL - Illegal instruction signal" },
10873         { SIGFPE, "SIGFPE - Floating point error signal" },
10874         { SIGSEGV, "SIGSEGV - Segmentation violation signal" },
10875         { SIGTERM, "SIGTERM - Termination request signal" },
10876         { SIGABRT, "SIGABRT - Abort (abnormal termination) signal" }
10877     };
10878
10879

```



```

10880 // Older GCCs trigger -Wmissing-field-initializers for T foo = {}
10881 // which is zero initialization, but not explicit. We want to avoid
10882 // that.
10883 #if defined(__GNUC__)
10884 #   pragma GCC diagnostic push
10885 #   pragma GCC diagnostic ignored "-Wmissing-field-initializers"
10886 #endif
10887
10888     static char* altStackMem = nullptr;
10889     static std::size_t altStackSize = 0;
10890     static stack_t oldSigStack{};
10891     static struct sigaction oldSigActions[sizeof(signalDefs) / sizeof(SignalDefs)]{};
10892
10893     static void restorePreviousSignalHandlers() {
10894         // We set signal handlers back to the previous ones. Hopefully
10895         // nobody overwrote them in the meantime, and doesn't expect
10896         // their signal handlers to live past ours given that they
10897         // installed them after ours..
10898         for (std::size_t i = 0; i < sizeof(signalDefs) / sizeof(SignalDefs); ++i) {
10899             sigaction(signalDefs[i].id, &oldSigActions[i], nullptr);
10900         }
10901         // Return the old stack
10902         sigaltstack(&oldSigStack, nullptr);
10903     }
10904
10905     static void handleSignal( int sig ) {
10906         char const * name = "<unknown signal>";
10907         for (auto const& def : signalDefs) {
10908             if (sig == def.id) {
10909                 name = def.name;
10910                 break;
10911             }
10912         }
10913         // We need to restore previous signal handlers and let them do
10914         // their thing, so that the users can have the debugger break
10915         // when a signal is raised, and so on.
10916         restorePreviousSignalHandlers();
10917         reportFatal( name );
10918         raise( sig );
10919     }
10920
10921     FatalConditionHandler::FatalConditionHandler() {
10922         assert(!altStackMem && "Cannot initialize POSIX signal handler when one already exists");
10923         if (altStackSize == 0) {
10924             altStackSize = std::max(static_cast<size_t>(SIGSTKSZ), minStackSizeForErrors);
10925         }
10926         altStackMem = new char[altStackSize]();
10927     }
10928
10929     FatalConditionHandler::~FatalConditionHandler() {
10930         delete[] altStackMem;
10931         // We signal that another instance can be constructed by zeroing
10932         // out the pointer.
10933         altStackMem = nullptr;
10934     }
10935
10936     void FatalConditionHandler::engage_platform() {
10937         stack_t sigStack;
10938         sigStack.ss_sp = altStackMem;
10939         sigStack.ss_size = altStackSize;
10940         sigStack.ss_flags = 0;
10941         sigaltstack(&sigStack, &oldSigStack);
10942         struct sigaction sa = { };
10943
10944         sa.sa_handler = handleSignal;
10945         sa.sa_flags = SA_ONSTACK;
10946         for (std::size_t i = 0; i < sizeof(signalDefs)/sizeof(SignalDefs); ++i) {
10947             sigaction(signalDefs[i].id, &sa, &oldSigActions[i]);
10948         }
10949     }
10950
10951 #if defined(__GNUC__)
10952 #   pragma GCC diagnostic pop
10953 #endif
10954
10955     void FatalConditionHandler::disengage_platform() {
10956         restorePreviousSignalHandlers();
10957     }
10958 } // end namespace Catch
10959
10960 #endif // CATCH_CONFIG_POSIX_SIGNALS
10961 // end catch_fatal_condition.cpp
10962 // start catch_generators.cpp
10963
10964 #include <limits>
10965 #include <set>

```

```

10967
10968 namespace Catch {
10969
10970     IGeneratorTracker::~IGeneratorTracker() {}
10971
10972     const char* GeneratorException::what() const noexcept {
10973         return m_msg;
10974     }
10975
10976     namespace Generators {
10977
10978         GeneratorUntypedBase::~GeneratorUntypedBase() {}
10979
10980         auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
10981             IGeneratorTracker& {
10982             return getResultCapture().acquireGeneratorTracker( generatorName, lineInfo );
10983         } // namespace Generators
10984     } // namespace Catch
10985 // end catch_generators.cpp
10986 // start catch_interfaces_capture.cpp
10987 // start catch_interfaces_config.cpp
10988 namespace Catch {
10989     IResultCapture::~IResultCapture() = default;
10990 } // end catch_interfaces_capture.cpp
10991 // start catch_interfaces_config.cpp
10992 namespace Catch {
10993     IConfig::~IConfig() = default;
10994 } // end catch_interfaces_config.cpp
10995 // start catch_interfaces_exception.cpp
10996 namespace Catch {
10997     IExceptionTranslator::~IExceptionTranslator() = default;
10998     IExceptionTranslatorRegistry::~IExceptionTranslatorRegistry() = default;
10999 } // end catch_interfaces_exception.cpp
11000 // start catch_interfaces_registry_hub.cpp
11001 namespace Catch {
11002     IRegistryHub::~IRegistryHub() = default;
11003     IMutableRegistryHub::~IMutableRegistryHub() = default;
11004 } // end catch_interfaces_registry_hub.cpp
11005 // start catch_interfaces_reporter.cpp
11006 // start catch_reporter_listening.h
11007 namespace Catch {
11008
11009     class ListeningReporter : public IStreamingReporter {
11010     public:
11011         ListeningReporter();
11012
11013         void addListener( IStreamingReporterPtr&& listener );
11014         void addReporter( IStreamingReporterPtr&& reporter );
11015
11016     public: // IStreamingReporter
11017         ReporterPreferences getPreferences() const override;
11018
11019         void noMatchingTestCases( std::string const& spec ) override;
11020
11021         void reportInvalidArguments( std::string const& arg ) override;
11022
11023         static std::set<Verbosity> getSupportedVerbsities();
11024
11025 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
11026         void benchmarkPreparing( std::string const& name ) override;
11027         void benchmarkStarting( BenchmarkInfo const& benchmarkInfo ) override;
11028         void benchmarkEnded( BenchmarkStats<> const& benchmarkStats ) override;
11029         void benchmarkFailed( std::string const& ) override;
11030 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
11031
11032         void testRunStarting( TestRunInfo const& testRunInfo ) override;
11033         void testGroupStarting( GroupInfo const& groupInfo ) override;
11034         void testCaseStarting( TestCaseInfo const& testInfo ) override;
11035         void sectionStarting( SectionInfo const& sectionInfo ) override;
11036         void assertionStarting( AssertionInfo const& assertionInfo ) override;
11037     };
11038 }

```

```

11053
11054     // The return value indicates if the messages buffer should be cleared:
11055     bool assertionEnded( AssertionStats const& assertionStats ) override;
11056     void sectionEnded( SectionStats const& sectionStats ) override;
11057     void testCaseEnded( TestCaseStats const& testCaseStats ) override;
11058     void testGroupEnded( TestGroupStats const& testGroupStats ) override;
11059     void testRunEnded( TestRunStats const& testRunStats ) override;
11060
11061     void skipTest( TestCaseInfo const& testInfo ) override;
11062     bool isMulti() const override;
11063
11064 };
11065
11066 } // end namespace Catch
11067
11068 // end catch_reporter_listening.h
11069 namespace Catch {
11070
11071     ReporterConfig::ReporterConfig( IConfigPtr const& _fullConfig )
11072         : m_stream( &_fullConfig->stream() ), m_fullConfig( _fullConfig ) {}
11073
11074     ReporterConfig::ReporterConfig( IConfigPtr const& _fullConfig, std::ostream& _stream )
11075         : m_stream( _stream ), m_fullConfig( _fullConfig ) {}
11076
11077     std::ostream& ReporterConfig::stream() const { return *m_stream; }
11078     IConfigPtr ReporterConfig::fullConfig() const { return m_fullConfig; }
11079
11080     TestRunInfo::TestRunInfo( std::string const& _name ) : name( _name ) {}
11081
11082     GroupInfo::GroupInfo( std::string const& _name,
11083                          std::size_t _groupIndex,
11084                          std::size_t _groupsCount )
11085         : name( _name ),
11086           groupIndex( _groupIndex ),
11087           groupsCounts( _groupsCount )
11088     {}
11089
11090     AssertionStats::AssertionStats( AssertionResult const& _assertionResult,
11091                                    std::vector<MessageInfo> const& _infoMessages,
11092                                    Totals const& _totals )
11093         : assertionResult( _assertionResult ),
11094           infoMessages( _infoMessages ),
11095           totals( _totals )
11096     {
11097         assertionResult.m_resultData.lazyExpression.m_transientExpression =
11098             _assertionResult.m_resultData.lazyExpression.m_transientExpression;
11099
11100         if( assertionResult.hasMessage() ) {
11101             // Copy message into messages list.
11102             // !TBD This should have been done earlier, somewhere
11103             MessageBuilder builder( assertionResult.getTestMacroName(),
11104                                     assertionResult.getSourceInfo(), assertionResult.getResultType() );
11105             builder « assertionResult.getMessage();
11106             builder.m_info.message = builder.m_stream.str();
11107             infoMessages.push_back( builder.m_info );
11108         }
11109     }
11110
11111     AssertionStats::~AssertionStats() = default;
11112
11113     SectionStats::SectionStats( SectionInfo const& _sectionInfo,
11114                                Counts const& _assertions,
11115                                double _durationInSeconds,
11116                                bool _missingAssertions )
11117         : sectionInfo( _sectionInfo ),
11118           assertions( _assertions ),
11119           durationInSeconds( _durationInSeconds ),
11120           missingAssertions( _missingAssertions )
11121     {}
11122
11123     SectionStats::~SectionStats() = default;
11124
11125     TestCaseStats::TestCaseStats( TestCaseInfo const& _testInfo,
11126                                  Totals const& _totals,
11127                                  std::string const& _stdOut,
11128                                  std::string const& _stdErr,
11129                                  bool _aborting )
11130         : testInfo( _testInfo ),
11131           totals( _totals ),
11132           stdOut( _stdOut ),
11133           stdErr( _stdErr ),
11134           aborting( _aborting )
11135     {}
11136
11137     TestCaseStats::~TestCaseStats() = default;

```

```

11138     TestGroupStats::TestGroupStats( GroupInfo const& _groupInfo,
11139                                     Totals const& _totals,
11140                                     bool _aborting )
11141     :   groupInfo( _groupInfo ),
11142         totals( _totals ),
11143         aborting( _aborting )
11144     {}
11145
11146     TestGroupStats::TestGroupStats( GroupInfo const& _groupInfo )
11147     :   groupInfo( _groupInfo ),
11148         aborting( false )
11149     {}
11150
11151     TestGroupStats::~TestGroupStats() = default;
11152
11153     TestRunStats::TestRunStats(   TestRunInfo const& _runInfo,
11154                                 Totals const& _totals,
11155                                 bool _aborting )
11156     :   runInfo( _runInfo ),
11157         totals( _totals ),
11158         aborting( _aborting )
11159     {}
11160
11161     TestRunStats::~TestRunStats() = default;
11162
11163     void IStreamingReporter::fatalErrorEncountered( StringRef ) {}
11164     bool IStreamingReporter::isMulti() const { return false; }
11165
11166     IReporterFactory::~IReporterFactory() = default;
11167     IReporterRegistry::~IReporterRegistry() = default;
11168
11169 } // end namespace Catch
11170 // end catch_interfaces_reporter.cpp
11171 // start catch_interfaces_runner.cpp
11172
11173 namespace Catch {
11174     IRunner::~IRunner() = default;
11175 }
11176 // end catch_interfaces_runner.cpp
11177 // start catch_interfaces_testcase.cpp
11178
11179 namespace Catch {
11180     ITestInvoker::~ITestInvoker() = default;
11181     ITestCaseRegistry::~ITestCaseRegistry() = default;
11182 }
11183 // end catch_interfaces_testcase.cpp
11184 // start catch_leak_detector.cpp
11185
11186 #ifdef CATCH_CONFIG_WINDOWS_CRTDBG
11187 #include <crtdbg.h>
11188
11189 namespace Catch {
11190
11191     LeakDetector::LeakDetector() {
11192         int flag = _CrtSetDbgFlag(_CRTDBG_REPORT_FLAG);
11193         flag |= _CRTDBG_LEAK_CHECK_DF;
11194         flag |= _CRTDBG_ALLOC_MEM_DF;
11195         _CrtSetDbgFlag(flag);
11196         _CrtSetReportMode(_CRT_WARN, _CRTDBG_MODE_FILE | _CRTDBG_MODE_DEBUG);
11197         _CrtSetReportFile(_CRT_WARN, _CRTDBG_FILE_STDERR);
11198         // Change this to leaking allocation's number to break there
11199         _CrtSetBreakAlloc(-1);
11200     }
11201 }
11202
11203 #else
11204
11205 Catch::LeakDetector::LeakDetector() {}
11206
11207 #endif
11208
11209 Catch::LeakDetector::~LeakDetector() {
11210     Catch::cleanUp();
11211 }
11212 // end catch_leak_detector.cpp
11213 // start catch_list.cpp
11214
11215 // start catch_list.h
11216
11217 #include <set>
11218
11219 namespace Catch {
11220
11221     std::size_t listTests( Config const& config );
11222
11223     std::size_t listTestsNamesOnly( Config const& config );
11224

```

```

11225     struct TagInfo {
11226         void add( std::string const& spelling );
11227         std::string all() const;
11228
11229         std::set<std::string> spellings;
11230         std::size_t count = 0;
11231     };
11232
11233     std::size_t listTags( Config const& config );
11234
11235     std::size_t listReporters();
11236
11237     Option<std::size_t> list( std::shared_ptr<Config> const& config );
11238
11239 } // end namespace Catch
11240
11241 // end catch_list.h
11242 // start catch_text.h
11243
11244 namespace Catch {
11245     using namespace clara::TextFlow;
11246 }
11247
11248 // end catch_text.h
11249 #include <limits>
11250 #include <algorithm>
11251 #include <iomanip>
11252
11253 namespace Catch {
11254
11255     std::size_t listTests( Config const& config ) {
11256         TestSpec const& testSpec = config.testSpec();
11257         if( config.hasTestFilters() )
11258             Catch::cout() << "Matching test cases:\n";
11259         else {
11260             Catch::cout() << "All available test cases:\n";
11261         }
11262
11263         auto matchedTestCases = filterTests( getAllTestCasesSorted( config ), testSpec, config );
11264         for( auto const& testCaseInfo : matchedTestCases ) {
11265             Colour::Code colour = testCaseInfo.isHidden()
11266                 ? Colour::SecondaryText
11267                 : Colour::None;
11268             Colour colourGuard( colour );
11269
11270             Catch::cout() << Column( testCaseInfo.name ).initialIndent( 2 ).indent( 4 ) << "\n";
11271             if( config.verbosity() >= Verbosity::High ) {
11272                 Catch::cout() << Column( Catch::Detail::stringify( testCaseInfo.lineInfo ) ).indent( 4 )
11273                 << std::endl;
11274                 std::string description = testCaseInfo.description;
11275                 if( description.empty() )
11276                     description = "(NO DESCRIPTION)";
11277                 Catch::cout() << Column( description ).indent( 4 ) << std::endl;
11278             }
11279             if( !testCaseInfo.tags.empty() )
11280                 Catch::cout() << Column( testCaseInfo.tagsAsString() ).indent( 6 ) << "\n";
11281
11282             if( !config.hasTestFilters() )
11283                 Catch::cout() << pluralise( matchedTestCases.size(), "test case" ) << '\n' << std::endl;
11284             else
11285                 Catch::cout() << pluralise( matchedTestCases.size(), "matching test case" ) << '\n' <<
11286                 std::endl;
11287             return matchedTestCases.size();
11288         }
11289
11290     std::size_t listTestsNamesOnly( Config const& config ) {
11291         TestSpec const& testSpec = config.testSpec();
11292         std::size_t matchedTests = 0;
11293         std::vector<TestCase> matchedTestCases = filterTests( getAllTestCasesSorted( config ),
11294             testSpec, config );
11295         for( auto const& testCaseInfo : matchedTestCases ) {
11296             matchedTests++;
11297             if( startsWith( testCaseInfo.name, '#' ) )
11298                 Catch::cout() << "'" << testCaseInfo.name << "'";
11299             else
11300                 Catch::cout() << testCaseInfo.name;
11301             if ( config.verbosity() >= Verbosity::High )
11302                 Catch::cout() << "\t@" << testCaseInfo.lineInfo;
11303             Catch::cout() << std::endl;
11304         }
11305         return matchedTests;
11306     }
11307
11308     void TagInfo::add( std::string const& spelling ) {
11309         ++count;
11310         spellings.insert( spelling );
11311     }

```

```

11309     }
11310
11311     std::string TagInfo::all() const {
11312         size_t size = 0;
11313         for (auto const& spelling : spellings) {
11314             // Add 2 for the brackets
11315             size += spelling.size() + 2;
11316         }
11317
11318         std::string out; out.reserve(size);
11319         for (auto const& spelling : spellings) {
11320             out += '[';
11321             out += spelling;
11322             out += ']';
11323         }
11324         return out;
11325     }
11326
11327     std::size_t listTags( Config const& config ) {
11328         TestSpec const& testSpec = config.testSpec();
11329         if( config.hasTestFilters() )
11330             Catch::cout() << "Tags for matching test cases:\n";
11331         else {
11332             Catch::cout() << "All available tags:\n";
11333         }
11334
11335         std::map<std::string, TagInfo> tagCounts;
11336
11337         std::vector<TestCase> matchedTestCases = filterTests( getAllTestCasesSorted( config ),
testSpec, config );
11338         for( auto const& testCase : matchedTestCases ) {
11339             for( auto const& tagName : testCase.getTestCaseInfo().tags ) {
11340                 std::string lcaseTagName = toLower( tagName );
11341                 auto countIt = tagCounts.find( lcaseTagName );
11342                 if( countIt == tagCounts.end() )
11343                     countIt = tagCounts.insert( std::make_pair( lcaseTagName, TagInfo() ) ).first;
11344                 countIt->second.add( tagName );
11345             }
11346         }
11347
11348         for( auto const& tagCount : tagCounts ) {
11349             ReusableStringStream rss;
11350             rss << " " << std::setw(2) << tagCount.second.count << " ";
11351             auto str = rss.str();
11352             auto wrapper = Column( tagCount.second.all() )
11353                 .initialIndent( 0 )
11354                 .indent( str.size() )
11355                 .width( CATCH_CONFIG_CONSOLE_WIDTH-10 );
11356             Catch::cout() << str << wrapper << '\n';
11357         }
11358         Catch::cout() << pluralise( tagCounts.size(), "tag" ) << '\n' << std::endl;
11359         return tagCounts.size();
11360     }
11361
11362     std::size_t listReporters() {
11363         Catch::cout() << "Available reporters:\n";
11364         IReporterRegistry::FactoryMap const& factories =
getRegistryHub().getReporterRegistry().getFactories();
11365         std::size_t maxNameLen = 0;
11366         for( auto const& factoryKvp : factories )
11367             maxNameLen = (std::max)( maxNameLen, factoryKvp.first.size() );
11368
11369         for( auto const& factoryKvp : factories ) {
11370             Catch::cout()
11371                 << Column( factoryKvp.first + ":" )
11372                     .indent(2)
11373                     .width( 5+maxNameLen )
11374                 + Column( factoryKvp.second->getDescription() )
11375                     .initialIndent(0)
11376                     .indent(2)
11377                     .width( CATCH_CONFIG_CONSOLE_WIDTH - maxNameLen-8 )
11378                 << "\n";
11379         }
11380         Catch::cout() << std::endl;
11381         return factories.size();
11382     }
11383
11384     Option<std::size_t> list( std::shared_ptr<Config> const& config ) {
11385         Option<std::size_t> listedCount;
11386         setCurrentMutableContext().setConfig( config );
11387         if( config->listTests() )
11388             listedCount = listedCount.valueOr(0) + listTests( *config );
11389         if( config->listTestNamesOnly() )
11390             listedCount = listedCount.valueOr(0) + listTestsNamesOnly( *config );
11391         if( config->listTags() )
11392             listedCount = listedCount.valueOr(0) + listTags( *config );
11393         if( config->listReporters() )

```

```

11394         listedCount = listedCount.valueOr(0) + listReporters();
11395         return listedCount;
11396     }
11397
11398 } // end namespace Catch
11399 // end catch_list.cpp
11400 // start catch_matchers.cpp
11401
11402 namespace Catch {
11403     namespace Matchers {
11404         namespace Impl {
11405
11406             std::string MatcherUntypedBase::toString() const {
11407                 if ( m_cachedToString.empty() )
11408                     m_cachedToString = describe();
11409                 return m_cachedToString;
11410             }
11411
11412             MatcherUntypedBase::~MatcherUntypedBase() = default;
11413
11414         } // namespace Impl
11415     } // namespace Matchers
11416
11417     using namespace Matchers;
11418     using Matchers::Impl::MatcherBase;
11419
11420 } // namespace Catch
11421 // end catch_matchers.cpp
11422 // start catch_matchers_exception.cpp
11423
11424 namespace Catch {
11425     namespace Matchers {
11426         namespace Exception {
11427
11428             bool ExceptionMessageMatcher::match(std::exception const& ex) const {
11429                 return ex.what() == m_message;
11430             }
11431
11432             std::string ExceptionMessageMatcher::describe() const {
11433                 return "exception message matches \"" + m_message + "\"";
11434             }
11435
11436         }
11437
11438         Exception::ExceptionMessageMatcher Message(std::string const& message) {
11439             return Exception::ExceptionMessageMatcher(message);
11440         }
11441     } // namespace Exception
11442 } // namespace Matchers
11443 } // namespace Catch
11444 // end catch_matchers_exception.cpp
11445 // start catch_matchers_floating.cpp
11446
11447 // start catch_polyfills.hpp
11448
11449 namespace Catch {
11450     bool isnan(float f);
11451     bool isnan(double d);
11452 }
11453
11454 // end catch_polyfills.hpp
11455 // start catch_to_string.hpp
11456
11457 #include <string>
11458
11459 namespace Catch {
11460     template <typename T>
11461     std::string to_string(T const& t) {
11462         #if defined(CATCH_CONFIG_CPP11_TO_STRING)
11463             return std::to_string(t);
11464         #else
11465             ReusableStringStream rss;
11466             rss << t;
11467             return rss.str();
11468         #endif
11469     }
11470 } // end namespace Catch
11471
11472 // end catch_to_string.hpp
11473 #include <algorithm>
11474 #include <cmath>
11475 #include <cstdlib>
11476 #include <cstdint>
11477 #include <cstring>
11478 #include <sstream>
11479 #include <type_traits>
11480 #include <iomanip>

```

```

11481 #include <limits>
11482
11483 namespace Catch {
11484     namespace {
11485
11486         int32_t convert(float f) {
11487             static_assert(sizeof(float) == sizeof(int32_t), "Important ULP matcher assumption
violated");
11488             int32_t i;
11489             std::memcpy(&i, &f, sizeof(f));
11490             return i;
11491         }
11492
11493         int64_t convert(double d) {
11494             static_assert(sizeof(double) == sizeof(int64_t), "Important ULP matcher assumption
violated");
11495             int64_t i;
11496             std::memcpy(&i, &d, sizeof(d));
11497             return i;
11498         }
11499
11500         template <typename FP>
11501         bool almostEqualUlp(FP lhs, FP rhs, uint64_t maxUlpDiff) {
11502             // Comparison with NaN should always be false.
11503             // This way we can rule it out before getting into the ugly details
11504             if (Catch::isnan(lhs) || Catch::isnan(rhs)) {
11505                 return false;
11506             }
11507
11508             auto lc = convert(lhs);
11509             auto rc = convert(rhs);
11510
11511             if ((lc < 0) != (rc < 0)) {
11512                 // Potentially we can have +0 and -0
11513                 return lhs == rhs;
11514             }
11515
11516             // static cast as a workaround for IBM XLC
11517             auto ulpDiff = std::abs(static_cast<FP>(lc - rc));
11518             return static_cast<uint64_t>(ulpDiff) <= maxUlpDiff;
11519         }
11520
11521 #if defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
11522
11523         float nextafter(float x, float y) {
11524             return ::nextafterf(x, y);
11525         }
11526
11527         double nextafter(double x, double y) {
11528             return ::nextafter(x, y);
11529         }
11530
11531 #endif // ^^^ CATCH_CONFIG_GLOBAL_NEXTAFTER ^^^
11532
11533         template <typename FP>
11534         FP step(FP start, FP direction, uint64_t steps) {
11535             for (uint64_t i = 0; i < steps; ++i) {
11536 #if defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
11537                 start = Catch::nextafter(start, direction);
11538 #else
11539                 start = std::nextafter(start, direction);
11540 #endif
11541             }
11542             return start;
11543         }
11544
11545         // Performs equivalent check of std::fabs(lhs - rhs) <= margin
11546         // But without the subtraction to allow for INFINITY in comparison
11547         bool marginComparison(double lhs, double rhs, double margin) {
11548             return (lhs + margin >= rhs) && (rhs + margin >= lhs);
11549         }
11550
11551         template <typename FloatingPoint>
11552         void write(std::ostream& out, FloatingPoint num) {
11553             out << std::scientific
11554                 << std::setprecision(std::numeric_limits<FloatingPoint>::max_digits10 - 1)
11555                 << num;
11556         }
11557     } // end anonymous namespace
11558
11559     namespace Matchers {
11560         namespace Floating {
11561
11562             enum class FloatingPointKind : uint8_t {
11563                 Float,
11564                 Double
11565             };

```



```

11566         };
11567
11568     WithinAbsMatcher::WithinAbsMatcher(double target, double margin)
11569         :m_target{ target }, m_margin{ margin } {
11570         CATCH_ENFORCE(margin >= 0, "Invalid margin: " << margin << ". '
11571         << " Margin has to be non-negative.");
11572     }
11573
11574     // Performs equivalent check of std::fabs(lhs - rhs) <= margin
11575     // But without the subtraction to allow for INFINITY in comparison
11576     bool WithinAbsMatcher::match(double const& matchee) const {
11577         return (matchee + m_margin >= m_target) && (m_target + m_margin >= matchee);
11578     }
11579
11580     std::string WithinAbsMatcher::describe() const {
11581         return "is within " + ::Catch::Detail::stringify(m_margin) + " of " +
11582         ::Catch::Detail::stringify(m_target);
11583     }
11584
11585     WithinUlpMatcher::WithinUlpMatcher(double target, uint64_t ulps, FloatingPointKind
baseType)
11586         :m_target{ target }, m_ulps{ ulps }, m_type{ baseType } {
11587         CATCH_ENFORCE(m_type == FloatingPointKind::Double
11588         || m_ulps < (std::numeric_limits<uint32_t>::max)(),
11589         "Provided ULP is impossibly large for a float comparison.");
11590     }
11591 #if defined(__clang__)
11592 #pragma clang diagnostic push
11593 // Clang <3.5 reports on the default branch in the switch below
11594 #pragma clang diagnostic ignored "-Wunreachable-code"
11595 #endif
11596
11597     bool WithinUlpMatcher::match(double const& matchee) const {
11598         switch (m_type) {
11599             case FloatingPointKind::Float:
11600                 return almostEqualUlp<float>(static_cast<float>(matchee),
static_cast<float>(m_target), m_ulps);
11601             case FloatingPointKind::Double:
11602                 return almostEqualUlp<double>(matchee, m_target, m_ulps);
11603             default:
11604                 CATCH_INTERNAL_ERROR( "Unknown FloatingPointKind value" );
11605         }
11606     }
11607
11608 #if defined(__clang__)
11609 #pragma clang diagnostic pop
11610 #endif
11611
11612     std::string WithinUlpMatcher::describe() const {
11613         std::stringstream ret;
11614
11615         ret << "is within " << m_ulps << " ULPs of ";
11616
11617         if (m_type == FloatingPointKind::Float) {
11618             write(ret, static_cast<float>(m_target));
11619             ret << 'f';
11620         } else {
11621             write(ret, m_target);
11622         }
11623
11624         ret << " [";
11625         if (m_type == FloatingPointKind::Double) {
11626             write(ret, step(m_target, static_cast<double>(-INFINITY), m_ulps));
11627             ret << ", ";
11628             write(ret, step(m_target, static_cast<double>( INFINITY), m_ulps));
11629         } else {
11630             // We have to cast INFINITY to float because of MinGW, see #1782
11631             write(ret, step(static_cast<float>(m_target), static_cast<float>(-INFINITY),
m_ulps));
11632             ret << ", ";
11633             write(ret, step(static_cast<float>(m_target), static_cast<float>( INFINITY),
m_ulps));
11634         }
11635         ret << " ]";
11636
11637         return ret.str();
11638     }
11639
11640     WithinRelMatcher::WithinRelMatcher(double target, double epsilon):
11641         m_target(target),
11642         m_epsilon(epsilon){
11643         CATCH_ENFORCE(m_epsilon >= 0., "Relative comparison with epsilon < 0 does not make
sense.");
11644         CATCH_ENFORCE(m_epsilon < 1., "Relative comparison with epsilon >= 1 does not make
sense.");
11645     }

```

```

11646
11647         bool WithinRelMatcher::match(double const& matchee) const {
11648             const auto relMargin = m_epsilon * (std::max)(std::fabs(matchee),
11649                 std::fabs(m_target));
11649             return marginComparison(matchee, m_target,
11650                 std::isinf(relMargin)? 0 : relMargin);
11651         }
11652
11653         std::string WithinRelMatcher::describe() const {
11654             Catch::ReusableStringStream sstr;
11655             sstr << "and " << m_target << " are within " << m_epsilon * 100. << "% of each other";
11656             return sstr.str();
11657         }
11658
11659     } // namespace Floating
11660
11661     Floating::WithinUlpMatcher WithinULP(double target, uint64_t maxUlpDiff) {
11662         return Floating::WithinUlpMatcher(target, maxUlpDiff,
11663             Floating::FloatingPointKind::Double);
11664     }
11665
11666     Floating::WithinUlpMatcher WithinULP(float target, uint64_t maxUlpDiff) {
11667         return Floating::WithinUlpMatcher(target, maxUlpDiff,
11668             Floating::FloatingPointKind::Float);
11669     }
11670
11671     Floating::WithinAbsMatcher WithinAbs(double target, double margin) {
11672         return Floating::WithinAbsMatcher(target, margin);
11673     }
11674
11675     Floating::WithinRelMatcher WithinRel(double target, double eps) {
11676         return Floating::WithinRelMatcher(target, eps);
11677     }
11678
11679     Floating::WithinRelMatcher WithinRel(double target) {
11680         return Floating::WithinRelMatcher(target, std::numeric_limits<double>::epsilon() * 100);
11681     }
11682
11683     Floating::WithinRelMatcher WithinRel(float target, float eps) {
11684         return Floating::WithinRelMatcher(target, eps);
11685     }
11686
11687     Floating::WithinRelMatcher WithinRel(float target) {
11688         return Floating::WithinRelMatcher(target, std::numeric_limits<float>::epsilon() * 100);
11689     }
11690
11691     } // namespace Matchers
11692 } // namespace Catch
11693 // end catch_matchers_floating.cpp
11694 // start catch_matchers_generic.cpp
11695
11696 std::string Catch::Matchers::Generic::Detail::finalizeDescription(const std::string& desc) {
11697     if (desc.empty()) {
11698         return "matches undescribed predicate";
11699     } else {
11700         return "matches predicate: \"" + desc + "\"";
11701     }
11702 }
11703 // end catch_matchers_generic.cpp
11704 // start catch_matchers_string.cpp
11705
11706 #include <regex>
11707
11708 namespace Catch {
11709     namespace Matchers {
11710         namespace StdString {
11711             CasedString::CasedString( std::string const& str, CaseSensitive::Choice caseSensitivity )
11712                 : m_caseSensitivity( caseSensitivity ),
11713                   m_str( adjustString( str ) )
11714             {}
11715             std::string CasedString::adjustString( std::string const& str ) const {
11716                 return m_caseSensitivity == CaseSensitive::No
11717                     ? toLower( str )
11718                     : str;
11719             }
11720             std::string CasedString::caseSensitivitySuffix() const {
11721                 return m_caseSensitivity == CaseSensitive::No
11722                     ? " (case insensitive)"
11723                     : std::string();
11724             }
11725
11726             StringMatcherBase::StringMatcherBase( std::string const& operation, CasedString const&
11727                 comparator )
11728                 : m_comparator( comparator ),
11729                   m_operation( operation ) {}

```

```

11729         }
11730
11731         std::string StringMatcherBase::describe() const {
11732             std::string description;
11733             description.reserve(5 + m_operation.size() + m_comparator.m_str.size() +
11734                                 m_comparator.caseSensitivitySuffix().size());
11735             description += m_operation;
11736             description += ": \"";
11737             description += m_comparator.m_str;
11738             description += "\"";
11739             description += m_comparator.caseSensitivitySuffix();
11740             return description;
11741         }
11742
11743         EqualsMatcher::EqualsMatcher( CasedString const& comparator ) : StringMatcherBase(
11744 "equals", comparator ) {}
11745
11746         bool EqualsMatcher::match( std::string const& source ) const {
11747             return m_comparator.adjustString( source ) == m_comparator.m_str;
11748         }
11749
11750         ContainsMatcher::ContainsMatcher( CasedString const& comparator ) : StringMatcherBase(
11751 "contains", comparator ) {}
11752
11753         bool ContainsMatcher::match( std::string const& source ) const {
11754             return contains( m_comparator.adjustString( source ), m_comparator.m_str );
11755         }
11756
11757         StartsWithMatcher::StartsWithMatcher( CasedString const& comparator ) : StringMatcherBase(
11758 "starts with", comparator ) {}
11759
11760         bool StartsWithMatcher::match( std::string const& source ) const {
11761             return startsWith( m_comparator.adjustString( source ), m_comparator.m_str );
11762         }
11763
11764         EndsWithMatcher::EndsWithMatcher( CasedString const& comparator ) : StringMatcherBase(
11765 "ends with", comparator ) {}
11766
11767         bool EndsWithMatcher::match( std::string const& source ) const {
11768             return endsWith( m_comparator.adjustString( source ), m_comparator.m_str );
11769         }
11770
11771         RegexMatcher::RegexMatcher( std::string regex, CaseSensitive::Choice caseSensitivity ):
11772 m_regex( std::move( regex ), m_caseSensitivity( caseSensitivity ) ) {}
11773
11774         bool RegexMatcher::match( std::string const& matchee ) const {
11775             auto flags = std::regex::ECMAScript; // ECMAScript is the default syntax option anyway
11776             if ( m_caseSensitivity == CaseSensitive::Choice::No ) {
11777                 flags |= std::regex::icase;
11778             }
11779             auto reg = std::regex( m_regex, flags );
11780             return std::regex_match( matchee, reg );
11781         }
11782
11783         std::string RegexMatcher::describe() const {
11784             return "matches " + ::Catch::Detail::stringify( m_regex ) + ( ( m_caseSensitivity ==
11785 CaseSensitive::Choice::Yes ) ? " case sensitively" : " case insensitively" );
11786         }
11787     } // namespace StdString
11788
11789     StdString::EqualsMatcher Equals( std::string const& str, CaseSensitive::Choice caseSensitivity
11790 ) {
11791         return StdString::EqualsMatcher( StdString::CasedString( str, caseSensitivity ) );
11792     }
11793
11794     StdString::ContainsMatcher Contains( std::string const& str, CaseSensitive::Choice
11795 caseSensitivity ) {
11796         return StdString::ContainsMatcher( StdString::CasedString( str, caseSensitivity ) );
11797     }
11798
11799     StdString::EndsWithMatcher EndsWith( std::string const& str, CaseSensitive::Choice
11800 caseSensitivity ) {
11801         return StdString::EndsWithMatcher( StdString::CasedString( str, caseSensitivity ) );
11802     }
11803
11804     StdString::StartsWithMatcher StartsWith( std::string const& str, CaseSensitive::Choice
11805 caseSensitivity ) {
11806         return StdString::StartsWithMatcher( StdString::CasedString( str, caseSensitivity ) );
11807     }
11808
11809     StdString::RegexMatcher Matches( std::string const& regex, CaseSensitive::Choice
11810 caseSensitivity ) {
11811         return StdString::RegexMatcher( regex, caseSensitivity );
11812     }
11813 } // namespace Matchers
11814 } // namespace Catch
11815 // end catch_matchers_string.cpp
11816 // start catch_message.cpp

```

```

11805
11806 // start catch_uncaught_exceptions.h
11807
11808 namespace Catch {
11809     bool uncaught_exceptions();
11810 } // end namespace Catch
11811
11812 // end catch_uncaught_exceptions.h
11813 #include <cassert>
11814 #include <stack>
11815
11816 namespace Catch {
11817     MessageInfo::MessageInfo( StringRef const& _macroName,
11818                               SourceLineInfo const& _lineInfo,
11819                               ResultWas::OfType _type )
11820         : macroName( _macroName ),
11821           lineInfo( _lineInfo ),
11822           type( _type ),
11823           sequence( ++globalCount )
11824     {}
11825
11826     bool MessageInfo::operator==( MessageInfo const& other ) const {
11827         return sequence == other.sequence;
11828     }
11829
11830     bool MessageInfo::operator<( MessageInfo const& other ) const {
11831         return sequence < other.sequence;
11832     }
11833
11834     // This may need protecting if threading support is added
11835     unsigned int MessageInfo::globalCount = 0;
11836
11837
11838     Catch::MessageBuilder::MessageBuilder( StringRef const& macroName,
11839                                             SourceLineInfo const& lineInfo,
11840                                             ResultWas::OfType type )
11841         : m_info(macroName, lineInfo, type) {}
11842
11843
11844     ScopedMessage::ScopedMessage( MessageBuilder const& builder )
11845         : m_info( builder.m_info ), m_moved()
11846     {
11847         m_info.message = builder.m_stream.str();
11848         getResultCapture().pushScopedMessage( m_info );
11849     }
11850
11851     ScopedMessage::ScopedMessage( ScopedMessage&& old )
11852         : m_info( old.m_info ), m_moved()
11853     {
11854         old.m_moved = true;
11855     }
11856
11857     ScopedMessage::~ScopedMessage() {
11858         if ( !uncaught_exceptions() && !m_moved ) {
11859             getResultCapture().popScopedMessage(m_info);
11860         }
11861     }
11862
11863     Capturer::Capturer( StringRef macroName, SourceLineInfo const& lineInfo, ResultWas::OfType
11864 resultType, StringRef names ) {
11865         auto trimmed = [&](size_t start, size_t end) {
11866             while (names[start] == ',' || isspace(static_cast<unsigned char>(names[start]))) {
11867                 ++start;
11868             }
11869             while (names[end] == ',' || isspace(static_cast<unsigned char>(names[end]))) {
11870                 --end;
11871             }
11872             return names.substr(start, end - start + 1);
11873         };
11874         auto skipq = [&](size_t start, char quote) {
11875             for (auto i = start + 1; i < names.size(); ++i) {
11876                 if (names[i] == quote)
11877                     return i;
11878                 if (names[i] == '\\')
11879                     ++i;
11880             }
11881             CATCH_INTERNAL_ERROR("CAPTURE parsing encountered unmatched quote");
11882         };
11883         size_t start = 0;
11884         std::stack<char> openings;
11885         for (size_t pos = 0; pos < names.size(); ++pos) {
11886             char c = names[pos];
11887             switch (c) {
11888                 case '[':
11889                 case '{':

```

```

11893         case '(':
11894             // It is basically impossible to disambiguate between
11895             // comparison and start of template args in this context
11896         // case '<':
11897             openings.push(c);
11898             break;
11899         case ']':
11900         case ')':
11901         case ')':
11902         // case '>':
11903             openings.pop();
11904             break;
11905         case '"':
11906         case '\\':
11907             pos = skipq(pos, c);
11908             break;
11909         case ',':
11910             if (start != pos && openings.empty()) {
11911                 m_messages.emplace_back(macroName, lineInfo, resultType);
11912                 m_messages.back().message = static_cast<std::string>(trimmed(start, pos));
11913                 m_messages.back().message += " := ";
11914                 start = pos;
11915             }
11916     }
11917 }
11918 assert(openings.empty() && "Mismatched openings");
11919 m_messages.emplace_back(macroName, lineInfo, resultType);
11920 m_messages.back().message = static_cast<std::string>(trimmed(start, names.size() - 1));
11921 m_messages.back().message += " := ";
11922 }
11923 Capturer::~Capturer() {
11924     if ( !uncaught_exceptions() ){
11925         assert( m_captured == m_messages.size() );
11926         for( size_t i = 0; i < m_captured; ++i )
11927             m_resultCapture.popScopedMessage( m_messages[i] );
11928     }
11929 }
11930
11931 void Capturer::captureValue( size_t index, std::string const& value ) {
11932     assert( index < m_messages.size() );
11933     m_messages[index].message += value;
11934     m_resultCapture.pushScopedMessage( m_messages[index] );
11935     m_captured++;
11936 }
11937
11938 } // end namespace Catch
11939 // end catch_message.cpp
11940 // start catch_output_redirect.cpp
11941
11942 // start catch_output_redirect.h
11943 #ifndef TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
11944 #define TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
11945
11946 #include <cstdio>
11947 #include <iosfwd>
11948 #include <string>
11949
11950 namespace Catch {
11951
11952     class RedirectedStream {
11953     public:
11954         RedirectedStream( std::ostream& originalStream, std::ostream& redirectionStream );
11955         ~RedirectedStream();
11956     };
11957
11958     class RedirectedStdOut {
11959     public:
11960         RedirectedStdOut();
11961         auto str() const -> std::string;
11962     };
11963
11964     // StdErr has two constituent streams in C++, std::cerr and std::clog
11965     // This means that we need to redirect 2 streams into 1 to keep proper
11966     // order of writes
11967     class RedirectedStdErr {
11968     public:
11969         RedirectedStdErr();
11970         auto str() const -> std::string;
11971     };
11972
11973 }
11974
11975 #endif

```

```

11980     };
11981
11982     class RedirectedStreams {
11983     public:
11984         RedirectedStreams(RedirectedStreams const&) = delete;
11985         RedirectedStreams& operator=(RedirectedStreams const&) = delete;
11986         RedirectedStreams(RedirectedStreams&&) = delete;
11987         RedirectedStreams& operator=(RedirectedStreams&&) = delete;
11988
11989         RedirectedStreams(std::string& redirectedCout, std::string& redirectedCerr);
11990         ~RedirectedStreams();
11991     private:
11992         std::string& m_redirectedCout;
11993         std::string& m_redirectedCerr;
11994         RedirectedStdOut m_redirectedStdOut;
11995         RedirectedStdErr m_redirectedStdErr;
11996     };
11997
11998 #if defined(CATCH_CONFIG_NEW_CAPTURE)
11999
12000     // Windows's implementation of std::tmpfile is terrible (it tries
12001     // to create a file inside system folder, thus requiring elevated
12002     // privileges for the binary), so we have to use tmpnam(_s) and
12003     // create the file ourselves there.
12004     class TempFile {
12005     public:
12006         TempFile(TempFile const&) = delete;
12007         TempFile& operator=(TempFile const&) = delete;
12008         TempFile(TempFile&&) = delete;
12009         TempFile& operator=(TempFile&&) = delete;
12010
12011         TempFile();
12012         ~TempFile();
12013
12014         std::FILE* getFile();
12015         std::string getContents();
12016
12017     private:
12018         std::FILE* m_file = nullptr;
12019         #if defined(_MSC_VER)
12020             char m_buffer[L_tmpnam] = { 0 };
12021         #endif
12022     };
12023
12024     class OutputRedirect {
12025     public:
12026         OutputRedirect(OutputRedirect const&) = delete;
12027         OutputRedirect& operator=(OutputRedirect const&) = delete;
12028         OutputRedirect(OutputRedirect&&) = delete;
12029         OutputRedirect& operator=(OutputRedirect&&) = delete;
12030
12031         OutputRedirect(std::string& stdout_dest, std::string& stderr_dest);
12032         ~OutputRedirect();
12033
12034     private:
12035         int m_originalStdout = -1;
12036         int m_originalStderr = -1;
12037         TempFile m_stdoutFile;
12038         TempFile m_stderrFile;
12039         std::string& m_stdoutDest;
12040         std::string& m_stderrDest;
12041     };
12042
12043 #endif
12044
12045 } // end namespace Catch
12046
12047 #endif // TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
12048 // end catch_output_redirect.h
12049 #include <cstdio>
12050 #include <cstring>
12051 #include <fstream>
12052 #include <sstream>
12053 #include <stdexcept>
12054
12055 #if defined(CATCH_CONFIG_NEW_CAPTURE)
12056 #if defined(_MSC_VER)
12057     #include <io.h> // _dup and _dup2
12058     #define dup _dup
12059     #define dup2 _dup2
12060     #define fileno _fileno
12061     #else
12062     #include <unistd.h> // dup and dup2
12063     #endif
12064 #endif
12065
12066 namespace Catch {

```

```

12067
12068     RedirectedStream::RedirectedStream( std::ostream& originalStream, std::ostream& redirectionStream
12069 )
12070     :   m_originalStream( originalStream ),
12071         m_redirectionStream( redirectionStream ),
12072         m_prevBuf( m_originalStream.rdbuf() )
12073     {
12074         m_originalStream.rdbuf( m_redirectionStream.rdbuf() );
12075     }
12076
12077     RedirectedStream::~RedirectedStream() {
12078         m_originalStream.rdbuf( m_prevBuf );
12079     }
12080
12081     RedirectedStdOut::RedirectedStdOut() : m_cout( Catch::cout(), m_rss.get() ) {}
12082     auto RedirectedStdOut::str() const -> std::string { return m_rss.str(); }
12083
12084     RedirectedStdErr::RedirectedStdErr()
12085     :   m_cerr( Catch::cerr(), m_rss.get() ),
12086         m_clog( Catch::clog(), m_rss.get() )
12087     {}
12088     auto RedirectedStdErr::str() const -> std::string { return m_rss.str(); }
12089
12090     RedirectedStreams::RedirectedStreams(std::string& redirectedCout, std::string& redirectedCerr)
12091     :   m_redirectedCout(redirectedCout),
12092         m_redirectedCerr(redirectedCerr)
12093     {}
12094
12095     RedirectedStreams::~RedirectedStreams() {
12096         m_redirectedCout += m_redirectedStdOut.str();
12097         m_redirectedCerr += m_redirectedStdErr.str();
12098     }
12099
12100     #if defined(CATCH_CONFIG_NEW_CAPTURE)
12101     #if defined(_MSC_VER)
12102     TempFile::TempFile() {
12103         if (tmpnam_s(m_buffer)) {
12104             CATCH_RUNTIME_ERROR("Could not get a temp filename");
12105         }
12106         if (fopen_s(&m_file, m_buffer, "w+")) {
12107             char buffer[100];
12108             if (strerror_s(buffer, errno)) {
12109                 CATCH_RUNTIME_ERROR("Could not translate errno to a string");
12110             }
12111             CATCH_RUNTIME_ERROR("Could not open the temp file: '" « m_buffer « "' because: " «
12112                 buffer);
12113         }
12114     #else
12115     TempFile::TempFile() {
12116         m_file = std::tmpfile();
12117         if (!m_file) {
12118             CATCH_RUNTIME_ERROR("Could not create a temp file.");
12119         }
12120     }
12121     #endif
12122     #endif
12123
12124     TempFile::~TempFile() {
12125         // TBD: What to do about errors here?
12126         std::fclose(m_file);
12127         // We manually create the file on Windows only, on Linux
12128         // it will be autodeleted
12129     #if defined(_MSC_VER)
12130         std::remove(m_buffer);
12131     #endif
12132     }
12133
12134     FILE* TempFile::getFile() {
12135         return m_file;
12136     }
12137
12138     std::string TempFile::getContents() {
12139         std::stringstream sstr;
12140         char buffer[100] = {};
12141         std::rewind(m_file);
12142         while (std::fgets(buffer, sizeof(buffer), m_file)) {
12143             sstr « buffer;
12144         }
12145         return sstr.str();
12146     }
12147
12148     OutputRedirect::OutputRedirect(std::string& stdout_dest, std::string& stderr_dest) :
12149     m_originalStdout( dup(1) ),
12150     m_originalStderr( dup(2) ),
12151     m_stdoutDest( stdout_dest ),

```

```

12152         m_stderrDest(stderr_dest) {
12153             dup2(fileno(m_stdoutFile.getFile()), 1);
12154             dup2(fileno(m_stderrFile.getFile()), 2);
12155         }
12156
12157     OutputRedirect::~OutputRedirect() {
12158         Catch::cout() << std::flush;
12159         fflush(stdout);
12160         // Since we support overriding these streams, we flush cerr
12161         // even though std::cerr is unbuffered
12162         Catch::cerr() << std::flush;
12163         Catch::clog() << std::flush;
12164         fflush(stderr);
12165
12166         dup2(m_originalStdout, 1);
12167         dup2(m_originalStderr, 2);
12168
12169         m_stdoutDest += m_stdoutFile.getContents();
12170         m_stderrDest += m_stderrFile.getContents();
12171     }
12172
12173 #endif // CATCH_CONFIG_NEW_CAPTURE
12174
12175 } // namespace Catch
12176
12177 #if defined(CATCH_CONFIG_NEW_CAPTURE)
12178 #if defined(_MSC_VER)
12179     #undef dup
12180     #undef dup2
12181     #undef fileno
12182 #endif
12183 #endif
12184 // end catch_output_redirect.cpp
12185 // start catch_polyfills.cpp
12186
12187 #include <cmath>
12188
12189 namespace Catch {
12190
12191 #if !defined(CATCH_CONFIG_POLYFILL_ISNAN)
12192     bool isnan(float f) {
12193         return std::isnan(f);
12194     }
12195     bool isnan(double d) {
12196         return std::isnan(d);
12197     }
12198 #else
12199     // For now we only use this for embarcadero
12200     bool isnan(float f) {
12201         return std::_isnan(f);
12202     }
12203     bool isnan(double d) {
12204         return std::_isnan(d);
12205     }
12206 #endif
12207 } // end namespace Catch
12208 // end catch_polyfills.cpp
12209 // start catch_random_number_generator.cpp
12210
12211 namespace Catch {
12212     namespace {
12213
12214         #if defined(_MSC_VER)
12215             #pragma warning(push)
12216             #pragma warning(disable:4146) // we negate uint32 during the rotate
12217         #endif
12218
12219         // Safe rotr implementation thanks to John Regehr
12220         uint32_t rotate_right(uint32_t val, uint32_t count) {
12221             const uint32_t mask = 31;
12222             count &= mask;
12223             return (val >> count) | (val << (-count & mask));
12224         }
12225
12226         #if defined(_MSC_VER)
12227             #pragma warning(pop)
12228         #endif
12229     }
12230
12231     SimplePcg32::SimplePcg32(result_type seed_) {
12232         seed(seed_);
12233     }
12234
12235     void SimplePcg32::seed(result_type seed_) {
12236         m_state = 0;

```



```

12239         (*this)();
12240         m_state += seed_;
12241         (*this)();
12242     }
12243
12244     void SimplePcg32::discard(uint64_t skip) {
12245         // We could implement this to run in O(log n) steps, but this
12246         // should suffice for our use case.
12247         for (uint64_t s = 0; s < skip; ++s) {
12248             static_cast<void>((*this)());
12249         }
12250     }
12251
12252     SimplePcg32::result_type SimplePcg32::operator()() {
12253         // prepare the output value
12254         const uint32_t xorshifted = static_cast<uint32_t>(((m_state >> 18u) ^ m_state) >> 27u);
12255         const auto output = rotate_right(xorshifted, m_state >> 59u);
12256
12257         // advance state
12258         m_state = m_state * 6364136223846793005ULL + s_inc;
12259
12260         return output;
12261     }
12262
12263     bool operator==(SimplePcg32 const& lhs, SimplePcg32 const& rhs) {
12264         return lhs.m_state == rhs.m_state;
12265     }
12266
12267     bool operator!=(SimplePcg32 const& lhs, SimplePcg32 const& rhs) {
12268         return lhs.m_state != rhs.m_state;
12269     }
12270 }
12271 // end catch_random_number_generator.cpp
12272 // start catch_registry_hub.cpp
12273
12274 // start catch_test_case_registry_impl.h
12275
12276 #include <vector>
12277 #include <set>
12278 #include <algorithm>
12279 #include <ios>
12280
12281 namespace Catch {
12282
12283     class TestCase;
12284     struct IConfig;
12285
12286     std::vector<TestCase> sortTests( IConfig const& config, std::vector<TestCase> const&
unsortedTestCases );
12287
12288     bool isThrowSafe( TestCase const& testCase, IConfig const& config );
12289     bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config );
12290
12291     void enforceNoDuplicateTestCases( std::vector<TestCase> const& functions );
12292
12293     std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
testSpec, IConfig const& config );
12294     std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config );
12295
12296     class TestRegistry : public ITestRegistry {
12297     public:
12298         virtual ~TestRegistry() = default;
12299
12300         virtual void registerTest( TestCase const& testCase );
12301
12302         std::vector<TestCase> const& getAllTests() const override;
12303         std::vector<TestCase> const& getAllTestsSorted( IConfig const& config ) const override;
12304
12305     private:
12306         std::vector<TestCase> m_functions;
12307         mutable RunTests::InWhatOrder m_currentSortOrder = RunTests::InDeclarationOrder;
12308         mutable std::vector<TestCase> m_sortedFunctions;
12309         std::size_t m_unnamedCount = 0;
12310         std::ios_base::Init m_ostreamInit; // Forces cout/ cerr to be initialised
12311     };
12312
12313
12314     class TestInvokerAsFunction : public ITestInvoker {
12315     public:
12316         void(*m_testAsFunction)();
12317
12318         TestInvokerAsFunction( void(*testAsFunction)() ) noexcept;
12319
12320         void invoke() const override;
12321     };
12322
12323     std::string extractClassName( StringRef const& classOrQualifiedMethodName );
12324

```

```

12326
12327 } // end namespace Catch
12328
12329 // end catch_test_case_registry_impl.h
12330 // start catch_reporter_registry.h
12331
12332 #include <map>
12333
12334 namespace Catch {
12335     class ReporterRegistry : public IReporterRegistry {
12336     public:
12337         ~ReporterRegistry() override;
12338
12339         IStreamingReporterPtr create( std::string const& name, IConfigPtr const& config ) const
12340         override;
12341
12342         void registerReporter( std::string const& name, IReporterFactoryPtr const& factory );
12343         void registerListener( IReporterFactoryPtr const& factory );
12344
12345         FactoryMap const& getFactories() const override;
12346         Listeners const& getListeners() const override;
12347
12348     private:
12349         FactoryMap m_factories;
12350         Listeners m_listeners;
12351     };
12352 }
12353
12354 // end catch_reporter_registry.h
12355 // start catch_tag_alias_registry.h
12356 // start catch_tag_alias.h
12357
12358 #include <string>
12359
12360 namespace Catch {
12361     struct TagAlias {
12362         TagAlias( std::string const& _tag, SourceLineInfo _lineInfo );
12363
12364         std::string tag;
12365         SourceLineInfo lineInfo;
12366     };
12367 } // end namespace Catch
12368
12369 // end catch_tag_alias.h
12370 #include <map>
12371
12372 namespace Catch {
12373     class TagAliasRegistry : public ITagAliasRegistry {
12374     public:
12375         ~TagAliasRegistry() override;
12376         TagAlias const* find( std::string const& alias ) const override;
12377         std::string expandAliases( std::string const& unexpandedTestSpec ) const override;
12378         void add( std::string const& alias, std::string const& tag, SourceLineInfo const& lineInfo );
12379
12380     private:
12381         std::map<std::string, TagAlias> m_registry;
12382     };
12383 } // end namespace Catch
12384
12385 // end catch_tag_alias_registry.h
12386 // start catch_startup_exception_registry.h
12387
12388 #include <vector>
12389 #include <exception>
12390
12391 namespace Catch {
12392     class StartupExceptionRegistry {
12393     public:
12394         void add( std::exception_ptr const& exception ) noexcept;
12395         std::vector<std::exception_ptr> const& getExceptions() const noexcept;
12396     private:
12397         std::vector<std::exception_ptr> m_exceptions;
12398     };
12399 } // end namespace Catch
12400
12401 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
12402     public:
12403         void add( std::exception_ptr const& exception ) noexcept;
12404         std::vector<std::exception_ptr> const& getExceptions() const noexcept;
12405     private:
12406         std::vector<std::exception_ptr> m_exceptions;
12407 #endif
12408 };
12409
12410 } // end namespace Catch
12411
```

```

12412 // end catch_startup_exception_registry.h
12413 // start catch_singletons.hpp
12414
12415 namespace Catch {
12416
12417     struct ISingleton {
12418         virtual ~ISingleton();
12419     };
12420
12421     void addSingleton( ISingleton* singleton );
12422     void cleanupSingletons();
12423
12424     template<typename SingletonImplT, typename InterfaceT = SingletonImplT, typename MutableInterfaceT
= InterfaceT>
12425     class Singleton : SingletonImplT, public ISingleton {
12426
12427     public:
12428         static auto getInternal() -> Singleton* {
12429             static Singleton* s_instance = nullptr;
12430             if( !s_instance ) {
12431                 s_instance = new Singleton;
12432                 addSingleton( s_instance );
12433             }
12434             return s_instance;
12435         }
12436
12437         static auto get() -> InterfaceT const& {
12438             return *getInternal();
12439         }
12440         static auto getMutable() -> MutableInterfaceT& {
12441             return *getInternal();
12442         }
12443     };
12444
12445 } // namespace Catch
12446
12447 // end catch_singletons.hpp
12448 namespace Catch {
12449
12450     namespace {
12451
12452         class RegistryHub : public IRegistryHub, public IMutableRegistryHub,
12453                             private NonCopyable {
12454
12455     public: // IRegistryHub
12456         RegistryHub() = default;
12457         IReporterRegistry const& getReporterRegistry() const override {
12458             return m_reporterRegistry;
12459         }
12460         ITestCaseRegistry const& getTestCaseRegistry() const override {
12461             return m_testCaseRegistry;
12462         }
12463         IExceptionTranslatorRegistry const& getExceptionTranslatorRegistry() const override {
12464             return m_exceptionTranslatorRegistry;
12465         }
12466         ITagAliasRegistry const& getTagAliasRegistry() const override {
12467             return m_tagAliasRegistry;
12468         }
12469         StartupExceptionRegistry const& getStartupExceptionRegistry() const override {
12470             return m_exceptionRegistry;
12471         }
12472
12473     public: // IMutableRegistryHub
12474         void registerReporter( std::string const& name, IReporterFactoryPtr const& factory )
12475             override {
12476             m_reporterRegistry.registerReporter( name, factory );
12477         }
12478         void registerListener( IReporterFactoryPtr const& factory ) override {
12479             m_reporterRegistry.registerListener( factory );
12480         }
12481         void registerTest( TestCase const& testInfo ) override {
12482             m_testCaseRegistry.registerTest( testInfo );
12483         }
12484         void registerTranslator( const IExceptionTranslator* translator ) override {
12485             m_exceptionTranslatorRegistry.registerTranslator( translator );
12486         }
12487         void registerTagAlias( std::string const& alias, std::string const& tag, SourceLineInfo
12488             const& lineInfo ) override {
12489             m_tagAliasRegistry.add( alias, tag, lineInfo );
12490         }
12491         void registerStartupException() noexcept override {
12492             m_exceptionRegistry.add(std::current_exception());
12493         }
12494     };
12495
12496     #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
12497         CATCH_INTERNAL_ERROR("Attempted to register active exception under
12498         CATCH_CONFIG_DISABLE_EXCEPTIONS!");
12499     #endif
12500 }

```

```

12495         }
12496         IMutableEnumValuesRegistry& getMutableEnumValuesRegistry() override {
12497             return m_enumValuesRegistry;
12498         }
12499     private:
12500         TestRegistry m_testCaseRegistry;
12501         ReporterRegistry m_reporterRegistry;
12502         ExceptionTranslatorRegistry m_exceptionTranslatorRegistry;
12503         TagAliasRegistry m_tagAliasRegistry;
12504         StartupExceptionRegistry m_exceptionRegistry;
12505         Detail::EnumValuesRegistry m_enumValuesRegistry;
12506     };
12507 };
12508
12509 using RegistryHubSingleton = Singleton<RegistryHub, IRegistryHub, IMutableRegistryHub>;
12510
12511 IRegistryHub const& getRegistryHub() {
12512     return RegistryHubSingleton::get();
12513 }
12514 IMutableRegistryHub& getMutableRegistryHub() {
12515     return RegistryHubSingleton::getMutable();
12516 }
12517 void cleanUp() {
12518     cleanupSingletons();
12519     cleanUpContext();
12520 }
12521 std::string translateActiveException() {
12522     return getRegistryHub().getExceptionTranslatorRegistry().translateActiveException();
12523 }
12524 } // end namespace Catch
12525 // end catch_registry_hub.cpp
12526 // start catch_reporter_registry.cpp
12527 namespace Catch {
12528     ReporterRegistry::~ReporterRegistry() = default;
12529     IStreamingReporterPtr ReporterRegistry::create( std::string const& name, IConfigPtr const& config ) const {
12530         auto it = m_factories.find( name );
12531         if ( it == m_factories.end() )
12532             return nullptr;
12533         return it->second->create( ReporterConfig( config ) );
12534     }
12535     void ReporterRegistry::registerReporter( std::string const& name, IReporterFactoryPtr const& factory ) {
12536         m_factories.emplace(name, factory);
12537     }
12538     void ReporterRegistry::registerListener( IReporterFactoryPtr const& factory ) {
12539         m_listeners.push_back( factory );
12540     }
12541     IReporterRegistry::FactoryMap const& ReporterRegistry::getFactories() const {
12542         return m_factories;
12543     }
12544     IReporterRegistry::Listeners const& ReporterRegistry::getListeners() const {
12545         return m_listeners;
12546     }
12547 }
12548 // end catch_reporter_registry.cpp
12549 // start catch_result_type.cpp
12550 namespace Catch {
12551     boolisOk( ResultWas::OfType resultType ) {
12552         return ( resultType & ResultWas::FailureBit ) == 0;
12553     }
12554     boolisJustInfo( int flags ) {
12555         return flags == ResultWas::Info;
12556     }
12557     ResultDisposition::Flags operator | ( ResultDisposition::Flags lhs, ResultDisposition::Flags rhs ) {
12558         return static_cast<ResultDisposition::Flags>( static_cast<int>( lhs ) | static_cast<int>( rhs ) );
12559     }
12560     bool shouldContinueOnFailure( int flags ) { return ( flags & ResultDisposition::ContinueOnFailure ) != 0; }
12561     bool shouldSuppressFailure( int flags ) { return ( flags & ResultDisposition::SuppressFail ) != 0; }
12562 } // end namespace Catch

```

```

12576 // end catch_result_type.cpp
12577 // start catch_run_context.cpp
12578
12579 #include <cassert>
12580 #include <algorithm>
12581 #include <sstream>
12582
12583 namespace Catch {
12584
12585     namespace Generators {
12586         struct GeneratorTracker : TestCaseTracking::TrackerBase, IGeneratorTracker {
12587             GeneratorBasePtr m_generator;
12588
12589             GeneratorTracker( TestCaseTracking::NameAndLocation const& nameAndLocation,
12590                             TrackerContext& ctx, ITracker* parent )
12591                 : TrackerBase( nameAndLocation, ctx, parent )
12592             {}
12593             ~GeneratorTracker();
12594
12595             static GeneratorTracker& acquire( TrackerContext& ctx, TestCaseTracking::NameAndLocation
12596             const& nameAndLocation ) {
12597                 std::shared_ptr<GeneratorTracker> tracker;
12598
12599                 ITracker& currentTracker = ctx.currentTracker();
12600                 // Under specific circumstances, the generator we want
12601                 // to acquire is also the current tracker. If this is
12602                 // the case, we have to avoid looking through current
12603                 // tracker's children, and instead return the current
12604                 // tracker.
12605                 // A case where this check is important is e.g.
12606                 // for (int i = 0; i < 5; ++i) {
12607                 //     int n = GENERATE(1, 2);
12608                 // }
12609                 // without it, the code above creates 5 nested generators.
12610                 if (currentTracker.nameAndLocation() == nameAndLocation) {
12611                     auto thisTracker = currentTracker.parent().findChild(nameAndLocation);
12612                     assert(thisTracker);
12613                     assert(thisTracker->isGeneratorTracker());
12614                     tracker = std::static_pointer_cast<GeneratorTracker>(thisTracker);
12615                 } else if ( TestCaseTracking::ITrackerPtr childTracker = currentTracker.findChild(
12616                 nameAndLocation ) ) {
12617                     assert( childTracker );
12618                     assert( childTracker->isGeneratorTracker() );
12619                     tracker = std::static_pointer_cast<GeneratorTracker>( childTracker );
12620                 } else {
12621                     tracker = std::make_shared<GeneratorTracker>( nameAndLocation, ctx,
12622                     &currentTracker );
12623                     currentTracker.addChild( tracker );
12624                 }
12625
12626                 if( !tracker->isComplete() ) {
12627                     tracker->open();
12628                 }
12629
12630                 return *tracker;
12631             }
12632
12633             // TrackerBase interface
12634             bool isGeneratorTracker() const override { return true; }
12635             auto hasGenerator() const -> bool override {
12636                 return !!m_generator;
12637             }
12638             void close() override {
12639                 TrackerBase::close();
12640                 // If a generator has a child (it is followed by a section)
12641                 // and none of its children have started, then we must wait
12642                 // until later to start consuming its values.
12643                 // This catches cases where `GENERATE` is placed between two
12644                 // `SECTION`'s.
12645                 // **The check for m_children.empty cannot be removed**.
12646                 // doing so would break `GENERATE` `_not_` followed by `SECTION`'s.
12647                 const bool should_wait_for_child = [&]() {
12648                     // No children -> nobody to wait for
12649                     if ( m_children.empty() ) {
12650                         return false;
12651                     }
12652                     // If at least one child started executing, don't wait
12653                     if ( std::find_if(
12654                         m_children.begin(),
12655                         m_children.end(),
12656                         []( TestCaseTracking::ITrackerPtr tracker ) {
12657                             return tracker->hasStarted();
12658                         } ) != m_children.end() ) {
12659                         return false;
12660                     }
12661                     return true;
12662                 };
12663             }
12664         };
12665     }
12666 }

```

```

12659         // No children have started. We need to check if they _can_
12660         // start, and thus we should wait for them, or they cannot
12661         // start (due to filters), and we shouldn't wait for them
12662         auto* parent = m_parent;
12663         // This is safe: there is always at least one section
12664         // tracker in a test case tracking tree
12665         while ( !parent->isSectionTracker() ) {
12666             parent = &(amp; parent->parent() );
12667         }
12668         assert( parent &&
12669             "Missing root (test case) level section" );
12670
12671         auto const& parentSection =
12672             static_cast<SectionTracker&>( *parent );
12673         auto const& filters = parentSection.getFilters();
12674         // No filters -> no restrictions on running sections
12675         if ( filters.empty() ) {
12676             return true;
12677         }
12678
12679         for ( auto const& child : m_children ) {
12680             if ( child->isSectionTracker() &&
12681                 std::find( filters.begin(),
12682                     filters.end(),
12683                         static_cast<SectionTracker&>( *child )
12684                             .trimmedName() ) !=
12685                     filters.end() ) {
12686                 return true;
12687             }
12688         }
12689         return false;
12690     }();
12691
12692     // This check is a bit tricky, because m_generator->next()
12693     // has a side-effect, where it consumes generator's current
12694     // value, but we do not want to invoke the side-effect if
12695     // this generator is still waiting for any child to start.
12696     if ( should_wait_for_child ||
12697         ( m_runState == CompletedSuccessfully &&
12698           m_generator->next() ) ) {
12699         m_children.clear();
12700         m_runState = Executing;
12701     }
12702 }
12703
12704 // IGeneratorTracker interface
12705 auto getGenerator() const -> GeneratorBasePtr const& override {
12706     return m_generator;
12707 }
12708 void setGenerator( GeneratorBasePtr&& generator ) override {
12709     m_generator = std::move( generator );
12710 }
12711 };
12712 GeneratorTracker::~GeneratorTracker() {}
12713 }
12714
12715 RunContext::RunContext( IConfigPtr const& _config, IStreamingReporterPtr&& reporter)
12716 :   m_runInfo(_config->name()),
12717     m_context( getCurrentMutableContext() ),
12718     m_config(_config),
12719     m_reporter(std::move(reporter)),
12720     m_lastAssertionInfo( StringRef(), SourceLineInfo("",0), StringRef(),
12721         ResultDisposition::Normal ),
12722     m_includeSuccessfulResults( m_config->includeSuccessfulResults() ||
12723 m_reporter->getPreferences().shouldReportAllAssertions )
12724 {
12725     m_context.setRunner( this );
12726     m_context.setConfig( m_config );
12727     m_context.setResultCapture( this );
12728     m_reporter->testRunStarting( m_runInfo );
12729 }
12730
12731 RunContext::~RunContext() {
12732     m_reporter->testRunEnded( TestRunStats( m_runInfo, m_totals, aborting() ) );
12733 }
12734
12735 void RunContext::testGroupStarting( std::string const& testSpec, std::size_t groupIndex,
12736     std::size_t groupsCount ) {
12737     m_reporter->testGroupStarting( GroupInfo( testSpec, groupIndex, groupsCount ) );
12738 }
12739
12740 void RunContext::testGroupEnded( std::string const& testSpec, Totals const& totals, std::size_t
12741     groupIndex, std::size_t groupsCount ) {
12742     m_reporter->testGroupEnded( TestGroupStats( GroupInfo( testSpec, groupIndex, groupsCount ),
12743         totals, aborting() ) );
12744 }

```

```

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));
12759             runCurrentTest(redirectedCout, redirectedCerr);
12760             } while (!m_testCaseTracker->isSuccessfullyCompleted() && !aborting());
12761
12762             Totals deltaTotals = m_totals.delta(prevTotals);
12763             if (testInfo.expectedToFail() && deltaTotals.testCases.passed > 0) {
12764                 deltaTotals.assertions.failed++;
12765                 deltaTotals.testCases.passed--;
12766                 deltaTotals.testCases.failed++;
12767             }
12768             m_totals.testCases += deltaTotals.testCases;
12769             m_reporter->testCaseEnded(TestCaseStats(testInfo,
12770                 deltaTotals,
12771                 redirectedCout,
12772                 redirectedCerr,
12773                 aborting()));
12774
12775             m_activeTestCase = nullptr;
12776             m_testCaseTracker = nullptr;
12777
12778             return deltaTotals;
12779         }
12780
12781         IConfigPtr RunContext::config() const {
12782             return m_config;
12783         }
12784
12785         IStreamingReporter& RunContext::reporter() const {
12786             return *m_reporter;
12787         }
12788
12789         void RunContext::assertionEnded(AssertionResult const & result) {
12790             if (result.getResultType() == ResultWas::Ok) {
12791                 m_totals.assertions.passed++;
12792                 m_lastAssertionPassed = true;
12793             } else if (!result.isOk()) {
12794                 m_lastAssertionPassed = false;
12795                 if (m_activeTestCase->getTestCaseInfo().okToFail())
12796                     m_totals.assertions.failedButOk++;
12797                 else
12798                     m_totals.assertions.failed++;
12799             }
12800             else {
12801                 m_lastAssertionPassed = true;
12802             }
12803
12804             // We have no use for the return value (whether messages should be cleared), because messages
12805             // were made scoped
12806             // and should be let to clear themselves out.
12807             static_cast<void>(m_reporter->assertionEnded(AssertionStats(result, m_messages, m_totals)));
12808
12809             if (result.getResultType() != ResultWas::Warning)
12810                 m_messageScopes.clear();
12811
12812             // Reset working state
12813             resetAssertionInfo();
12814             m_lastResult = result;
12815         }
12816         void RunContext::resetAssertionInfo() {
12817             m_lastAssertionInfo.macroName = StringRef();
12818             m_lastAssertionInfo.capturedExpression = "{Unknown expression after the reported line}"_sr;
12819         }
12820         bool RunContext::sectionStarted(SectionInfo const & sectionInfo, Counts & assertions) {
12821             ITracker& sectionTracker = SectionTracker::acquire(m_trackerContext,
12822                 TestCaseTracking::NameAndLocation(sectionInfo.name, sectionInfo.lineInfo));
12823             if (!sectionTracker.isOpen())
12824                 return false;
12825             return true;
12826             m_activeSections.push_back(&sectionTracker);
12827         }

```

```

12825
12826     m_lastAssertionInfo.lineInfo = sectionInfo.lineInfo;
12827
12828     m_reporter->sectionStarting(sectionInfo);
12829
12830     assertions = m_totals.assertions;
12831
12832     return true;
12833 }
12834 auto RunContext::acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo
12835 ) -> IGeneratorTracker& {
12836     using namespace Generators;
12837     GeneratorTracker& tracker = GeneratorTracker::acquire(m_trackerContext,
12838                                                         TestCaseTracking::NameAndLocation(
12839         static_cast<std::string>(generatorName), lineInfo ) );
12838     m_lastAssertionInfo.lineInfo = lineInfo;
12839     return tracker;
12840 }
12841
12842 bool RunContext::testForMissingAssertions(Counts& assertions) {
12843     if (assertions.total() != 0)
12844         return false;
12845     if (!m_config->warnAboutMissingAssertions())
12846         return false;
12847     if (m_trackerContext.currentTracker().hasChildren())
12848         return false;
12849     m_totals.assertions.failed++;
12850     assertions.failed++;
12851     return true;
12852 }
12853
12854 void RunContext::sectionEnded(SectionEndInfo const & endInfo) {
12855     Counts assertions = m_totals.assertions - endInfo.prevAssertions;
12856     bool missingAssertions = testForMissingAssertions(assertions);
12857
12858     if (!m_activeSections.empty()) {
12859         m_activeSections.back()->close();
12860         m_activeSections.pop_back();
12861     }
12862
12863     m_reporter->sectionEnded(SectionStats(endInfo.sectionInfo, assertions,
12864     endInfo.durationInSeconds, missingAssertions));
12865     m_messages.clear();
12866     m_messageScopes.clear();
12867 }
12868
12869 void RunContext::sectionEndedEarly(SectionEndInfo const & endInfo) {
12870     if (m_unfinishedSections.empty())
12871         m_activeSections.back()->fail();
12872     else
12873         m_activeSections.back()->close();
12874     m_activeSections.pop_back();
12875     m_unfinishedSections.push_back(endInfo);
12876 }
12877
12878 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
12879 void RunContext::benchmarkPreparing(std::string const& name) {
12880     m_reporter->benchmarkPreparing(name);
12881 }
12882 void RunContext::benchmarkStarting( BenchmarkInfo const& info ) {
12883     m_reporter->benchmarkStarting( info );
12884 }
12885 void RunContext::benchmarkEnded( BenchmarkStats<> const& stats ) {
12886     m_reporter->benchmarkEnded( stats );
12887 }
12888 void RunContext::benchmarkFailed(std::string const& error) {
12889     m_reporter->benchmarkFailed(error);
12890 }
12891 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
12892
12893 void RunContext::pushScopedMessage(MessageInfo const& message) {
12894     m_messages.push_back(message);
12895 }
12896
12897 void RunContext::popScopedMessage(MessageInfo const& message) {
12898     m_messages.erase(std::remove(m_messages.begin(), m_messages.end(), message),
12899     m_messages.end());
12900 }
12901
12902 void RunContext::emplaceUnscopedMessage( MessageBuilder const& builder ) {
12903     m_messageScopes.emplace_back( builder );
12904 }
12905
12906 std::string RunContext::getCurrentTestName() const {
12907     return m_activeTestCase
12908         ? m_activeTestCase->getTestCaseInfo().name

```



```

12908         : std::string();
12909     }
12910
12911     const AssertionResult * RunContext::getLastResult() const {
12912         return &(*m_lastResult);
12913     }
12914
12915     void RunContext::exceptionEarlyReported() {
12916         m_shouldReportUnexpected = false;
12917     }
12918
12919     void RunContext::handleFatalErrorCondition( StringRef message ) {
12920         // First notify reporter that bad things happened
12921         m_reporter->fatalErrorEncountered(message);
12922
12923         // Don't rebuild the result -- the stringification itself can cause more fatal errors
12924         // Instead, fake a result data.
12925         AssertionResultData tempResult( ResultWas::FatalErrorCondition, { false } );
12926         tempResult.message = static_cast<std::string>(message);
12927         AssertionResult result(m_lastAssertionInfo, tempResult);
12928
12929         assertionEnded(result);
12930
12931         handleUnfinishedSections();
12932
12933         // Recreate section for test case (as we will lose the one that was in scope)
12934         auto const& testCaseInfo = m_activeTestCase->getTestCaseInfo();
12935         SectionInfo testCaseSection(testCaseInfo.lineInfo, testCaseInfo.name);
12936
12937         Counts assertions;
12938         assertions.failed = 1;
12939         SectionStats testCaseSectionStats(testCaseSection, assertions, 0, false);
12940         m_reporter->sectionEnded(testCaseSectionStats);
12941
12942         auto const& testInfo = m_activeTestCase->getTestCaseInfo();
12943
12944         Totals deltaTotals;
12945         deltaTotals.testCases.failed = 1;
12946         deltaTotals.assertions.failed = 1;
12947         m_reporter->testCaseEnded(TestCaseStats(testInfo,
12948                                             deltaTotals,
12949                                             std::string(),
12950                                             std::string(),
12951                                             false));
12952         m_totals.testCases.failed++;
12953         testGroupEnded(std::string(), m_totals, 1, 1);
12954         m_reporter->testRunEnded(TestRunStats(m_runInfo, m_totals, false));
12955     }
12956
12957     bool RunContext::lastAssertionPassed() {
12958         return m_lastAssertionPassed;
12959     }
12960
12961     void RunContext::assertionPassed() {
12962         m_lastAssertionPassed = true;
12963         ++m_totals.assertions.passed;
12964         resetAssertionInfo();
12965         m_messageScopes.clear();
12966     }
12967
12968     bool RunContext::aborting() const {
12969         return m_totals.assertions.failed >= static_cast<std::size_t>(m_config->abortAfter());
12970     }
12971
12972     void RunContext::runCurrentTest(std::string & redirectedCout, std::string & redirectedCerr) {
12973         auto const& testCaseInfo = m_activeTestCase->getTestCaseInfo();
12974         SectionInfo testCaseSection(testCaseInfo.lineInfo, testCaseInfo.name);
12975         m_reporter->sectionStarting(testCaseSection);
12976         Counts prevAssertions = m_totals.assertions;
12977         double duration = 0;
12978         m_shouldReportUnexpected = true;
12979         m_lastAssertionInfo = { "TEST_CASE"_sr, testCaseInfo.lineInfo, StringRef(),
ResultDisposition::Normal };
12980
12981         seedRng(*m_config);
12982
12983         Timer timer;
12984         CATCH_TRY {
12985             if (m_reporter->getPreferences().shouldRedirectStdOut) {
12986                 #if !defined(CATCH_CONFIG_EXPERIMENTAL_REDIRECT)
12987                 RedirectedStreams redirectedStreams(redirectedCout, redirectedCerr);
12988
12989                 timer.start();
12990                 invokeActiveTestCase();
12991             #else
12992                 OutputRedirect r(redirectedCout, redirectedCerr);
12993                 timer.start();

```

```

12994         invokeActiveTestCase();
12995 #endif
12996     } else {
12997         timer.start();
12998         invokeActiveTestCase();
12999     }
13000     duration = timer.getElapsedSeconds();
13001 } CATCH_CATCH_ANON (TestFailureException&) {
13002     // This just means the test was aborted due to failure
13003 } CATCH_CATCH_ALL {
13004     // Under CATCH_CONFIG_FAST_COMPILE, unexpected exceptions under REQUIRE assertions
13005     // are reported without translation at the point of origin.
13006     if( m_shouldReportUnexpected ) {
13007         AssertionReaction dummyReaction;
13008         handleUnexpectedInflightException( m_lastAssertionInfo, translateActiveException(),
dummyReaction );
13009     }
13010 }
13011 Counts assertions = m_totals.assertions - prevAssertions;
13012 bool missingAssertions = testForMissingAssertions(assertions);
13013
13014 m_testCaseTracker->close();
13015 handleUnfinishedSections();
13016 m_messages.clear();
13017 m_messageScopes.clear();
13018
13019 SectionStats testCaseSectionStats(testCaseSection, assertions, duration, missingAssertions);
13020 m_reporter->sectionEnded(testCaseSectionStats);
13021 }
13022
13023 void RunContext::invokeActiveTestCase() {
13024     FatalConditionHandlerGuard _(&m_fatalConditionhandler);
13025     m_activeTestCase->invoke();
13026 }
13027
13028 void RunContext::handleUnfinishedSections() {
13029     // If sections ended prematurely due to an exception we stored their
13030     // infos here so we can tear them down outside the unwind process.
13031     for (auto it = m_unfinishedSections.rbegin(),
13032          itEnd = m_unfinishedSections.rend();
13033          it != itEnd; ++it)
13034         sectionEnded(*it);
13035     m_unfinishedSections.clear();
13036 }
13037
13038 void RunContext::handleExpr(
13039     AssertionInfo const& info,
13040     ITransientExpression const& expr,
13041     AssertionReaction& reaction
13042 ) {
13043     m_reporter->assertionStarting( info );
13044
13045     bool negated = isFalseTest( info.resultDisposition );
13046     bool result = expr.getResult() != negated;
13047
13048     if( result ) {
13049         if (!m_includeSuccessfulResults) {
13050             assertionPassed();
13051         }
13052     }
13053     else {
13054         reportExpr(info, ResultWas::Ok, &expr, negated);
13055     }
13056 }
13057 else {
13058     reportExpr(info, ResultWas::ExpressionFailed, &expr, negated );
13059     populateReaction( reaction );
13060 }
13061 }
13062 void RunContext::reportExpr(
13063     AssertionInfo const& info,
13064     ResultWas::OfType resultType,
13065     ITransientExpression const* expr,
13066     bool negated ) {
13067     m_lastAssertionInfo = info;
13068     AssertionResultData data( resultType, LazyExpression( negated ) );
13069
13070     AssertionResult assertionResult( info, data );
13071     assertionResult.m_resultData.lazyExpression.m_transientExpression = expr;
13072
13073     assertionEnded( assertionResult );
13074 }
13075
13076 void RunContext::handleMessage(
13077     AssertionInfo const& info,
13078     ResultWas::OfType resultType,
13079 
```

```

13080        StringRef const& message,
13081         AssertionReaction& reaction
13082     ) {
13083         m_reporter->assertionStarting( info );
13084
13085         m_lastAssertionInfo = info;
13086
13087         AssertionResultData data( resultType, LazyExpression( false ) );
13088         data.message = static_cast<std::string>(message);
13089         AssertionResult assertionResult{ m_lastAssertionInfo, data };
13090         assertionEnded( assertionResult );
13091         if( !assertionResult.isOk() )
13092             populateReaction( reaction );
13093     }
13094     void RunContext::handleUnexpectedExceptionNotThrown(
13095         AssertionInfo const& info,
13096         AssertionReaction& reaction
13097     ) {
13098         handleNonExpr( info, Catch::ResultWas::DidntThrowException, reaction );
13099     }
13100
13101     void RunContext::handleUnexpectedInflightException(
13102         AssertionInfo const& info,
13103         std::string const& message,
13104         AssertionReaction& reaction
13105     ) {
13106         m_lastAssertionInfo = info;
13107
13108         AssertionResultData data( ResultWas::ThrewException, LazyExpression( false ) );
13109         data.message = message;
13110         AssertionResult assertionResult{ info, data };
13111         assertionEnded( assertionResult );
13112         populateReaction( reaction );
13113     }
13114
13115     void RunContext::populateReaction( AssertionReaction& reaction ) {
13116         reaction.shouldDebugBreak = m_config->shouldDebugBreak();
13117         reaction.shouldThrow = aborting() || (m_lastAssertionInfo.resultDisposition &
ResultDisposition::Normal);
13118     }
13119
13120     void RunContext::handleIncomplete(
13121         AssertionInfo const& info
13122     ) {
13123         m_lastAssertionInfo = info;
13124
13125         AssertionResultData data( ResultWas::ThrewException, LazyExpression( false ) );
13126         data.message = "Exception translation was disabled by CATCH_CONFIG_FAST_COMPILE";
13127         AssertionResult assertionResult{ info, data };
13128         assertionEnded( assertionResult );
13129     }
13130     void RunContext::handleNonExpr(
13131         AssertionInfo const& info,
13132         ResultWas::OfType resultType,
13133         AssertionReaction &reaction
13134     ) {
13135         m_lastAssertionInfo = info;
13136
13137         AssertionResultData data( resultType, LazyExpression( false ) );
13138         AssertionResult assertionResult{ info, data };
13139         assertionEnded( assertionResult );
13140
13141         if( !assertionResult.isOk() )
13142             populateReaction( reaction );
13143     }
13144
13145     IResultCapture& getResultCapture() {
13146         if (auto* capture = getCurrentContext().getResultCapture())
13147             return *capture;
13148         else
13149             CATCH_INTERNAL_ERROR("No result capture instance");
13150     }
13151
13152     void seedRng( IConfig const& config ) {
13153         if (config.rngSeed() != 0) {
13154             std::srand( config.rngSeed() );
13155             rng().seed( config.rngSeed() );
13156         }
13157     }
13158
13159     unsigned int rngSeed() {
13160         return getCurrentContext().getConfig()->rngSeed();
13161     }
13162 }
13163 }
13164 // end catch_run_context.cpp
13165 // start catch_section.cpp

```

```

13166
13167 namespace Catch {
13168
13169     Section::Section( SectionInfo const& info )
13170         :   m_info( info ),
13171             m_sectionIncluded( getResultCapture().sectionStarted( m_info, m_assertions ) )
13172     {
13173         m_timer.start();
13174     }
13175
13176     Section::~~Section() {
13177         if( m_sectionIncluded ) {
13178             SectionEndInfo endInfo{ m_info, m_assertions, m_timer.getElapsedSeconds() };
13179             if( uncaught_exceptions() )
13180                 getResultCapture().sectionEndedEarly( endInfo );
13181             else
13182                 getResultCapture().sectionEnded( endInfo );
13183         }
13184     }
13185
13186     // This indicates whether the section should be executed or not
13187     Section::operator bool() const {
13188         return m_sectionIncluded;
13189     }
13190 } // end namespace Catch
13191 // end catch_section.cpp
13192 // start catch_section_info.cpp
13193 namespace Catch {
13194
13195     SectionInfo::SectionInfo
13196         (   SourceLineInfo const& _lineInfo,
13197             std::string const& _name )
13198         :   name( _name ),
13199             lineInfo( _lineInfo )
13200     {}
13201
13202 } // end namespace Catch
13203 // end catch_section_info.cpp
13204 // start catch_session.cpp
13205 // start catch_session.h
13206 #include <memory>
13207 namespace Catch {
13208
13209     class Session : NonCopyable {
13210     public:
13211         Session();
13212         ~Session() override;
13213
13214         void showHelp() const;
13215         void libIdentify();
13216
13217         int applyCommandLine( int argc, char const * const * argv );
13218 #if defined(CATCH_CONFIG_WCHAR) && defined(WIN32) && defined(UNICODE)
13219         int applyCommandLine( int argc, wchar_t const * const * argv );
13220 #endif
13221
13222         void useConfigData( ConfigData const& configData );
13223
13224         template<typename CharT>
13225         int run(int argc, CharT const * const argv[]) {
13226             if (m_startupExceptions)
13227                 return 1;
13228             int returnCode = applyCommandLine(argc, argv);
13229             if (returnCode == 0)
13230                 returnCode = run();
13231             return returnCode;
13232         }
13233
13234         int run();
13235
13236         clara::Parser const& cli() const;
13237         void cli( clara::Parser const& newParser );
13238         ConfigData& configData();
13239         Config& config();
13240     private:
13241         int runInternal();
13242
13243         clara::Parser m_cli;
13244         ConfigData m_configData;
13245         std::shared_ptr<Config> m_config;
13246         bool m_startupExceptions = false;
13247     };
13248 }

```

```

13253     };
13254
13255 } // end namespace Catch
13256
13257 // end catch_session.h
13258 // start catch_version.h
13259
13260 #include <iosfwd>
13261
13262 namespace Catch {
13263
13264     // Versioning information
13265     struct Version {
13266         Version( Version const& ) = delete;
13267         Version& operator=( Version const& ) = delete;
13268         Version(      unsigned int _majorVersion,
13269                  unsigned int _minorVersion,
13270                  unsigned int _patchNumber,
13271                  char const * _branchName,
13272                  unsigned int _buildNumber );
13273
13274         unsigned int const majorVersion;
13275         unsigned int const minorVersion;
13276         unsigned int const patchNumber;
13277
13278         // buildNumber is only used if branchName is not null
13279         char const * const branchName;
13280         unsigned int const buildNumber;
13281
13282         friend std::ostream& operator < ( std::ostream& os, Version const& version );
13283     };
13284
13285     Version const& libraryVersion();
13286 }
13287
13288 // end catch_version.h
13289 #include <cstdlib>
13290 #include <iomanip>
13291 #include <set>
13292 #include <iterator>
13293
13294 namespace Catch {
13295
13296     namespace {
13297         const int MaxExitCode = 255;
13298
13299         IStreamingReporterPtr createReporter(std::string const& reporterName, IConfigPtr const&
13300 config) {
13301             auto reporter = Catch::getRegistryHub().getReporterRegistry().create(reporterName,
13302 config);
13303             CATCH_ENFORCE(reporter, "No reporter registered with name: '" << reporterName << "'");
13304             return reporter;
13305         }
13306
13307         IStreamingReporterPtr makeReporter(std::shared_ptr<Config> const& config) {
13308             if (Catch::getRegistryHub().getReporterRegistry().getListeners().empty()) {
13309                 return createReporter(config->getReporterName(), config);
13310             }
13311
13312             // On older platforms, returning std::unique_ptr<ListeningReporter>
13313             // when the return type is std::unique_ptr<IStreamingReporter>
13314             // doesn't compile without a std::move call. However, this causes
13315             // a warning on newer platforms. Thus, we have to work around
13316             // it a bit and downcast the pointer manually.
13317             auto ret = std::unique_ptr<IStreamingReporter>(new ListeningReporter);
13318             auto& multi = static_cast<ListeningReporter&*>(*ret);
13319             auto const& listeners = Catch::getRegistryHub().getReporterRegistry().getListeners();
13320             for (auto const& listener : listeners) {
13321                 multi.addListener(listener->create(Catch::ReporterConfig(config)));
13322             }
13323             multi.addReporter(createReporter(config->getReporterName(), config));
13324             return ret;
13325         }
13326
13327         class TestGroup {
13328         public:
13329             explicit TestGroup(std::shared_ptr<Config> const& config)
13330                 : m_config{config}
13331                 , m_context{config, makeReporter(config)}
13332             {
13333                 auto const& allTestCases = getAllTestCasesSorted(*m_config);
13334                 m_matches = m_config->testSpec().matchesByFilter(allTestCases, *m_config);
13335                 auto const& invalidArgs = m_config->testSpec().getInvalidArgs();
13336
13337                 if (m_matches.empty() && invalidArgs.empty()) {
13338                     for (auto const& test : allTestCases)

```

```

13338         if (!test.isHidden())
13339             m_tests.emplace(&test);
13340     } else {
13341         for (auto const& match : m_matches)
13342             m_tests.insert(match.tests.begin(), match.tests.end());
13343     }
13344 }
13345
13346 Totals execute() {
13347     auto const& invalidArgs = m_config->testSpec().getInvalidArgs();
13348     Totals totals;
13349     m_context.testGroupStarting(m_config->name(), 1, 1);
13350     for (auto const& testCase : m_tests) {
13351         if (!m_context.aborting())
13352             totals += m_context.runTest(*testCase);
13353         else
13354             m_context.reporter().skipTest(*testCase);
13355     }
13356
13357     for (auto const& match : m_matches) {
13358         if (match.tests.empty()) {
13359             m_context.reporter().noMatchingTestCases(match.name);
13360             totals.error = -1;
13361         }
13362     }
13363
13364     if (!invalidArgs.empty()) {
13365         for (auto const& invalidArg : invalidArgs)
13366             m_context.reporter().reportInvalidArguments(invalidArg);
13367     }
13368
13369     m_context.testGroupEnded(m_config->name(), totals, 1, 1);
13370     return totals;
13371 }
13372
13373 private:
13374     using Tests = std::set<TestCase const*>;
13375
13376     std::shared_ptr<Config> m_config;
13377     RunContext m_context;
13378     Tests m_tests;
13379     TestSpec::Matches m_matches;
13380 };
13381
13382 void applyFileNamesAsTags(Catch::IConfig const& config) {
13383     auto& tests = const_cast<std::vector<TestCase>&>(getAllTestCasesSorted(config));
13384     for (auto& testCase : tests) {
13385         auto tags = testCase.tags;
13386
13387         std::string filename = testCase.lineInfo.file;
13388         auto lastSlash = filename.find_last_of("\\/");
13389         if (lastSlash != std::string::npos) {
13390             filename.erase(0, lastSlash);
13391             filename[0] = '#';
13392         }
13393         else
13394         {
13395             filename.insert(0, "#");
13396         }
13397
13398         auto lastDot = filename.find_last_of('.');
13399         if (lastDot != std::string::npos) {
13400             filename.erase(lastDot);
13401         }
13402
13403         tags.push_back(std::move(filename));
13404         setTags(testCase, tags);
13405     }
13406 }
13407
13408 } // anon namespace
13409
13410 Session::Session() {
13411     static bool alreadyInstantiated = false;
13412     if (alreadyInstantiated) {
13413         CATCH_TRY { CATCH_INTERNAL_ERROR( "Only one instance of Catch::Session can ever be used"
13414 ); }
13415     }
13416     CATCH_CATCH_ALL { getMutableRegistryHub().registerStartupException(); }
13417
13418     // There cannot be exceptions at startup in no-exception mode.
13419 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
13420     const auto& exceptions = getRegistryHub().getStartupExceptionRegistry().getExceptions();
13421     if (!exceptions.empty()) {
13422         config();
13423         getCurrentMutableContext().setConfig(m_config);
13424     }

```

```

13424         m_startupExceptions = true;
13425         Colour colourGuard( Colour::Red );
13426         Catch::cerr() << "Errors occurred during startup!" << '\n';
13427         // iterate over all exceptions and notify user
13428         for ( const auto& ex_ptr : exceptions ) {
13429             try {
13430                 std::rethrow_exception(ex_ptr);
13431             } catch ( std::exception const& ex ) {
13432                 Catch::cerr() << Column( ex.what() ).indent(2) << '\n';
13433             }
13434         }
13435     }
13436 #endif
13437
13438     alreadyInstantiated = true;
13439     m_cli = makeCommandLineParser( m_configData );
13440 }
13441 Session::~Session() {
13442     Catch::cleanUp();
13443 }
13444
13445 void Session::showHelp() const {
13446     Catch::cout()
13447         << "\nCatch v" << libraryVersion() << "\n"
13448         << m_cli << std::endl
13449         << "For more detailed usage please see the project docs\n" << std::endl;
13450 }
13451 void Session::libIdentify() {
13452     Catch::cout()
13453         << std::left << std::setw(16) << "description: " << "A Catch2 test executable\n"
13454         << std::left << std::setw(16) << "category: " << "testframework\n"
13455         << std::left << std::setw(16) << "framework: " << "Catch Test\n"
13456         << std::left << std::setw(16) << "version: " << libraryVersion() << std::endl;
13457 }
13458
13459 int Session::applyCommandLine( int argc, char const * const * argv ) {
13460     if( m_startupExceptions )
13461         return 1;
13462
13463     auto result = m_cli.parse( clara::Args( argc, argv ) );
13464     if( !result ) {
13465         config();
13466         getCurrentMutableContext().setConfig(m_config);
13467         Catch::cerr()
13468             << Colour( Colour::Red )
13469             << "\nError(s) in input:\n"
13470             << Column( result.errorMessage() ).indent( 2 )
13471             << "\n\n";
13472         Catch::cerr() << "Run with -? for usage\n" << std::endl;
13473         return MaxExitCode;
13474     }
13475
13476     if( m_configData.showHelp )
13477         showHelp();
13478     if( m_configData.libIdentify )
13479         libIdentify();
13480     m_config.reset();
13481     return 0;
13482 }
13483
13484 #if defined(CATCH_CONFIG_WCHAR) && defined(_WIN32) && defined(UNICODE)
13485 int Session::applyCommandLine( int argc, wchar_t const * const * argv ) {
13486     char **utf8Argv = new char * [ argc ];
13487
13488     for ( int i = 0; i < argc; ++i ) {
13489         int bufSize = WideCharToMultiByte( CP_UTF8, 0, argv[i], -1, nullptr, 0, nullptr, nullptr );
13490     };
13491
13492     utf8Argv[ i ] = new char [ bufSize ];
13493
13494     WideCharToMultiByte( CP_UTF8, 0, argv[i], -1, utf8Argv[i], bufSize, nullptr, nullptr );
13495 }
13496
13497 int returnCode = applyCommandLine( argc, utf8Argv );
13498
13499 for ( int i = 0; i < argc; ++i )
13500     delete [] utf8Argv[ i ];
13501
13502 delete [] utf8Argv;
13503
13504 return returnCode;
13505 }
13506 #endif
13507
13508 void Session::useConfigData( ConfigData const& configData ) {
13509     m_configData = configData;

```

```

13510         m_config.reset();
13511     }
13512
13513     int Session::run() {
13514         if( ( m_configData.waitForKeypress & WaitForKeypress::BeforeStart ) != 0 ) {
13515             Catch::cout() << "...waiting for enter/ return before starting" << std::endl;
13516             static_cast<void>(std::getchar());
13517         }
13518         int exitCode = runInternal();
13519         if( ( m_configData.waitForKeypress & WaitForKeypress::BeforeExit ) != 0 ) {
13520             Catch::cout() << "...waiting for enter/ return before exiting, with code: " << exitCode <<
std::endl;
13521             static_cast<void>(std::getchar());
13522         }
13523         return exitCode;
13524     }
13525
13526     clara::Parser const& Session::cli() const {
13527         return m_cli;
13528     }
13529     void Session::cli( clara::Parser const& newParser ) {
13530         m_cli = newParser;
13531     }
13532     ConfigData& Session::configData() {
13533         return m_configData;
13534     }
13535     Config& Session::config() {
13536         if( !m_config )
13537             m_config = std::make_shared<Config>( m_configData );
13538         return *m_config;
13539     }
13540
13541     int Session::runInternal() {
13542         if( m_startupExceptions )
13543             return 1;
13544
13545         if( m_configData.showHelp || m_configData.libIdentify ) {
13546             return 0;
13547         }
13548
13549         CATCH_TRY {
13550             config(); // Force config to be constructed
13551
13552             seedRng( *m_config );
13553
13554             if( m_configData.filenameAsTags )
13555                 applyFilenameAsTags( *m_config );
13556
13557             // Handle list request
13558             if( Option<std::size_t> listed = list( m_config ) )
13559                 return (std::min)(MaxExitCode, static_cast<int>(*listed));
13560
13561             TestGroup tests( m_config );
13562             auto const totals = tests.execute();
13563
13564             if( m_config->warnAboutNoTests() && totals.error == -1 )
13565                 return 2;
13566
13567             // Note that on unices only the lower 8 bits are usually used, clamping
13568             // the return value to 255 prevents false negative when some multiple
13569             // of 256 tests has failed
13570             return (std::min)(MaxExitCode, (std::max)(totals.error,
static_cast<int>(totals.assertions.failed)));
13571         }
13572         #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
13573         catch( std::exception& ex ) {
13574             Catch::cerr() << ex.what() << std::endl;
13575             return MaxExitCode;
13576         }
13577     #endif
13578     }
13579
13580 } // end namespace Catch
13581 // end catch_session.cpp
13582 // start catch_singletons.cpp
13583
13584 #include <vector>
13585
13586 namespace Catch {
13587
13588     namespace {
13589         static autogetSingletons() -> std::vector<ISingleton*>*& {
13590             static std::vector<ISingleton*> g_singletons = nullptr;
13591             if( !g_singletons )
13592                 g_singletons = new std::vector<ISingleton*>();
13593             return g_singletons;
13594         }
13595     }

```



```

13595     }
13596
13597     ISingleton::~ISingleton() {}
13598
13599     void addSingleton(ISingleton* singleton ) {
13600         getSingletons()->push_back( singleton );
13601     }
13602     void cleanupSingletons() {
13603         auto& singletons = getSingletons();
13604         for( auto singleton : *singletons )
13605             delete singleton;
13606         delete singletons;
13607         singletons = nullptr;
13608     }
13609
13610 } // namespace Catch
13611 // end catch_singletons.cpp
13612 // start catch_startup_exception_registry.cpp
13613
13614 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
13615 namespace Catch {
13616     void StartupExceptionRegistry::add( std::exception_ptr const& exception ) noexcept {
13617         CATCH_TRY {
13618             m_exceptions.push_back(exception);
13619         } CATCH_CATCH_ALL {
13620             // If we run out of memory during start-up there's really not a lot more we can do about
13621             it
13622             std::terminate();
13623         }
13624     }
13625     std::vector<std::exception_ptr> const& StartupExceptionRegistry::getExceptions() const noexcept {
13626         return m_exceptions;
13627     }
13628 }
13629 } // end namespace Catch
13630 #endif
13631 // end catch_startup_exception_registry.cpp
13632 // start catch_stream.cpp
13633
13634 #include <cstdio>
13635 #include <iostream>
13636 #include <fstream>
13637 #include <sstream>
13638 #include <vector>
13639 #include <memory>
13640
13641 namespace Catch {
13642
13643     Catch::IStream::~IStream() = default;
13644
13645     namespace Detail { namespace {
13646         template<typename WriterF, std::size_t bufferSize=256>
13647         class StreamBufImpl : public std::streambuf {
13648             char data[bufferSize];
13649             WriterF m_writer;
13650
13651         public:
13652             StreamBufImpl() {
13653                 setp( data, data + sizeof(data) );
13654             }
13655
13656             ~StreamBufImpl() noexcept {
13657                 StreamBufImpl::sync();
13658             }
13659
13660         private:
13661             int overflow( int c ) override {
13662                 sync();
13663
13664                 if( c != EOF ) {
13665                     if( pbase() == epptr() )
13666                         m_writer( std::string( 1, static_cast<char>( c ) ) );
13667                     else
13668                         sputc( static_cast<char>( c ) );
13669                 }
13670                 return 0;
13671             }
13672
13673             int sync() override {
13674                 if( pbase() != pptr() ) {
13675                     m_writer( std::string( pbase(), static_cast<std::string::size_type>( pptr() -
13676 pbase() ) ) );
13677                     setp( pbase(), pptr() );
13678                 }
13679                 return 0;
13680             }
13681         };
13682     } }

```

```

13680         };
13681
13682
13683
13684     struct OutputDebugWriter {
13685
13686         void operator()( std::string const&str ) {
13687             writeToDebugConsole( str );
13688         }
13689     };
13690
13691
13692
13693     class FileStream : public IStream {
13694     public:
13695         mutable std::ofstream m_ofs;
13696         FileStream( StringRef filename ) {
13697             m_ofs.open( filename.c_str() );
13698             CATCH_ENFORCE( !m_ofs.fail(), "Unable to open file: '" « filename « "' );
13699         }
13700         ~FileStream() override = default;
13701     public: // IStream
13702         std::ostream& stream() const override {
13703             return m_ofs;
13704         }
13705     };
13706
13707
13708
13709     class CoutStream : public IStream {
13710     public:
13711         mutable std::ostream m_os;
13712         // Store the streambuf from cout up-front because
13713         // cout may get redirected when running tests
13714         CoutStream() : m_os( Catch::cout().rdbuf() ) {}
13715         ~CoutStream() override = default;
13716
13717     public: // IStream
13718         std::ostream& stream() const override { return m_os; }
13719     };
13720
13721
13722
13723     class DebugOutStream : public IStream {
13724     public:
13725         mutable std::ostream m_os;
13726         DebugOutStream()
13727             : m_streamBuf( new StreamBufImpl<OutputDebugWriter>() ),
13728               m_os( m_streamBuf.get() )
13729         {}
13730         ~DebugOutStream() override = default;
13731     public: // IStream
13732         std::ostream& stream() const override { return m_os; }
13733     };
13734
13735     } // namespace anon::detail
13736
13737
13738
13739
13740
13741
13742     auto makeStream( StringRef const &filename ) -> IStream const* {
13743         if( filename.empty() )
13744             return new Detail::CoutStream();
13745         else if( filename[0] == '%' ) {
13746             if( filename == "%debug" )
13747                 return new Detail::DebugOutStream();
13748             else
13749                 CATCH_ERROR( "Unrecognised stream: '" « filename « "' );
13750         }
13751         else
13752             return new Detail::FileStream( filename );
13753     }
13754
13755     // This class encapsulates the idea of a pool of ostringstreams that can be reused.
13756     struct StringStreams {
13757     public:
13758         std::vector<std::unique_ptr<std::ostringstream>> m_streams;
13759         std::vector<std::size_t> m_unused;
13760         std::ostringstream m_referenceStream; // Used for copy state/ flags from
13761
13762         auto add() -> std::size_t {
13763             if( m_unused.empty() ) {
13764                 m_streams.push_back( std::unique_ptr<std::ostringstream>( new std::ostringstream ) );
13765                 return m_streams.size()-1;
13766             }
13767             else {
13768                 auto index = m_unused.back();
13769                 m_unused.pop_back();
13770                 return index;
13771             }
13772         }
13773     };

```

```

13772
13773     void release( std::size_t index ) {
13774         m_streams[index]->copyfmt( m_referenceStream ); // Restore initial flags and other state
13775         m_unused.push_back(index);
13776     }
13777 };
13778
13779 ReusableStringStream::ReusableStringStream()
13780 :     m_index( Singleton<StringStreams>::getMutable().add() ),
13781     m_oss( Singleton<StringStreams>::getMutable().m_streams[m_index].get() )
13782 {}
13783
13784 ReusableStringStream::~ReusableStringStream() {
13785     static_cast<std::ostream*>( m_oss )->str("");
13786     m_oss->clear();
13787     Singleton<StringStreams>::getMutable().release( m_index );
13788 }
13789
13790 auto ReusableStringStream::str() const -> std::string {
13791     return static_cast<std::ostream*>( m_oss )->str();
13792 }
13793
13794
13795
13796 #ifndef CATCH_CONFIG_NOSTDOUT // If you #define this you must implement these functions
13797     std::ostream& cout() { return std::cout; }
13798     std::ostream& cerr() { return std::cerr; }
13799     std::ostream& clog() { return std::clog; }
13800 #endif
13801 }
13802 // end catch_stream.cpp
13803 // start catch_string_manip.cpp
13804
13805 #include <algorithm>
13806 #include <ostream>
13807 #include <cstring>
13808 #include <cctype>
13809 #include <vector>
13810
13811 namespace Catch {
13812
13813     namespace {
13814         char toLowerCh(char c) {
13815             return static_cast<char>( std::tolower( static_cast<unsigned char>(c) ) );
13816         }
13817     }
13818
13819     bool startsWith( std::string const& s, std::string const& prefix ) {
13820         return s.size() >= prefix.size() && std::equal(prefix.begin(), prefix.end(), s.begin());
13821     }
13822     bool startsWith( std::string const& s, char prefix ) {
13823         return !s.empty() && s[0] == prefix;
13824     }
13825     bool endsWith( std::string const& s, std::string const& suffix ) {
13826         return s.size() >= suffix.size() && std::equal(suffix.rbegin(), suffix.rend(), s.rbegin());
13827     }
13828     bool endsWith( std::string const& s, char suffix ) {
13829         return !s.empty() && s[s.size()-1] == suffix;
13830     }
13831     bool contains( std::string const& s, std::string const& infix ) {
13832         return s.find( infix ) != std::string::npos;
13833     }
13834     void toLowerInPlace( std::string& s ) {
13835         std::transform( s.begin(), s.end(), s.begin(), toLowerCh );
13836     }
13837     std::string toLower( std::string const& s ) {
13838         std::string lc = s;
13839         toLowerInPlace( lc );
13840         return lc;
13841     }
13842     std::string trim( std::string const& str ) {
13843         static char const* whitespaceChars = "\n\r\t ";
13844         std::string::size_type start = str.find_first_not_of( whitespaceChars );
13845         std::string::size_type end = str.find_last_not_of( whitespaceChars );
13846
13847         return start != std::string::npos ? str.substr( start, 1+end-start ) : std::string();
13848     }
13849
13850     StringRef trim(StringRef ref) {
13851         const auto is_ws = [](char c) {
13852             return c == ' ' || c == '\t' || c == '\n' || c == '\r';
13853         };
13854         size_t real_begin = 0;
13855         while (real_begin < ref.size() && is_ws(ref[real_begin])) { ++real_begin; }
13856         size_t real_end = ref.size();
13857         while (real_end > real_begin && is_ws(ref[real_end - 1])) { --real_end; }
13858
13859         return ref.substr(real_begin, real_end - real_begin);

```

```

13860     }
13861
13862     bool replaceInPlace( std::string& str, std::string const& replaceThis, std::string const& withThis
    ) {
13863         bool replaced = false;
13864         std::size_t i = str.find( replaceThis );
13865         while( i != std::string::npos ) {
13866             replaced = true;
13867             str = str.substr( 0, i ) + withThis + str.substr( i+replaceThis.size() );
13868             if( i < str.size()-withThis.size() )
13869                 i = str.find( replaceThis, i+withThis.size() );
13870             else
13871                 i = std::string::npos;
13872         }
13873         return replaced;
13874     }
13875
13876     std::vector<StringRef> splitStringRef( StringRef str, char delimiter ) {
13877         std::vector<StringRef> subStrings;
13878         std::size_t start = 0;
13879         for( std::size_t pos = 0; pos < str.size(); ++pos ) {
13880             if( str[pos] == delimiter ) {
13881                 if( pos - start > 1 )
13882                     subStrings.push_back( str.substr( start, pos-start ) );
13883                 start = pos+1;
13884             }
13885         }
13886         if( start < str.size() )
13887             subStrings.push_back( str.substr( start, str.size()-start ) );
13888         return subStrings;
13889     }
13890
13891     pluralise::pluralise( std::size_t count, std::string const& label )
13892         : m_count( count ),
13893           m_label( label )
13894     {}
13895
13896     std::ostream& operator << ( std::ostream& os, pluralise const& pluraliser ) {
13897         os << pluraliser.m_count << ' ' << pluraliser.m_label;
13898         if( pluraliser.m_count != 1 )
13899             os << 's';
13900         return os;
13901     }
13902 }
13903
13904 // end catch_string_manip.cpp
13905 // start catch_stringref.cpp
13906
13907 #include <algorithm>
13908 #include <ostream>
13909 #include <cstring>
13910 #include <cstdint>
13911
13912 namespace Catch {
13913     StringRef::StringRef( char const* rawChars ) noexcept
13914         : StringRef( rawChars, static_cast<StringRef::size_type>(std::strlen(rawChars)) )
13915     {}
13916
13917     auto StringRef::c_str() const -> char const* {
13918         CATCH_ENFORCE( !IsNullTerminated(), "Called StringRef::c_str() on a non-null-terminated
instance" );
13919         return m_start;
13920     }
13921     auto StringRef::data() const noexcept -> char const* {
13922         return m_start;
13923     }
13924
13925     auto StringRef::substr( size_type start, size_type size ) const noexcept -> StringRef {
13926         if (start < m_size) {
13927             return StringRef(m_start + start, (std::min)(m_size - start, size));
13928         } else {
13929             return StringRef();
13930         }
13931     }
13932     auto StringRef::operator == ( StringRef const& other ) const noexcept -> bool {
13933         return m_size == other.m_size
            && (std::memcmp( m_start, other.m_start, m_size ) == 0);
13934     }
13935
13936     auto operator << ( std::ostream& os, StringRef const& str ) -> std::ostream& {
13937         return os.write(str.data(), str.size());
13938     }
13939
13940     auto operator+=( std::string& lhs, StringRef const& rhs ) -> std::string& {
13941         lhs.append(rhs.data(), rhs.size());
13942         return lhs;
13943     }
13944 }

```

```

13945
13946 } // namespace Catch
13947 // end catch_stringref.cpp
13948 // start catch_tag_alias.cpp
13949
13950 namespace Catch {
13951     TagAlias::TagAlias(std::string const& _tag, SourceLineInfo _lineInfo): tag(_tag),
13952     lineInfo(_lineInfo) {}
13953 }
13954 // end catch_tag_alias.cpp
13955 // start catch_tag_alias_autoregistrar.cpp
13956 namespace Catch {
13957
13958     RegistrarForTagAliases::RegistrarForTagAliases(char const* alias, char const* tag, SourceLineInfo
13959     const& lineInfo) {
13960         CATCH_TRY {
13961             getMutableRegistryHub().registerTagAlias(alias, tag, lineInfo);
13962         } CATCH_CATCH_ALL {
13963             // Do not throw when constructing global objects, instead register the exception to be
13964             // processed later
13965             getMutableRegistryHub().registerStartupException();
13966         }
13967     }
13968 } // end catch_tag_alias_autoregistrar.cpp
13969 // start catch_tag_alias_registry.cpp
13970
13971 #include <sstream>
13972
13973 namespace Catch {
13974
13975     TagAliasRegistry::~TagAliasRegistry() {}
13976
13977     TagAlias const* TagAliasRegistry::find( std::string const& alias ) const {
13978         auto it = m_registry.find( alias );
13979         if( it != m_registry.end() )
13980             return &(it->second);
13981         else
13982             return nullptr;
13983     }
13984
13985     std::string TagAliasRegistry::expandAliases( std::string const& unexpandedTestSpec ) const {
13986         std::string expandedTestSpec = unexpandedTestSpec;
13987         for( auto const& registryKvp : m_registry ) {
13988             std::size_t pos = expandedTestSpec.find( registryKvp.first );
13989             if( pos != std::string::npos ) {
13990                 expandedTestSpec = expandedTestSpec.substr( 0, pos ) +
13991                                     registryKvp.second.tag +
13992                                     expandedTestSpec.substr( pos + registryKvp.first.size() );
13993             }
13994         }
13995         return expandedTestSpec;
13996     }
13997
13998     void TagAliasRegistry::add( std::string const& alias, std::string const& tag, SourceLineInfo
13999     const& lineInfo ) {
14000         CATCH_ENFORCE( startsWith(alias, "[@]") && endsWith(alias, ']'),
14001             "error: tag alias, '" < alias < "' is not of the form [@alias name].\n" <
14002             lineInfo );
14003
14004         CATCH_ENFORCE( m_registry.insert(std::make_pair(alias, TagAlias(tag, lineInfo))).second,
14005             "error: tag alias, '" < alias < "' already registered.\n"
14006             < "\tFirst seen at: " < find(alias)->lineInfo < "\n"
14007             < "\tRedefined at: " < lineInfo );
14008     }
14009
14010     ITagAliasRegistry::~ITagAliasRegistry() {}
14011
14012     ITagAliasRegistry const* ITagAliasRegistry::get() {
14013         return getRegistryHub().getTagAliasRegistry();
14014     }
14015 } // end namespace Catch
14016 // end catch_tag_alias_registry.cpp
14017 // start catch_test_case_info.cpp
14018 #include <cctype>
14019 #include <exception>
14020 #include <algorithm>
14021 #include <sstream>
14022
14023 namespace Catch {
14024
14025     namespace {
14026         TestCaseInfo::SpecialProperties parseSpecialTag( std::string const& tag ) {

```

```

14027         if( startsWith( tag, '.' ) ||
14028             tag == "!hide" )
14029             return TestCaseInfo::IsHidden;
14030         else if( tag == "!throws" )
14031             return TestCaseInfo::Throws;
14032         else if( tag == "!shouldfail" )
14033             return TestCaseInfo::ShouldFail;
14034         else if( tag == "!mayfail" )
14035             return TestCaseInfo::MayFail;
14036         else if( tag == "!nonportable" )
14037             return TestCaseInfo::NonPortable;
14038         else if( tag == "!benchmark" )
14039             return static_cast<TestCaseInfo::SpecialProperties>( TestCaseInfo::Benchmark |
TestCaseInfo::IsHidden );
14040         else
14041             return TestCaseInfo::None;
14042     }
14043     bool isReservedTag( std::string const& tag ) {
14044         return parseSpecialTag( tag ) == TestCaseInfo::None && tag.size() > 0 && !std::isalnum(
static_cast<unsigned char>(tag[0]) );
14045     }
14046     void enforceNotReservedTag( std::string const& tag, SourceLineInfo const& _lineInfo ) {
14047         CATCH_ENFORCE( !isReservedTag(tag),
14048             "Tag name: [ " < tag < " ] is not allowed.\n"
14049             << "Tag names starting with non alphanumeric characters are
reserved\n"
14050             << _lineInfo );
14051     }
14052 }
14053
14054 TestCase makeTestCase( ITestInvoker* _testCase,
14055     std::string const& _className,
14056     NameAndTags const& nameAndTags,
14057     SourceLineInfo const& _lineInfo )
14058 {
14059     bool isHidden = false;
14060
14061     // Parse out tags
14062     std::vector<std::string> tags;
14063     std::string desc, tag;
14064     bool inTag = false;
14065     for (char c : nameAndTags.tags) {
14066         if( !inTag ) {
14067             if( c == '[' )
14068                 inTag = true;
14069             else
14070                 desc += c;
14071         }
14072         else {
14073             if( c == ']' ) {
14074                 TestCaseInfo::SpecialProperties prop = parseSpecialTag( tag );
14075                 if( ( prop & TestCaseInfo::IsHidden ) != 0 )
14076                     isHidden = true;
14077                 else if( prop == TestCaseInfo::None )
14078                     enforceNotReservedTag( tag, _lineInfo );
14079
14080                 // Merged hide tags like `[.approvals]` should be added as
14081                 // `[.]approvals`. The `[.]` is added at later point, so
14082                 // we only strip the prefix
14083                 if (startsWith(tag, '.') && tag.size() > 1) {
14084                     tag.erase(0, 1);
14085                 }
14086                 tags.push_back( tag );
14087                 tag.clear();
14088                 inTag = false;
14089             }
14090             else
14091                 tag += c;
14092         }
14093     }
14094     if( isHidden ) {
14095         // Add all "hidden" tags to make them behave identically
14096         tags.insert( tags.end(), { ".", "!hide" } );
14097     }
14098     TestCaseInfo info( static_cast<std::string>(nameAndTags.name), _className, desc, tags,
_lineInfo );
14099     return TestCase( _testCase, std::move(info) );
14100 }
14101
14102 void setTags( TestCaseInfo& testCaseInfo, std::vector<std::string> tags ) {
14103     std::sort(begin(tags), end(tags));
14104     tags.erase(std::unique(begin(tags), end(tags)), end(tags));
14105     testCaseInfo.lcaseTags.clear();
14106
14107     for( auto const& tag : tags ) {
14108         std::string lcaseTag = toLower( tag );
14109     }

```

```

14110         testCaseInfo.properties = static_cast<TestCaseInfo::SpecialProperties>(
testCaseInfo.properties | parseSpecialTag( lcaseTag ) );
14111         testCaseInfo.lcaseTags.push_back( lcaseTag );
14112     }
14113     testCaseInfo.tags = std::move(tags);
14114 }
14115
14116     TestCaseInfo::TestCaseInfo( std::string const& _name,
14117                                std::string const& _className,
14118                                std::string const& _description,
14119                                std::vector<std::string> const& _tags,
14120                                SourceLineInfo const& _lineInfo )
14121     :   name( _name ),
14122         className( _className ),
14123         description( _description ),
14124         lineInfo( _lineInfo ),
14125         properties( None )
14126     {
14127         setTags( *this, _tags );
14128     }
14129
14130     bool TestCaseInfo::isHidden() const {
14131         return ( properties & IsHidden ) != 0;
14132     }
14133     bool TestCaseInfo::throws() const {
14134         return ( properties & Throws ) != 0;
14135     }
14136     bool TestCaseInfo::okToFail() const {
14137         return ( properties & (ShouldFail | MayFail) ) != 0;
14138     }
14139     bool TestCaseInfo::expectedToFail() const {
14140         return ( properties & (ShouldFail) ) != 0;
14141     }
14142
14143     std::string TestCaseInfo::tagsAsString() const {
14144         std::string ret;
14145         // '[' and ']' per tag
14146         std::size_t full_size = 2 * tags.size();
14147         for (const auto& tag : tags) {
14148             full_size += tag.size();
14149         }
14150         ret.reserve(full_size);
14151         for (const auto& tag : tags) {
14152             ret.push_back('[');
14153             ret.append(tag);
14154             ret.push_back(']');
14155         }
14156
14157         return ret;
14158     }
14159
14160     TestCase::TestCase( ITestInvoker* testCase, TestCaseInfo&& info ) : TestCaseInfo( std::move(info)
), test( testCase ) {}
14161
14162     TestCase TestCase::withName( std::string const& _newName ) const {
14163         TestCase other( *this );
14164         other.name = _newName;
14165         return other;
14166     }
14167
14168     void TestCase::invoke() const {
14169         test->invoke();
14170     }
14171
14172     bool TestCase::operator == ( TestCase const& other ) const {
14173         return test.get() == other.test.get() &&
14174             name == other.name &&
14175             className == other.className;
14176     }
14177
14178     bool TestCase::operator < ( TestCase const& other ) const {
14179         return name < other.name;
14180     }
14181
14182     TestCaseInfo const& TestCase::getTestCaseInfo() const
14183     {
14184         return *this;
14185     }
14186
14187 } // end namespace Catch
14188 // end catch_test_case_info.cpp
14189 // start catch_test_case_registry_impl.cpp
14190
14191 #include <algorithm>
14192 #include <sstream>
14193
14194 namespace Catch {

```

```

14195
14196 namespace {
14197     struct TestHasher {
14198         using hash_t = uint64_t;
14199
14200         explicit TestHasher( hash_t hashSuffix ) :
14201             m_hashSuffix{ hashSuffix } {}
14202
14203         uint32_t operator()( TestCase const& t ) const {
14204             // FNV-1a hash with multiplication fold.
14205             const hash_t prime = 1099511628211u;
14206             hash_t hash = 14695981039346656037u;
14207             for ( const char c : t.name ) {
14208                 hash ^= c;
14209                 hash *= prime;
14210             }
14211             hash ^= m_hashSuffix;
14212             hash *= prime;
14213             const uint32_t low{ static_cast<uint32_t>( hash ) };
14214             const uint32_t high{ static_cast<uint32_t>( hash >> 32 ) };
14215             return low * high;
14216         }
14217
14218     private:
14219         hash_t m_hashSuffix;
14220     };
14221 } // end unnamed namespace
14222
14223 std::vector<TestCase> sortTests( IConfig const& config, std::vector<TestCase> const&
14224     unsortedTestCases ) {
14225     switch( config.runOrder() ) {
14226     case RunTests::InDeclarationOrder:
14227         // already in declaration order
14228         break;
14229
14230     case RunTests::InLexicographicalOrder: {
14231         std::vector<TestCase> sorted = unsortedTestCases;
14232         std::sort( sorted.begin(), sorted.end() );
14233         return sorted;
14234     }
14235
14236     case RunTests::InRandomOrder: {
14237         seedRng( config );
14238         TestHasher h{ config.rngSeed() };
14239
14240         using hashedTest = std::pair<TestHasher::hash_t, TestCase const*>;
14241         std::vector<hashedTest> indexed_tests;
14242         indexed_tests.reserve( unsortedTestCases.size() );
14243
14244         for ( auto const& testCase : unsortedTestCases ) {
14245             indexed_tests.emplace_back( h( testCase ), &testCase );
14246         }
14247
14248         std::sort( indexed_tests.begin(), indexed_tests.end(),
14249             []( hashedTest const& lhs, hashedTest const& rhs ) {
14250                 if ( lhs.first == rhs.first ) {
14251                     return lhs.second->name < rhs.second->name;
14252                 }
14253                 return lhs.first < rhs.first;
14254             } );
14255
14256         std::vector<TestCase> sorted;
14257         sorted.reserve( indexed_tests.size() );
14258
14259         for ( auto const& hashed : indexed_tests ) {
14260             sorted.emplace_back( *hashed.second );
14261         }
14262
14263         return sorted;
14264     }
14265     return unsortedTestCases;
14266 }
14267
14268 bool isThrowSafe( TestCase const& testCase, IConfig const& config ) {
14269     return !testCase.throws() || config.allowThrows();
14270 }
14271
14272 bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config ) {
14273     return testSpec.matches( testCase ) && isThrowSafe( testCase, config );
14274 }
14275
14276 void enforceNoDuplicateTestCases( std::vector<TestCase> const& functions ) {
14277     std::set<TestCase> seenFunctions;
14278     for( auto const& function : functions ) {
14279         auto prev = seenFunctions.insert( function );
14280         CATCH_ENFORCE( prev.second,

```



```

14281             "error: TEST_CASE( \"" « function.name « "\" ) already defined.\n"
14282             « "\tFirst seen at " «
prev.first->getTestCaseInfo().lineInfo « "\n"
14283             « "\tRedefined at " «
function.getTestCaseInfo().lineInfo );
14284     }
14285 }
14286
14287     std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
testSpec, IConfig const& config ) {
14288         std::vector<TestCase> filtered;
14289         filtered.reserve( testCases.size() );
14290         for (auto const& testCase : testCases) {
14291             if ( (!testSpec.hasFilters() && !testCase.isHidden()) ||
14292                 (testSpec.hasFilters() && matchTest(testCase, testSpec, config)) ) {
14293                 filtered.push_back(testCase);
14294             }
14295         }
14296         return filtered;
14297     }
14298     std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config ) {
14299         return getRegistryHub().getTestCaseRegistry().getAllTestsSorted( config );
14300     }
14301
14302     void TestRegistry::registerTest( TestCase const& testCase ) {
14303         std::string name = testCase.getTestCaseInfo().name;
14304         if( name.empty() ) {
14305             ReusableStringStream rss;
14306             rss << "Anonymous test case " << ++m_unnamedCount;
14307             return registerTest( testCase.withName( rss.str() ) );
14308         }
14309         m_functions.push_back( testCase );
14310     }
14311
14312     std::vector<TestCase> const& TestRegistry::getAllTests() const {
14313         return m_functions;
14314     }
14315     std::vector<TestCase> const& TestRegistry::getAllTestsSorted( IConfig const& config ) const {
14316         if( m_sortedFunctions.empty() )
14317             enforceNoDuplicateTestCases( m_functions );
14318         if( m_currentSortOrder != config.runOrder() || m_sortedFunctions.empty() ) {
14319             m_sortedFunctions = sortTests( config, m_functions );
14320             m_currentSortOrder = config.runOrder();
14321         }
14322         return m_sortedFunctions;
14323     }
14324 }
14325
14327     TestInvokerAsFunction::TestInvokerAsFunction( void(*testAsFunction)() ) noexcept :
m_testAsFunction( testAsFunction ) {}
14328
14329     void TestInvokerAsFunction::invoke() const {
14330         m_testAsFunction();
14331     }
14332
14333     std::string extractClassName( StringRef const& classOrQualifiedMethodName ) {
14334         std::string className(classOrQualifiedMethodName);
14335         if( startsWith( className, '&' ) )
14336         {
14337             std::size_t lastColons = className.rfind( "::" );
14338             std::size_t penultimateColons = className.rfind( "::", lastColons-1 );
14339             if( penultimateColons == std::string::npos )
14340                 penultimateColons = 1;
14341             className = className.substr( penultimateColons, lastColons-penultimateColons );
14342         }
14343         return className;
14344     }
14345
14346 } // end namespace Catch
14347 // end catch_test_case_registry_impl.cpp
14348 // start catch_test_case_tracker.cpp
14349
14350 #include <algorithm>
14351 #include <cassert>
14352 #include <stdexcept>
14353 #include <memory>
14354 #include <sstream>
14355
14356 #if defined(__clang__)
14357 #   pragma clang diagnostic push
14358 #   pragma clang diagnostic ignored "-Wexit-time-destructors"
14359 #endif
14360
14361 namespace Catch {
14362     namespace TestCaseTracking {
14363
14364         NameAndLocation::NameAndLocation( std::string const& _name, SourceLineInfo const& _location )

```

```

14365         :   name( _name ),
14366           location( _location )
14367     {}
14368
14369     ITracker::~ITracker() = default;
14370
14371     ITracker& TrackerContext::startRun() {
14372         m_rootTracker = std::make_shared<SectionTracker>( NameAndLocation( "{root}",
14373             CATCH_INTERNAL_LINEINFO ), *this, nullptr );
14374         m_currentTracker = nullptr;
14375         m_runState = Executing;
14376         return *m_rootTracker;
14377     }
14378
14379     void TrackerContext::endRun() {
14380         m_rootTracker.reset();
14381         m_currentTracker = nullptr;
14382         m_runState = NotStarted;
14383     }
14384
14385     void TrackerContext::startCycle() {
14386         m_currentTracker = m_rootTracker.get();
14387         m_runState = Executing;
14388     }
14389     void TrackerContext::completeCycle() {
14390         m_runState = CompletedCycle;
14391     }
14392
14393     bool TrackerContext::completedCycle() const {
14394         return m_runState == CompletedCycle;
14395     }
14396     ITracker& TrackerContext::currentTracker() {
14397         return *m_currentTracker;
14398     }
14399     void TrackerContext::setCurrentTracker( ITracker* tracker ) {
14400         m_currentTracker = tracker;
14401     }
14402
14403     TrackerBase::TrackerBase( NameAndLocation const& nameAndLocation, TrackerContext& ctx,
14404         ITracker* parent ) :
14405         ITracker( nameAndLocation ),
14406         m_ctx( ctx ),
14407         m_parent( parent )
14408     {}
14409
14410     bool TrackerBase::isComplete() const {
14411         return m_runState == CompletedSuccessfully || m_runState == Failed;
14412     }
14413     bool TrackerBase::isSuccessfullyCompleted() const {
14414         return m_runState == CompletedSuccessfully;
14415     }
14416     bool TrackerBase::isOpen() const {
14417         return m_runState != NotStarted && !isComplete();
14418     }
14419     bool TrackerBase::hasChildren() const {
14420         return !m_children.empty();
14421     }
14422
14423     void TrackerBase::addChild( ITrackerPtr const& child ) {
14424         m_children.push_back( child );
14425     }
14426
14427     ITrackerPtr TrackerBase::findChild( NameAndLocation const& nameAndLocation ) {
14428         auto it = std::find_if( m_children.begin(), m_children.end(),
14429             [&nameAndLocation]( ITrackerPtr const& tracker ) {
14430                 return
14431                     nameAndLocation.location &&
14432                     tracker->nameAndLocation().location ==
14433                     nameAndLocation.name;
14434             } );
14435         return( it != m_children.end() )
14436             ? *it
14437             : nullptr;
14438     }
14439     ITracker& TrackerBase::parent() {
14440         assert( m_parent ); // Should always be non-null except for root
14441         return *m_parent;
14442     }
14443
14444     void TrackerBase::openChild() {
14445         if( m_runState != ExecutingChildren ) {
14446             m_runState = ExecutingChildren;
14447             if( m_parent )
14448                 m_parent->openChild();
14449         }
14450     }

```

```

14448
14449     bool TrackerBase::isSectionTracker() const { return false; }
14450     bool TrackerBase::isGeneratorTracker() const { return false; }
14451
14452     void TrackerBase::open() {
14453         m_runState = Executing;
14454         moveToThis();
14455         if( m_parent )
14456             m_parent->openChild();
14457     }
14458
14459     void TrackerBase::close() {
14460
14461         // Close any still open children (e.g. generators)
14462         while( &m_ctx.currentTracker() != this )
14463             m_ctx.currentTracker().close();
14464
14465         switch( m_runState ) {
14466             case NeedsAnotherRun:
14467                 break;
14468
14469             case Executing:
14470                 m_runState = CompletedSuccessfully;
14471                 break;
14472             case ExecutingChildren:
14473                 if( std::all_of(m_children.begin(), m_children.end(), [](ITrackerPtr const& t){
14474                     return t->isComplete(); }) )
14475                     m_runState = CompletedSuccessfully;
14476                     break;
14477             case NotStarted:
14478             case CompletedSuccessfully:
14479             case Failed:
14480                 CATCH_INTERNAL_ERROR( "Illogical state: " « m_runState );
14481
14482             default:
14483                 CATCH_INTERNAL_ERROR( "Unknown state: " « m_runState );
14484         }
14485         moveToParent();
14486         m_ctx.completeCycle();
14487     }
14488     void TrackerBase::fail() {
14489         m_runState = Failed;
14490         if( m_parent )
14491             m_parent->markAsNeedingAnotherRun();
14492         moveToParent();
14493         m_ctx.completeCycle();
14494     }
14495     void TrackerBase::markAsNeedingAnotherRun() {
14496         m_runState = NeedsAnotherRun;
14497     }
14498
14499     void TrackerBase::moveToParent() {
14500         assert( m_parent );
14501         m_ctx.setCurrentTracker( m_parent );
14502     }
14503     void TrackerBase::moveToThis() {
14504         m_ctx.setCurrentTracker( this );
14505     }
14506
14507     SectionTracker::SectionTracker( NameAndLocation const& nameAndLocation, TrackerContext& ctx,
14508     ITracker* parent )
14509         : TrackerBase( nameAndLocation, ctx, parent ),
14510           m_trimmed_name(trim(nameAndLocation.name))
14511     {
14512         if( parent ) {
14513             while( !parent->isSectionTracker() )
14514                 parent = &parent->parent();
14515
14516             SectionTracker& parentSection = static_cast<SectionTracker&>( *parent );
14517             addNextFilters( parentSection.m_filters );
14518         }
14519     }
14520     bool SectionTracker::isComplete() const {
14521         bool complete = true;
14522
14523         if (m_filters.empty()
14524             || m_filters[0] == ""
14525             || std::find(m_filters.begin(), m_filters.end(), m_trimmed_name) != m_filters.end()) {
14526             complete = TrackerBase::isComplete();
14527         }
14528         return complete;
14529     }
14530
14531     bool SectionTracker::isSectionTracker() const { return true; }
14532

```

```

14533     SectionTracker& SectionTracker::acquire( TrackerContext& ctx, NameAndLocation const&
nameAndLocation ) {
14534         std::shared_ptr<SectionTracker> section;
14535
14536         ITracker& currentTracker = ctx.currentTracker();
14537         if( ITrackerPtr childTracker = currentTracker.findChild( nameAndLocation ) ) {
14538             assert( childTracker );
14539             assert( childTracker->isSectionTracker() );
14540             section = std::static_pointer_cast<SectionTracker>( childTracker );
14541         }
14542         else {
14543             section = std::make_shared<SectionTracker>( nameAndLocation, ctx, &currentTracker );
14544             currentTracker.addChild( section );
14545         }
14546         if( !ctx.completedCycle() )
14547             section->tryOpen();
14548         return *section;
14549     }
14550
14551     void SectionTracker::tryOpen() {
14552         if( !isComplete() )
14553             open();
14554     }
14555
14556     void SectionTracker::addInitialFilters( std::vector<std::string> const& filters ) {
14557         if( !filters.empty() ) {
14558             m_filters.reserve( m_filters.size() + filters.size() + 2 );
14559             m_filters.emplace_back(""); // Root - should never be consulted
14560             m_filters.emplace_back(""); // Test Case - not a section filter
14561             m_filters.insert( m_filters.end(), filters.begin(), filters.end() );
14562         }
14563     }
14564     void SectionTracker::addNextFilters( std::vector<std::string> const& filters ) {
14565         if( filters.size() > 1 )
14566             m_filters.insert( m_filters.end(), filters.begin()+1, filters.end() );
14567     }
14568
14569     std::vector<std::string> const& SectionTracker::getFilters() const {
14570         return m_filters;
14571     }
14572
14573     std::string const& SectionTracker::trimmedName() const {
14574         return m_trimmed_name;
14575     }
14576
14577 } // namespace TestCaseTracking
14578
14579 using TestCaseTracking::ITracker;
14580 using TestCaseTracking::TrackerContext;
14581 using TestCaseTracking::SectionTracker;
14582
14583 } // namespace Catch
14584
14585 #if defined(__clang__)
14586 #    pragma clang diagnostic pop
14587 #endif
14588 // end catch_test_case_tracker.cpp
14589 // start catch_test_registry.cpp
14590
14591 namespace Catch {
14592
14593     auto makeTestInvoker( void(*testAsFunction)() ) noexcept -> ITestInvoker* {
14594         return new(std::nothrow) TestInvokerAsFunction( testAsFunction );
14595     }
14596
14597     NameAndTags::NameAndTags( StringRef const& name_ , StringRef const& tags_ ) noexcept : name( name_
), tags( tags_ ) {}
14598
14599     AutoReg::AutoReg( ITestInvoker* invoker, SourceLineInfo const& lineInfo, StringRef const&
classOrMethod, NameAndTags const& nameAndTags ) noexcept {
14600         CATCH_TRY {
14601             getMutableRegistryHub()
14602                 .registerTest(
14603                     makeTestCase(
14604                         invoker,
14605                         extractClassName( classOrMethod ),
14606                         nameAndTags,
14607                         lineInfo));
14608         } CATCH_CATCH_ALL {
14609             // Do not throw when constructing global objects, instead register the exception to be
processed later
14610             getMutableRegistryHub().registerStartupException();
14611         }
14612     }
14613
14614     AutoReg::~AutoReg() = default;
14615 }

```

```

14616 // end catch_test_registry.cpp
14617 // start catch_test_spec.cpp
14618
14619 #include <algorithm>
14620 #include <string>
14621 #include <vector>
14622 #include <memory>
14623
14624 namespace Catch {
14625
14626     TestSpec::Pattern::Pattern( std::string const& name )
14627         : m_name( name )
14628     {}
14629
14630     TestSpec::Pattern::~~Pattern() = default;
14631
14632     std::string const& TestSpec::Pattern::name() const {
14633         return m_name;
14634     }
14635
14636     TestSpec::NamePattern::NamePattern( std::string const& name, std::string const& filterString )
14637         : Pattern( filterString )
14638         , m_wildcardPattern( toLower( name ), CaseSensitive::No )
14639     {}
14640
14641     bool TestSpec::NamePattern::matches( TestCaseInfo const& testCase ) const {
14642         return m_wildcardPattern.matches( testCase.name );
14643     }
14644
14645     TestSpec::TagPattern::TagPattern( std::string const& tag, std::string const& filterString )
14646         : Pattern( filterString )
14647         , m_tag( toLower( tag ) )
14648     {}
14649
14650     bool TestSpec::TagPattern::matches( TestCaseInfo const& testCase ) const {
14651         return std::find( begin( testCase.lcaseTags ),
14652             end( testCase.lcaseTags ),
14653             m_tag ) != end( testCase.lcaseTags );
14654     }
14655
14656     TestSpec::ExcludedPattern::ExcludedPattern( PatternPtr const& underlyingPattern )
14657         : Pattern( underlyingPattern->name() )
14658         , m_underlyingPattern( underlyingPattern )
14659     {}
14660
14661     bool TestSpec::ExcludedPattern::matches( TestCaseInfo const& testCase ) const {
14662         return !m_underlyingPattern->matches( testCase );
14663     }
14664
14665     bool TestSpec::Filter::matches( TestCaseInfo const& testCase ) const {
14666         return std::all_of( m_patterns.begin(), m_patterns.end(), [&]( PatternPtr const& p ){ return
p->matches( testCase ); } );
14667     }
14668
14669     std::string TestSpec::Filter::name() const {
14670         std::string name;
14671         for( auto const& p : m_patterns )
14672             name += p->name();
14673         return name;
14674     }
14675
14676     bool TestSpec::hasFilters() const {
14677         return !m_filters.empty();
14678     }
14679
14680     bool TestSpec::matches( TestCaseInfo const& testCase ) const {
14681         return std::any_of( m_filters.begin(), m_filters.end(), [&]( Filter const& f ){ return
f.matches( testCase ); } );
14682     }
14683
14684     TestSpec::Matches TestSpec::matchesByFilter( std::vector<TestCase> const& testCases, IConfig
const& config ) const
14685     {
14686         Matches matches( m_filters.size() );
14687         std::transform( m_filters.begin(), m_filters.end(), matches.begin(), [&]( Filter const& filter
){
14688             std::vector<TestCase const*> currentMatches;
14689             for( auto const& test : testCases )
14690                 if( isThrowSafe( test, config ) && filter.matches( test ) )
14691                     currentMatches.emplace_back( &test );
14692             return FilterMatch{ filter.name(), currentMatches };
14693         } );
14694         return matches;
14695     }
14696
14697     const TestSpec::vectorStrings& TestSpec::getInvalidArgs() const{
14698         return (m_invalidArgs);
14699     }

```

```

14699     }
14700
14701 }
14702 // end catch_test_spec.cpp
14703 // start catch_test_spec_parser.cpp
14704
14705 namespace Catch {
14706
14707     TestSpecParser::TestSpecParser( ITagAliasRegistry const& tagAliases ) : m_tagAliases( &tagAliases
14708 ) {}
14709
14710     TestSpecParser& TestSpecParser::parse( std::string const& arg ) {
14711         m_mode = None;
14712         m_exclusion = false;
14713         m_arg = m_tagAliases->expandAliases( arg );
14714         m_escapeChars.clear();
14715         m_substring.reserve(m_arg.size());
14716         m_patternName.reserve(m_arg.size());
14717         m_realPatternPos = 0;
14718
14719         for( m_pos = 0; m_pos < m_arg.size(); ++m_pos )
14720             //if visitChar fails
14721             if( !visitChar( m_arg[m_pos] ) ){
14722                 m_testSpec.m_invalidArgs.push_back(arg);
14723                 break;
14724             }
14725         endMode();
14726         return *this;
14727     }
14728     TestSpec TestSpecParser::testSpec() {
14729         addFilter();
14730         return m_testSpec;
14731     }
14732     bool TestSpecParser::visitChar( char c ) {
14733         if( (m_mode != EscapedName) && (c == '\\') ) {
14734             escape();
14735             addCharToPattern(c);
14736             return true;
14737         } else if( (m_mode != EscapedName) && (c == ',') ) {
14738             return separate();
14739         }
14740
14741         switch( m_mode ) {
14742             case None:
14743                 if( processNoneChar( c ) )
14744                     return true;
14745                 break;
14746             case Name:
14747                 processNameChar( c );
14748                 break;
14749             case EscapedName:
14750                 endMode();
14751                 addCharToPattern(c);
14752                 return true;
14753             default:
14754                 case Tag:
14755                 case QuotedName:
14756                     if( processOtherChar( c ) )
14757                         return true;
14758                     break;
14759         }
14760
14761         m_substring += c;
14762         if( !isControlChar( c ) ) {
14763             m_patternName += c;
14764             m_realPatternPos++;
14765         }
14766         return true;
14767     }
14768     // Two of the processing methods return true to signal the caller to return
14769     // without adding the given character to the current pattern strings
14770     bool TestSpecParser::processNoneChar( char c ) {
14771         switch( c ) {
14772             case '~':
14773                 return true;
14774             case '!':
14775                 m_exclusion = true;
14776                 return false;
14777             case '[':
14778                 startNewMode( Tag );
14779                 return false;
14780             case '"':
14781                 startNewMode( QuotedName );
14782                 return false;
14783             default:
14784                 startNewMode( Name );
14785                 return false;
14786         }

```

```

14785     }
14786 }
14787 void TestSpecParser::processNameChar( char c ) {
14788     if( c == '[' ) {
14789         if( m_substring == "exclude:" )
14790             m_exclusion = true;
14791         else
14792             endMode();
14793         startNewMode( Tag );
14794     }
14795 }
14796 bool TestSpecParser::processOtherChar( char c ) {
14797     if( !isControlChar( c ) )
14798         return false;
14799     m_substring += c;
14800     endMode();
14801     return true;
14802 }
14803 void TestSpecParser::startNewMode( Mode mode ) {
14804     m_mode = mode;
14805 }
14806 void TestSpecParser::endMode() {
14807     switch( m_mode ) {
14808         case Name:
14809             return addNamePattern();
14810         case Tag:
14811             return addTagPattern();
14812         case EscapedName:
14813             revertBackToLastMode();
14814             return;
14815         case None:
14816         default:
14817             return startNewMode( None );
14818     }
14819 }
14820 void TestSpecParser::escape() {
14821     saveLastMode();
14822     m_mode = EscapedName;
14823     m_escapeChars.push_back( m_realPatternPos );
14824 }
14825 bool TestSpecParser::isControlChar( char c ) const {
14826     switch( m_mode ) {
14827         default:
14828             return false;
14829         case None:
14830             return c == '~';
14831         case Name:
14832             return c == '[';
14833         case EscapedName:
14834             return true;
14835         case QuotedName:
14836             return c == '"';
14837         case Tag:
14838             return c == '[' || c == ']';
14839     }
14840 }
14841 void TestSpecParser::addFilter() {
14842     if( !m_currentFilter.m_patterns.empty() ) {
14843         m_testSpec.m_filters.push_back( m_currentFilter );
14844         m_currentFilter = TestSpec::Filter();
14845     }
14846 }
14847 void TestSpecParser::saveLastMode() {
14848     lastMode = m_mode;
14849 }
14850 void TestSpecParser::revertBackToLastMode() {
14851     m_mode = lastMode;
14852 }
14853 bool TestSpecParser::separate() {
14854     if( (m_mode==QuotedName) || (m_mode==Tag) ){
14855         //invalid argument, signal failure to previous scope.
14856         m_mode = None;
14857         m_pos = m_arg.size();
14858         m_substring.clear();
14859         m_patternName.clear();
14860         m_realPatternPos = 0;
14861         return false;
14862     }
14863     endMode();
14864     addFilter();
14865     return true; //success
14866 }

```

```

14872
14873     std::string TestSpecParser::preprocessPattern() {
14874         std::string token = m_patternName;
14875         for (std::size_t i = 0; i < m_escapeChars.size(); ++i)
14876             token = token.substr(0, m_escapeChars[i] - i) + token.substr(m_escapeChars[i] - i + 1);
14877         m_escapeChars.clear();
14878         if (startsWith(token, "exclude:")) {
14879             m_exclusion = true;
14880             token = token.substr(8);
14881         }
14882
14883         m_patternName.clear();
14884         m_realPatternPos = 0;
14885
14886         return token;
14887     }
14888
14889     void TestSpecParser::addNamePattern() {
14890         auto token = preprocessPattern();
14891
14892         if (!token.empty()) {
14893             TestSpec::PatternPtr pattern = std::make_shared<TestSpec::NamePattern>(token,
14894 m_substring);
14895             if (m_exclusion)
14896                 pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
14897             m_currentFilter.m_patterns.push_back(pattern);
14898             m_substring.clear();
14899             m_exclusion = false;
14900             m_mode = None;
14901         }
14902
14903     void TestSpecParser::addTagPattern() {
14904         auto token = preprocessPattern();
14905
14906         if (!token.empty()) {
14907             // If the tag pattern is the "hide and tag" shorthand (e.g. [.foo])
14908             // we have to create a separate hide tag and shorten the real one
14909             if (token.size() > 1 && token[0] == '.') {
14910                 token.erase(token.begin());
14911                 TestSpec::PatternPtr pattern = std::make_shared<TestSpec::TagPattern>(".",
14912 m_substring);
14913                 if (m_exclusion) {
14914                     pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
14915                 }
14916                 m_currentFilter.m_patterns.push_back(pattern);
14917
14918                 TestSpec::PatternPtr pattern = std::make_shared<TestSpec::TagPattern>(token, m_substring);
14919                 if (m_exclusion) {
14920                     pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
14921                 }
14922                 m_currentFilter.m_patterns.push_back(pattern);
14923             }
14924             m_substring.clear();
14925             m_exclusion = false;
14926             m_mode = None;
14927         }
14928     }
14929
14930     TestSpec parseTestSpec( std::string const& arg ) {
14931         return TestSpecParser( ITagAliasRegistry::get() ).parse( arg ).testSpec();
14932     }
14933
14934 } // namespace Catch
14935 // end catch_test_spec_parser.cpp
14936 // start catch_timer.cpp
14937
14938 #include <chrono>
14939
14940 static const uint64_t nanosecondsInSecond = 1000000000;
14941
14942 namespace Catch {
14943
14944     auto getCurrentNanosecondsSinceEpoch() -> uint64_t {
14945         return std::chrono::duration_cast<std::chrono::nanoseconds>(
14946 std::chrono::high_resolution_clock::now().time_since_epoch() ).count();
14947     }
14948
14949     namespace {
14950         auto estimateClockResolution() -> uint64_t {
14951             uint64_t sum = 0;
14952             static const uint64_t iterations = 1000000;
14953
14954             auto startTime = getCurrentNanosecondsSinceEpoch();
14955
14956             for( std::size_t i = 0; i < iterations; ++i ) {

```



```

14956
14957         uint64_t ticks;
14958         uint64_t baseTicks = getCurrentNanosecondsSinceEpoch();
14959         do {
14960             ticks = getCurrentNanosecondsSinceEpoch();
14961         } while( ticks == baseTicks );
14962
14963         auto delta = ticks - baseTicks;
14964         sum += delta;
14965
14966         // If we have been calibrating for over 3 seconds -- the clock
14967         // is terrible and we should move on.
14968         // TBD: How to signal that the measured resolution is probably wrong?
14969         if (ticks > startTime + 3 * nanosecondsInSecond) {
14970             return sum / ( i + 1u );
14971         }
14972     }
14973
14974     // We're just taking the mean, here. To do better we could take the std. dev and exclude
14975     outliers
14976     // - and potentially do more iterations if there's a high variance.
14977     return sum/iterations;
14978 }
14979
14980 auto getEstimatedClockResolution() -> uint64_t {
14981     static auto s_resolution = estimateClockResolution();
14982     return s_resolution;
14983 }
14984
14985 void Timer::start() {
14986     m_nanoseconds = getCurrentNanosecondsSinceEpoch();
14987 }
14988 auto Timer::getElapsedNanoseconds() const -> uint64_t {
14989     return getCurrentNanosecondsSinceEpoch() - m_nanoseconds;
14990 }
14991 auto Timer::getElapsedMicroseconds() const -> uint64_t {
14992     return getElapsedNanoseconds()/1000;
14993 }
14994 auto Timer::getElapsedMilliseconds() const -> unsigned int {
14995     return static_cast<unsigned int>(getElapsedMicroseconds()/1000);
14996 }
14997 auto Timer::getElapsedSeconds() const -> double {
14998     return getElapsedMicroseconds()/1000000.0;
14999 }
15000 } // namespace Catch
15001 // end catch_timer.cpp
15002 // start catch_tostring.cpp
15003
15004 #if defined(__clang__)
15005 #   pragma clang diagnostic push
15006 #   pragma clang diagnostic ignored "-Wexit-time-destructors"
15007 #   pragma clang diagnostic ignored "-Wglobal-constructors"
15008 #endif
15009
15010 // Enable specific decls locally
15011 #if !defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
15012 #define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
15013 #endif
15014
15015 #include <cmath>
15016 #include <iomanip>
15017
15018 namespace Catch {
15019
15020     namespace Detail {
15021
15022         const std::string unprintableString = "{?}";
15023
15024         namespace {
15025             const int hexThreshold = 255;
15026
15027             struct Endianness {
15028                 enum Arch { Big, Little };
15029
15030                 static Arch which() {
15031                     int one = 1;
15032                     // If the lowest byte we read is non-zero, we can assume
15033                     // that little endian format is used.
15034                     auto value = *reinterpret_cast<char*>(&one);
15035                     return value ? Little : Big;
15036                 }
15037             };
15038         }
15039
15040         std::string rawMemoryToString( const void *object, std::size_t size ) {
15041             // Reverse order for little endian architectures

```

```

15042         int i = 0, end = static_cast<int>( size ), inc = 1;
15043         if( Endianness::which() == Endianness::Little ) {
15044             i = end-1;
15045             end = inc = -1;
15046         }
15047
15048         unsigned char const *bytes = static_cast<unsigned char const *>(object);
15049         ReusableStringStream rss;
15050         rss << "0x" << std::setfill('0') << std::hex;
15051         for( ; i != end; i += inc )
15052             rss << std::setw(2) << static_cast<unsigned>(bytes[i]);
15053         return rss.str();
15054     }
15055 }
15056
15057 template<typename T>
15058 std::string fpToString( T value, int precision ) {
15059     if (Catch::isnan(value)) {
15060         return "nan";
15061     }
15062
15063     ReusableStringStream rss;
15064     rss << std::setprecision( precision )
15065         << std::fixed
15066         << value;
15067     std::string d = rss.str();
15068     std::size_t i = d.find_last_not_of( '0' );
15069     if( i != std::string::npos && i != d.size()-1 ) {
15070         if( d[i] == '.' )
15071             i++;
15072         d = d.substr( 0, i+1 );
15073     }
15074     return d;
15075 }
15076
15077 //
15078 // Out-of-line defs for full specialization of StringMaker
15079 //
15080 //
15081
15082 std::string StringMaker<std::string>::convert(const std::string& str) {
15083     if (!getCurrentContext().getConfig()->showInvisibles()) {
15084         return "'" + str + "'";
15085     }
15086
15087     std::string s("\\");
15088     for (char c : str) {
15089         switch (c) {
15090             case '\\n':
15091                 s.append("\\n");
15092                 break;
15093             case '\\t':
15094                 s.append("\\t");
15095                 break;
15096             default:
15097                 s.push_back(c);
15098                 break;
15099         }
15100     }
15101     s.append("\\");
15102     return s;
15103 }
15104
15105 #ifndef CATCH_CONFIG_CPP17_STRING_VIEW
15106 std::string StringMaker<std::string_view>::convert(std::string_view str) {
15107     return ::Catch::Detail::stringify(std::string{ str });
15108 }
15109 #endif
15110
15111 std::string StringMaker<char const*>::convert(char const* str) {
15112     if (str) {
15113         return ::Catch::Detail::stringify(std::string{ str });
15114     } else {
15115         return{ "{null string}" };
15116     }
15117 }
15118
15119 std::string StringMaker<char*>::convert(char* str) {
15120     if (str) {
15121         return ::Catch::Detail::stringify(std::string{ str });
15122     } else {
15123         return{ "{null string}" };
15124     }
15125 }
15126
15127 #ifndef CATCH_CONFIG_WCHAR
15128 std::string StringMaker<std::wstring>::convert(const std::wstring& wstr) {
15129     std::string s;
15130     s.reserve(wstr.size());

```

```

15131         for (auto c : wstr) {
15132             s += (c <= 0xff) ? static_cast<char>(c) : '?';
15133         }
15134         return ::Catch::Detail::stringify(s);
15135     }
15136
15137 # ifdef CATCH_CONFIG_CPP17_STRING_VIEW
15138     std::string StringMaker<std::wstring_view>::convert(std::wstring_view str) {
15139         return StringMaker<std::wstring>::convert(std::wstring(str));
15140     }
15141 # endif
15142
15143     std::string StringMaker<wchar_t const*>::convert(wchar_t const * str) {
15144         if (str) {
15145             return ::Catch::Detail::stringify(std::wstring{ str });
15146         } else {
15147             return{ "{null string}" };
15148         }
15149     }
15150     std::string StringMaker<wchar_t *>::convert(wchar_t * str) {
15151         if (str) {
15152             return ::Catch::Detail::stringify(std::wstring{ str });
15153         } else {
15154             return{ "{null string}" };
15155         }
15156     }
15157 #endif
15158
15159 #if defined(CATCH_CONFIG_CPP17_BYTE)
15160 #include <cstdint>
15161     std::string StringMaker<std::byte>::convert(std::byte value) {
15162         return ::Catch::Detail::stringify(std::to_integer<unsigned long long>(value));
15163     }
15164 #endif // defined(CATCH_CONFIG_CPP17_BYTE)
15165
15166     std::string StringMaker<int>::convert(int value) {
15167         return ::Catch::Detail::stringify(static_cast<long long>(value));
15168     }
15169     std::string StringMaker<long>::convert(long value) {
15170         return ::Catch::Detail::stringify(static_cast<long long>(value));
15171     }
15172     std::string StringMaker<long long>::convert(long long value) {
15173         ReusableStringStream rss;
15174         rss << value;
15175         if (value > Detail::hexThreshold) {
15176             rss << " (0x" << std::hex << value << ')';
15177         }
15178         return rss.str();
15179     }
15180
15181     std::string StringMaker<unsigned int>::convert(unsigned int value) {
15182         return ::Catch::Detail::stringify(static_cast<unsigned long long>(value));
15183     }
15184     std::string StringMaker<unsigned long>::convert(unsigned long value) {
15185         return ::Catch::Detail::stringify(static_cast<unsigned long long>(value));
15186     }
15187     std::string StringMaker<unsigned long long>::convert(unsigned long long value) {
15188         ReusableStringStream rss;
15189         rss << value;
15190         if (value > Detail::hexThreshold) {
15191             rss << " (0x" << std::hex << value << ')';
15192         }
15193         return rss.str();
15194     }
15195
15196     std::string StringMaker<bool>::convert(bool b) {
15197         return b ? "true" : "false";
15198     }
15199
15200     std::string StringMaker<signed char>::convert(signed char value) {
15201         if (value == '\r') {
15202             return "'\\r'";
15203         } else if (value == '\f') {
15204             return "'\\f'";
15205         } else if (value == '\n') {
15206             return "'\\n'";
15207         } else if (value == '\t') {
15208             return "'\\t'";
15209         } else if ('\0' <= value && value < ' ') {
15210             return ::Catch::Detail::stringify(static_cast<unsigned int>(value));
15211         } else {
15212             char chstr[] = "' '";
15213             chstr[1] = value;
15214             return chstr;
15215         }
15216     }
15217     std::string StringMaker<char>::convert(char c) {

```

```

15218         return ::Catch::Detail::stringify(static_cast<signed char>(c));
15219     }
15220     std::string StringMaker<unsigned char>::convert(unsigned char c) {
15221         return ::Catch::Detail::stringify(static_cast<char>(c));
15222     }
15223
15224     std::string StringMaker<std::nullptr_t>::convert(std::nullptr_t) {
15225         return "nullptr";
15226     }
15227
15228     int StringMaker<float>::precision = 5;
15229
15230     std::string StringMaker<float>::convert(float value) {
15231         return fpToString(value, precision) + 'f';
15232     }
15233
15234     int StringMaker<double>::precision = 10;
15235
15236     std::string StringMaker<double>::convert(double value) {
15237         return fpToString(value, precision);
15238     }
15239
15240     std::string ratio_string<std::atto>::symbol() { return "a"; }
15241     std::string ratio_string<std::femto>::symbol() { return "f"; }
15242     std::string ratio_string<std::pico>::symbol() { return "p"; }
15243     std::string ratio_string<std::nano>::symbol() { return "n"; }
15244     std::string ratio_string<std::micro>::symbol() { return "u"; }
15245     std::string ratio_string<std::milli>::symbol() { return "m"; }
15246
15247 } // end namespace Catch
15248
15249 #if defined(__clang__)
15250 #   pragma clang diagnostic pop
15251 #endif
15252
15253 // end catch_tostring.cpp
15254 // start catch_totals.cpp
15255
15256 namespace Catch {
15257
15258     Counts Counts::operator - ( Counts const& other ) const {
15259         Counts diff;
15260         diff.passed = passed - other.passed;
15261         diff.failed = failed - other.failed;
15262         diff.failedButOk = failedButOk - other.failedButOk;
15263         return diff;
15264     }
15265
15266     Counts& Counts::operator += ( Counts const& other ) {
15267         passed += other.passed;
15268         failed += other.failed;
15269         failedButOk += other.failedButOk;
15270         return *this;
15271     }
15272
15273     std::size_t Counts::total() const {
15274         return passed + failed + failedButOk;
15275     }
15276     bool Counts::allPassed() const {
15277         return failed == 0 && failedButOk == 0;
15278     }
15279     bool Counts::allOk() const {
15280         return failed == 0;
15281     }
15282
15283     Totals Totals::operator - ( Totals const& other ) const {
15284         Totals diff;
15285         diff.assertions = assertions - other.assertions;
15286         diff.testCases = testCases - other.testCases;
15287         return diff;
15288     }
15289
15290     Totals& Totals::operator += ( Totals const& other ) {
15291         assertions += other.assertions;
15292         testCases += other.testCases;
15293         return *this;
15294     }
15295
15296     Totals Totals::delta( Totals const& prevTotals ) const {
15297         Totals diff = *this - prevTotals;
15298         if( diff.assertions.failed > 0 )
15299             ++diff.testCases.failed;
15300         else if( diff.assertions.failedButOk > 0 )
15301             ++diff.testCases.failedButOk;
15302         else
15303             ++diff.testCases.passed;
15304         return diff;

```

```

15305     }
15306
15307 }
15308 // end catch_totals.cpp
15309 // start catch_uncaught_exceptions.cpp
15310
15311 // start catch_config_uncaught_exceptions.hpp
15312
15313 //          Copyright Catch2 Authors
15314 // Distributed under the Boost Software License, Version 1.0.
15315 // (See accompanying file LICENSE_1_0.txt or copy at
15316 //  https://www.boost.org/LICENSE\_1\_0.txt)
15317
15318 // SPDX-License-Identifier: BSL-1.0
15319
15320 #ifndef CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
15321 #define CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
15322
15323 #if defined(_MSC_VER)
15324 #   if _MSC_VER >= 1900 // Visual Studio 2015 or newer
15325 #       define CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
15326 #   endif
15327 #endif
15328
15329 #include <exception>
15330
15331 #if defined(__cpp_lib_uncaught_exceptions) \
15332     && !defined(CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)
15333
15334 #   define CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
15335 #endif // __cpp_lib_uncaught_exceptions
15336
15337 #if defined(CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS) \
15338     && !defined(CATCH_CONFIG_NO_CPP17_UNCAUGHT_EXCEPTIONS) \
15339     && !defined(CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)
15340
15341 #   define CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
15342 #endif
15343
15344 #endif // CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
15345 // end catch_config_uncaught_exceptions.hpp
15346 #include <exception>
15347
15348 namespace Catch {
15349     bool uncaught_exceptions() {
15350 #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
15351         return false;
15352 #elif defined(CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)
15353         return std::uncaught_exceptions() > 0;
15354 #else
15355         return std::uncaught_exception();
15356 #endif
15357     }
15358 } // end namespace Catch
15359 // end catch_uncaught_exceptions.cpp
15360 // start catch_version.cpp
15361
15362 #include <ostream>
15363
15364 namespace Catch {
15365     Version::Version
15366     (   unsigned int _majorVersion,
15367         unsigned int _minorVersion,
15368         unsigned int _patchNumber,
15369         char const * _branchName,
15370         unsigned int _buildNumber )
15371     :   majorVersion( _majorVersion ),
15372         minorVersion( _minorVersion ),
15373         patchNumber( _patchNumber ),
15374         branchName( _branchName ),
15375         buildNumber( _buildNumber )
15376     {}
15377
15378
15379     std::ostream& operator << ( std::ostream& os, Version const& version ) {
15380         os << version.majorVersion << '.'
15381            << version.minorVersion << '.'
15382            << version.patchNumber;
15383         // branchName is never null -> 0th char is \0 if it is empty
15384         if (version.branchName[0]) {
15385             os << '-' << version.branchName
15386                << '.' << version.buildNumber;
15387         }
15388         return os;
15389     }
15390
15391     Version const& libraryVersion() {

```

```

15392         static Version version( 2, 13, 10, "", 0 );
15393         return version;
15394     }
15395 }
15396 // end catch_version.cpp
15397 // start catch_wildcard_pattern.cpp
15398 namespace Catch {
15400     WildcardPattern::WildcardPattern( std::string const& pattern,
15401                                       CaseSensitive::Choice caseSensitivity )
15402     :   m_caseSensitivity( caseSensitivity ),
15403         m_pattern( normaliseString( pattern ) )
15404     {
15405         if( startsWith( m_pattern, '*' ) ) {
15406             m_pattern = m_pattern.substr( 1 );
15407             m_wildcard = WildcardAtStart;
15408         }
15409         if( endsWith( m_pattern, '*' ) ) {
15410             m_pattern = m_pattern.substr( 0, m_pattern.size()-1 );
15411             m_wildcard = static_cast<WildcardPosition>( m_wildcard | WildcardAtEnd );
15412         }
15413     }
15414
15415     bool WildcardPattern::matches( std::string const& str ) const {
15416         switch( m_wildcard ) {
15417             case NoWildcard:
15418                 return m_pattern == normaliseString( str );
15419             case WildcardAtStart:
15420                 return endsWith( normaliseString( str ), m_pattern );
15421             case WildcardAtEnd:
15422                 return startsWith( normaliseString( str ), m_pattern );
15423             case WildcardAtBothEnds:
15424                 return contains( normaliseString( str ), m_pattern );
15425             default:
15426                 CATCH_INTERNAL_ERROR( "Unknown enum" );
15427         }
15428     }
15429
15430     std::string WildcardPattern::normaliseString( std::string const& str ) const {
15431         return trim( m_caseSensitivity == CaseSensitive::No ? toLower( str ) : str );
15432     }
15433 }
15434 // end catch_wildcard_pattern.cpp
15435 // start catch_xmlwriter.cpp
15436 #include <iomanip>
15437 #include <type_traits>
15438 namespace Catch {
15439     namespace {
15440         size_t trailingBytes( unsigned char c ) {
15441             if ( (c & 0xE0) == 0xC0 ) {
15442                 return 2;
15443             }
15444             if ( (c & 0xF0) == 0xE0 ) {
15445                 return 3;
15446             }
15447             if ( (c & 0xF8) == 0xF0 ) {
15448                 return 4;
15449             }
15450             CATCH_INTERNAL_ERROR( "Invalid multibyte utf-8 start byte encountered" );
15451         }
15452
15453         uint32_t headerValue( unsigned char c ) {
15454             if ( (c & 0xE0) == 0xC0 ) {
15455                 return c & 0x1F;
15456             }
15457             if ( (c & 0xF0) == 0xE0 ) {
15458                 return c & 0x0F;
15459             }
15460             if ( (c & 0xF8) == 0xF0 ) {
15461                 return c & 0x07;
15462             }
15463             CATCH_INTERNAL_ERROR( "Invalid multibyte utf-8 start byte encountered" );
15464         }
15465
15466         void hexEscapeChar( std::ostream& os, unsigned char c ) {
15467             std::ios_base::fmtflags f( os.flags() );
15468             os << "\\x"
15469                 << std::uppercase << std::hex << std::setfill( '0' ) << std::setw( 2 )
15470                 << static_cast<int>( c );
15471             os.flags( f );
15472         }
15473     }
15474 }

```

```

15479
15480         bool shouldNewline(XmlFormatting fmt) {
15481             return !!(static_cast<std::underlying_type<XmlFormatting>::type>(fmt &
15482                 XmlFormatting::Newline));
15483         }
15484         bool shouldIndent(XmlFormatting fmt) {
15485             return !!(static_cast<std::underlying_type<XmlFormatting>::type>(fmt &
15486                 XmlFormatting::Indent));
15487         }
15488     } // anonymous namespace
15489
15490     XmlFormatting operator | (XmlFormatting lhs, XmlFormatting rhs) {
15491         return static_cast<XmlFormatting>(
15492             static_cast<std::underlying_type<XmlFormatting>::type>(lhs) |
15493             static_cast<std::underlying_type<XmlFormatting>::type>(rhs)
15494         );
15495     }
15496
15497     XmlFormatting operator & (XmlFormatting lhs, XmlFormatting rhs) {
15498         return static_cast<XmlFormatting>(
15499             static_cast<std::underlying_type<XmlFormatting>::type>(lhs) &
15500             static_cast<std::underlying_type<XmlFormatting>::type>(rhs)
15501         );
15502     }
15503
15504     XmlEncode::XmlEncode( std::string const& str, ForWhat forWhat )
15505         :   m_str( str ),
15506             m_forWhat( forWhat )
15507     {}
15508
15509     void XmlEncode::encodeTo( std::ostream& os ) const {
15510         // Apostrophe escaping not necessary if we always use " to write attributes
15511         // (see: http://www.w3.org/TR/xml/#syntax)
15512
15513         for( std::size_t idx = 0; idx < m_str.size(); ++ idx ) {
15514             unsigned char c = m_str[idx];
15515             switch (c) {
15516                 case '<':   os << "&lt;"; break;
15517                 case '&':   os << "&amp;"; break;
15518
15519                 case '>':
15520                     // See: http://www.w3.org/TR/xml/#syntax
15521                     if (idx > 2 && m_str[idx - 1] == '[' && m_str[idx - 2] == ']')
15522                         os << "&gt;";
15523                     else
15524                         os << c;
15525                     break;
15526
15527                 case '\"':
15528                     if (m_forWhat == ForAttributes)
15529                         os << "&quot;";
15530                     else
15531                         os << c;
15532                     break;
15533
15534                 default:
15535                     // Check for control characters and invalid utf-8
15536
15537                     // Escape control characters in standard ascii
15538                     // see
15539                     http://stackoverflow.com/questions/404107/why-are-control-characters-illegal-in-xml-1-0
15540                     if (c < 0x09 || (c > 0x0D && c < 0x20) || c == 0x7F) {
15541                         hexEscapeChar(os, c);
15542                     }
15543
15544                     // Plain ASCII: Write it to stream
15545                     if (c < 0x7F) {
15546                         os << c;
15547                     }
15548                     break;
15549
15550                     // UTF-8 territory
15551                     // Check if the encoding is valid and if it is not, hex escape bytes.
15552                     // Important: We do not check the exact decoded values for validity, only the
15553                     encoding format
15554                     // First check that this bytes is a valid lead byte:
15555                     // This means that it is not encoded as 1111 1XXX
15556                     // Or as 10XX XXXX
15557                     if (c < 0xC0 ||
15558                         c >= 0xF8) {
15559                         hexEscapeChar(os, c);
15560                     }
15561

```

```

15562         auto encBytes = trailingBytes(c);
15563         // Are there enough bytes left to avoid accessing out-of-bounds memory?
15564         if (idx + encBytes - 1 >= m_str.size()) {
15565             hexEscapeChar(os, c);
15566             break;
15567         }
15568         // The header is valid, check data
15569         // The next encBytes bytes must together be a valid utf-8
15570         // This means: bitpattern 10XX XXXX and the extracted value is sane (ish)
15571         bool valid = true;
15572         uint32_t value = headerValue(c);
15573         for (std::size_t n = 1; n < encBytes; ++n) {
15574             unsigned char nc = m_str[idx + n];
15575             valid &= ((nc & 0xC0) == 0x80);
15576             value = (value << 6) | (nc & 0x3F);
15577         }
15578
15579         if (
15580             // Wrong bit pattern of following bytes
15581             (!valid) ||
15582             // Overlong encodings
15583             (value < 0x80) ||
15584             (0x80 <= value && value < 0x800 && encBytes > 2) ||
15585             (0x800 < value && value < 0x10000 && encBytes > 3) ||
15586             // Encoded value out of range
15587             (value >= 0x110000)
15588         ) {
15589             hexEscapeChar(os, c);
15590             break;
15591         }
15592
15593         // If we got here, this is in fact a valid(ish) utf-8 sequence
15594         for (std::size_t n = 0; n < encBytes; ++n) {
15595             os << m_str[idx + n];
15596         }
15597         idx += encBytes - 1;
15598         break;
15599     }
15600 }
15601 }
15602
15603 std::ostream& operator << ( std::ostream& os, XmlEncode const& xmlEncode ) {
15604     xmlEncode.encodeTo( os );
15605     return os;
15606 }
15607
15608 XmlWriter::ScopedElement::ScopedElement( XmlWriter* writer, XmlFormatting fmt )
15609     : m_writer( writer ),
15610     m_fmt( fmt )
15611 {}
15612
15613 XmlWriter::ScopedElement::ScopedElement( ScopedElement&& other ) noexcept
15614     : m_writer( other.m_writer ),
15615     m_fmt( other.m_fmt )
15616 {
15617     other.m_writer = nullptr;
15618     other.m_fmt = XmlFormatting::None;
15619 }
15620
15621 XmlWriter::ScopedElement& XmlWriter::ScopedElement::operator=( ScopedElement&& other ) noexcept {
15622     if ( m_writer ) {
15623         m_writer->endElement();
15624     }
15625     m_writer = other.m_writer;
15626     other.m_writer = nullptr;
15627     m_fmt = other.m_fmt;
15628     other.m_fmt = XmlFormatting::None;
15629     return *this;
15630 }
15631
15632 XmlWriter::ScopedElement::~~ScopedElement() {
15633     if (m_writer) {
15634         m_writer->endElement(m_fmt);
15635     }
15636 }
15637
15638 XmlWriter::ScopedElement& XmlWriter::ScopedElement::writeText( std::string const& text,
15639     XmlFormatting fmt ) {
15640     m_writer->writeText( text, fmt );
15641     return *this;
15642 }
15643
15644 XmlWriter::XmlWriter( std::ostream& os ) : m_os( os )
15645 {
15646     writeDeclaration();
15647 }
15648
15649 XmlWriter::~~XmlWriter() {

```



```

15648         while (!m_tags.empty()) {
15649             endElement();
15650         }
15651         newlineIfNecessary();
15652     }
15653
15654     XmlWriter& XmlWriter::startElement( std::string const& name, XmlFormatting fmt ) {
15655         ensureTagClosed();
15656         newlineIfNecessary();
15657         if (shouldIndent(fmt)) {
15658             m_os << m_indent;
15659             m_indent += " ";
15660         }
15661         m_os << '<' << name;
15662         m_tags.push_back( name );
15663         m_tagIsOpen = true;
15664         applyFormatting(fmt);
15665         return *this;
15666     }
15667
15668     XmlWriter::ScopedElement XmlWriter::scopedElement( std::string const& name, XmlFormatting fmt ) {
15669         ScopedElement scoped( this, fmt );
15670         startElement( name, fmt );
15671         return scoped;
15672     }
15673
15674     XmlWriter& XmlWriter::endElement(XmlFormatting fmt) {
15675         m_indent = m_indent.substr(0, m_indent.size() - 2);
15676
15677         if( m_tagIsOpen ) {
15678             m_os << ">";
15679             m_tagIsOpen = false;
15680         } else {
15681             newlineIfNecessary();
15682             if (shouldIndent(fmt)) {
15683                 m_os << m_indent;
15684             }
15685             m_os << "</" << m_tags.back() << ">";
15686         }
15687         m_os << std::flush;
15688         applyFormatting(fmt);
15689         m_tags.pop_back();
15690         return *this;
15691     }
15692
15693     XmlWriter& XmlWriter::writeAttribute( std::string const& name, std::string const& attribute ) {
15694         if( !name.empty() && !attribute.empty() )
15695             m_os << ' ' << name << "=" << XmlEncode( attribute, XmlEncode::ForAttributes ) << "'";
15696         return *this;
15697     }
15698
15699     XmlWriter& XmlWriter::writeAttribute( std::string const& name, bool attribute ) {
15700         m_os << ' ' << name << "=" << ( attribute ? "true" : "false" ) << "'";
15701         return *this;
15702     }
15703
15704     XmlWriter& XmlWriter::writeText( std::string const& text, XmlFormatting fmt ) {
15705         if( !text.empty() ) {
15706             bool tagWasOpen = m_tagIsOpen;
15707             ensureTagClosed();
15708             if (tagWasOpen && shouldIndent(fmt)) {
15709                 m_os << m_indent;
15710             }
15711             m_os << XmlEncode( text );
15712             applyFormatting(fmt);
15713         }
15714         return *this;
15715     }
15716
15717     XmlWriter& XmlWriter::writeComment( std::string const& text, XmlFormatting fmt ) {
15718         ensureTagClosed();
15719         if (shouldIndent(fmt)) {
15720             m_os << m_indent;
15721         }
15722         m_os << "<!--" << text << "-->";
15723         applyFormatting(fmt);
15724         return *this;
15725     }
15726
15727     void XmlWriter::writeStylesheetRef( std::string const& url ) {
15728         m_os << "<?xml-stylesheet type=\"" << "text/xsl" << " href=\"" << url << "\"?>\n";
15729     }
15730
15731     XmlWriter& XmlWriter::writeBlankLine() {
15732         ensureTagClosed();
15733         m_os << '\n';
15734         return *this;

```

```

15735     }
15736
15737     void XmlWriter::ensureTagClosed() {
15738         if( m_tagIsOpen ) {
15739             m_os << '>' << std::flush;
15740             newlineIfNecessary();
15741             m_tagIsOpen = false;
15742         }
15743     }
15744
15745     void XmlWriter::applyFormatting(XmlFormatting fmt) {
15746         m_needsNewline = shouldNewline(fmt);
15747     }
15748
15749     void XmlWriter::writeDeclaration() {
15750         m_os << "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
15751     }
15752
15753     void XmlWriter::newlineIfNecessary() {
15754         if( m_needsNewline ) {
15755             m_os << std::endl;
15756             m_needsNewline = false;
15757         }
15758     }
15759 }
15760 // end catch_xmlwriter.cpp
15761 // start catch_reporter_bases.cpp
15762
15763 #include <cstring>
15764 #include <cfloat>
15765 #include <cstdio>
15766 #include <cassert>
15767 #include <memory>
15768
15769 namespace Catch {
15770     void prepareExpandedExpression( AssertionResult& result ) {
15771         result.getExpandedExpression();
15772     }
15773
15774     // Because formatting using c++ streams is stateful, drop down to C is required
15775     // Alternatively we could use stringstream, but its performance is... not good.
15776     std::string getFormattedDuration( double duration ) {
15777         // Max exponent + 1 is required to represent the whole part
15778         // + 1 for decimal point
15779         // + 3 for the 3 decimal places
15780         // + 1 for null terminator
15781         const std::size_t maxDoubleSize = DBL_MAX_10_EXP + 1 + 1 + 3 + 1;
15782         char buffer[maxDoubleSize];
15783
15784         // Save previous errno, to prevent sprintf from overwriting it
15785         ErrnoGuard guard;
15786 #ifdef _MSC_VER
15787         sprintf_s(buffer, "%.3f", duration);
15788 #else
15789         std::sprintf(buffer, "%.3f", duration);
15790 #endif
15791         return std::string(buffer);
15792     }
15793
15794     bool shouldShowDuration( IConfig const& config, double duration ) {
15795         if ( config.showDurations() == ShowDurations::Always ) {
15796             return true;
15797         }
15798         if ( config.showDurations() == ShowDurations::Never ) {
15799             return false;
15800         }
15801         const double min = config.minDuration();
15802         return min >= 0 && duration >= min;
15803     }
15804
15805     std::string serializeFilters( std::vector<std::string> const& container ) {
15806         ReusableStringStream oss;
15807         bool first = true;
15808         for (auto&& filter : container)
15809         {
15810             if (!first)
15811                 oss << ' ';
15812             else
15813                 first = false;
15814
15815             oss << filter;
15816         }
15817         return oss.str();
15818     }
15819
15820     TestEventListenerBase::TestEventListenerBase(ReporterConfig const & _config)
15821         : StreamingReporterBase(_config) {}

```

```

15822
15823     std::set<Verbosity> TestEventListenerBase::getSupportedVerbsities() {
15824         return { Verbosity::Quiet, Verbosity::Normal, Verbosity::High };
15825     }
15826
15827     void TestEventListenerBase::assertionStarting(AssertionInfo const &) {}
15828
15829     bool TestEventListenerBase::assertionEnded(AssertionStats const &) {
15830         return false;
15831     }
15832
15833 } // end namespace Catch
15834 // end catch_reporter_bases.cpp
15835 // start catch_reporter_compact.cpp
15836
15837 namespace {
15838
15839 #ifndef CATCH_PLATFORM_MAC
15840     const char* failedString() { return "FAILED"; }
15841     const char* passedString() { return "PASSED"; }
15842 #else
15843     const char* failedString() { return "failed"; }
15844     const char* passedString() { return "passed"; }
15845 #endif
15846
15847     // Colour::LightGrey
15848     Catch::Colour::Code dimColour() { return Catch::Colour::FileName; }
15849
15850     std::string bothOrAll( std::size_t count ) {
15851         return count == 1 ? std::string() :
15852             count == 2 ? "both " : "all " ;
15853     }
15854
15855 } // anon namespace
15856
15857 namespace Catch {
15858     namespace {
15859         // Colour, message variants:
15860         // - white: No tests ran.
15861         // - red: Failed [both/all] N test cases, failed [both/all] M assertions.
15862         // - white: Passed [both/all] N test cases (no assertions).
15863         // - red: Failed N tests cases, failed M assertions.
15864         // - green: Passed [both/all] N tests cases with M assertions.
15865         void printTotals(std::ostream& out, const Totals& totals) {
15866             if (totals.testCases.total() == 0) {
15867                 out << "No tests ran.";
15868             } else if (totals.testCases.failed == totals.testCases.total()) {
15869                 Colour colour(Colour::ResultError);
15870                 const std::string qualify_assertions_failed =
15871                     totals.assertions.failed == totals.assertions.total() ?
15872                     bothOrAll(totals.assertions.failed) : std::string();
15873                 out <<
15874                     "Failed " << bothOrAll(totals.testCases.failed)
15875                     << pluralise(totals.testCases.failed, "test case") << ", "
15876                     << "failed " <<
15877                     qualify_assertions_failed <<
15878                     pluralise(totals.assertions.failed, "assertion") << '.';
15879             } else if (totals.assertions.total() == 0) {
15880                 out <<
15881                     "Passed " << bothOrAll(totals.testCases.total())
15882                     << pluralise(totals.testCases.total(), "test case")
15883                     << " (no assertions).";
15884             } else if (totals.assertions.failed) {
15885                 Colour colour(Colour::ResultError);
15886                 out <<
15887                     "Failed " << pluralise(totals.testCases.failed, "test case") << ", "
15888                     << "failed " <<
15889                     pluralise(totals.assertions.failed, "assertion") << '.';
15890             } else {
15891                 Colour colour(Colour::ResultSuccess);
15892                 out <<
15893                     "Passed " << bothOrAll(totals.testCases.passed)
15894                     << pluralise(totals.testCases.passed, "test case") <<
15895                     " with " << pluralise(totals.assertions.passed, "assertion") << '.';
15896             }
15897         }
15898     }
15899
15900     // Implementation of CompactReporter formatting
15901     class AssertionPrinter {
15902     public:
15903         AssertionPrinter& operator= (AssertionPrinter const&) = delete;
15904         AssertionPrinter(AssertionPrinter const&) = delete;
15905         AssertionPrinter(std::ostream& _stream, AssertionStats const& _stats, bool
15906             _printInfoMessages)
15907             : stream(_stream)
15908             , result(_stats.assertionResult)
15909             , messages(_stats.infoMessages)

```

```

15906         , itMessage(_stats.infoMessages.begin())
15907         , printInfoMessages(_printInfoMessages) {}
15908
15909     void print() {
15910         printSourceInfo();
15911
15912         itMessage = messages.begin();
15913
15914         switch (result.getResultType()) {
15915             case ResultWas::Ok:
15916                 printResultType(Colour::ResultSuccess, passedString());
15917                 printOriginalExpression();
15918                 printReconstructedExpression();
15919                 if (!result.hasExpression())
15920                     printRemainingMessages(Colour::None);
15921             else
15922                 printRemainingMessages();
15923             break;
15924             case ResultWas::ExpressionFailed:
15925                 if (result.isOk())
15926                     printResultType(Colour::ResultSuccess, failedString() + std::string(" -
but was ok"));
15927                 else
15928                     printResultType(Colour::Error, failedString());
15929                 printOriginalExpression();
15930                 printReconstructedExpression();
15931                 printRemainingMessages();
15932                 break;
15933             case ResultWas::ThrowException:
15934                 printResultType(Colour::Error, failedString());
15935                 printIssue("unexpected exception with message:");
15936                 printMessage();
15937                 printExpressionWas();
15938                 printRemainingMessages();
15939                 break;
15940             case ResultWas::FatalErrorCondition:
15941                 printResultType(Colour::Error, failedString());
15942                 printIssue("fatal error condition with message:");
15943                 printMessage();
15944                 printExpressionWas();
15945                 printRemainingMessages();
15946                 break;
15947             case ResultWas::DidntThrowException:
15948                 printResultType(Colour::Error, failedString());
15949                 printIssue("expected exception, got none");
15950                 printExpressionWas();
15951                 printRemainingMessages();
15952                 break;
15953             case ResultWas::Info:
15954                 printResultType(Colour::None, "info");
15955                 printMessage();
15956                 printRemainingMessages();
15957                 break;
15958             case ResultWas::Warning:
15959                 printResultType(Colour::None, "warning");
15960                 printMessage();
15961                 printRemainingMessages();
15962                 break;
15963             case ResultWas::ExplicitFailure:
15964                 printResultType(Colour::Error, failedString());
15965                 printIssue("explicitly");
15966                 printRemainingMessages(Colour::None);
15967                 break;
15968             // These cases are here to prevent compiler warnings
15969             case ResultWas::Unknown:
15970             case ResultWas::FailureBit:
15971             case ResultWas::Exception:
15972                 printResultType(Colour::Error, "** internal error **");
15973                 break;
15974         }
15975     }
15976
15977     private:
15978     void printSourceInfo() const {
15979         Colour colourGuard(Colour::FileName);
15980         stream << result.getSourceInfo() << ":'";
15981     }
15982
15983     void printResultType(Colour::Code colour, std::string const& passOrFail) const {
15984         if (!passOrFail.empty()) {
15985             {
15986                 Colour colourGuard(colour);
15987                 stream << ' ' << passOrFail;
15988             }
15989             stream << ":'";
15990         }
15991     }

```

```

15992
15993     void printIssue(std::string const& issue) const {
15994         stream << ' ' << issue;
15995     }
15996
15997     void printExpressionWas() {
15998         if (result.hasExpression()) {
15999             stream << ' ';
16000             {
16001                 Colour colour(dimColour());
16002                 stream << " expression was:";
16003             }
16004             printOriginalExpression();
16005         }
16006     }
16007
16008     void printOriginalExpression() const {
16009         if (result.hasExpression()) {
16010             stream << ' ' << result.getExpression();
16011         }
16012     }
16013
16014     void printReconstructedExpression() const {
16015         if (result.hasExpandedExpression()) {
16016             {
16017                 Colour colour(dimColour());
16018                 stream << " for: ";
16019             }
16020             stream << result.getExpandedExpression();
16021         }
16022     }
16023
16024     void printMessage() {
16025         if (itMessage != messages.end()) {
16026             stream << " '" << itMessage->message << '\n';
16027             ++itMessage;
16028         }
16029     }
16030
16031     void printRemainingMessages(Colour::Code colour = dimColour()) {
16032         if (itMessage == messages.end())
16033             return;
16034
16035         const auto itEnd = messages.cend();
16036         const auto N = static_cast<std::size_t>(std::distance(itMessage, itEnd));
16037
16038         {
16039             Colour colourGuard(colour);
16040             stream << " with " << pluralise(N, "message") << ':';
16041         }
16042
16043         while (itMessage != itEnd) {
16044             // If this assertion is a warning ignore any INFO messages
16045             if (printInfoMessages || itMessage->type != ResultWas::Info) {
16046                 printMessage();
16047                 if (itMessage != itEnd) {
16048                     Colour colourGuard(dimColour());
16049                     stream << " and";
16050                 }
16051                 continue;
16052             }
16053             ++itMessage;
16054         }
16055     }
16056
16057 private:
16058     std::ostream& stream;
16059     AssertionResult const& result;
16060     std::vector<MessageInfo> messages;
16061     std::vector<MessageInfo>::const_iterator itMessage;
16062     bool printInfoMessages;
16063 };
16064
16065 } // anon namespace
16066
16067 std::string CompactReporter::getDescription() {
16068     return "Reports test results on a single line, suitable for IDEs";
16069 }
16070
16071 void CompactReporter::noMatchingTestCases( std::string const& spec ) {
16072     stream << "No test cases matched '" << spec << '\n' << std::endl;
16073 }
16074
16075 void CompactReporter::assertionStarting( AssertionInfo const& ) {}
16076
16077 bool CompactReporter::assertionEnded( AssertionStats const& _assertionStats ) {
16078     AssertionResult const& result = _assertionStats.assertionResult;

```

```

16079
16080     bool printInfoMessages = true;
16081
16082     // Drop out if result was successful and we're not printing those
16083     if( !m_config->includeSuccessfulResults() && result.isOk() ) {
16084         if( result.getResultType() != ResultWas::Warning )
16085             return false;
16086         printInfoMessages = false;
16087     }
16088
16089     AssertionPrinter printer( stream, _assertionStats, printInfoMessages );
16090     printer.print();
16091
16092     stream << std::endl;
16093     return true;
16094 }
16095
16096 void CompactReporter::sectionEnded(SectionStats const& _sectionStats) {
16097     double dur = _sectionStats.durationInSeconds;
16098     if ( shouldShowDuration( *m_config, dur ) ) {
16099         stream << getFormattedDuration( dur ) << " s: " << _sectionStats.sectionInfo.name <<
std::endl;
16100     }
16101 }
16102
16103 void CompactReporter::testRunEnded( TestRunStats const& _testRunStats ) {
16104     printTotals( stream, _testRunStats.totals );
16105     stream << '\n' << std::endl;
16106     StreamingReporterBase::testRunEnded( _testRunStats );
16107 }
16108
16109 CompactReporter::~CompactReporter() {}
16110
16111 CATCH_REGISTER_REPORTER( "compact", CompactReporter )
16112
16113 } // end namespace Catch
16114 // end catch_reporter_compact.cpp
16115 // start catch_reporter_console.cpp
16116
16117 #include <cfloat>
16118 #include <cstdio>
16119
16120 #if defined(_MSC_VER)
16121 #pragma warning(push)
16122 #pragma warning(disable:4061) // Not all labels are EXPLICITLY handled in switch
16123 // Note that 4062 (not all labels are handled and default is missing) is enabled
16124 #endif
16125
16126 #if defined(__clang__)
16127 # pragma clang diagnostic push
16128 // For simplicity, benchmarking-only helpers are always enabled
16129 # pragma clang diagnostic ignored "-Wunused-function"
16130 #endif
16131
16132 namespace Catch {
16133     namespace {
16134
16135         // Formatter impl for ConsoleReporter
16136         class ConsoleAssertionPrinter {
16137         public:
16138             ConsoleAssertionPrinter& operator= (ConsoleAssertionPrinter const&) = delete;
16139             ConsoleAssertionPrinter(ConsoleAssertionPrinter const&) = delete;
16140             ConsoleAssertionPrinter(std::ostream& _stream, AssertionStats const& _stats, bool
_printInfoMessages)
16141             : stream(_stream),
16142               stats(_stats),
16143               result(_stats.assertionResult),
16144               colour(Colour::None),
16145               message(result.getMessage()),
16146               messages(_stats.infoMessages),
16147               printInfoMessages(_printInfoMessages) {
16148                 printInfoMessages(_printInfoMessages) {
16149                     switch (result.getResultType()) {
16150                         case ResultWas::Ok:
16151                             colour = Colour::Success;
16152                             passOrFail = "PASSED";
16153                             //if( result.hasMessage() )
16154                             if (_stats.infoMessages.size() == 1)
16155                                 messageLabel = "with message";
16156                             if (_stats.infoMessages.size() > 1)
16157                                 messageLabel = "with messages";
16158                             break;
16159                         case ResultWas::ExpressionFailed:
16160                             if (result.isOk()) {
16161                                 colour = Colour::Success;
16162                                 passOrFail = "FAILED - but was ok";
16163                             } else {

```

```

16164         colour = Colour::Error;
16165         passOrFail = "FAILED";
16166     }
16167     if (_stats.infoMessages.size() == 1)
16168         messageLabel = "with message";
16169     if (_stats.infoMessages.size() > 1)
16170         messageLabel = "with messages";
16171     break;
16172 case ResultWas::ThrowException:
16173     colour = Colour::Error;
16174     passOrFail = "FAILED";
16175     messageLabel = "due to unexpected exception with ";
16176     if (_stats.infoMessages.size() == 1)
16177         messageLabel += "message";
16178     if (_stats.infoMessages.size() > 1)
16179         messageLabel += "messages";
16180     break;
16181 case ResultWas::FatalErrorCondition:
16182     colour = Colour::Error;
16183     passOrFail = "FAILED";
16184     messageLabel = "due to a fatal error condition";
16185     break;
16186 case ResultWas::DidntThrowException:
16187     colour = Colour::Error;
16188     passOrFail = "FAILED";
16189     messageLabel = "because no exception was thrown where one was expected";
16190     break;
16191 case ResultWas::Info:
16192     messageLabel = "info";
16193     break;
16194 case ResultWas::Warning:
16195     messageLabel = "warning";
16196     break;
16197 case ResultWas::ExplicitFailure:
16198     passOrFail = "FAILED";
16199     colour = Colour::Error;
16200     if (_stats.infoMessages.size() == 1)
16201         messageLabel = "explicitly with message";
16202     if (_stats.infoMessages.size() > 1)
16203         messageLabel = "explicitly with messages";
16204     break;
16205     // These cases are here to prevent compiler warnings
16206 case ResultWas::Unknown:
16207 case ResultWas::FailureBit:
16208 case ResultWas::Exception:
16209     passOrFail = "*** internal error ***";
16210     colour = Colour::Error;
16211     break;
16212     }
16213 }
16214
16215 void print() const {
16216     printSourceInfo();
16217     if (stats.totals.assertions.total() > 0) {
16218         printResultType();
16219         printOriginalExpression();
16220         printReconstructedExpression();
16221     } else {
16222         stream << '\n';
16223     }
16224     printMessage();
16225 }
16226
16227 private:
16228     void printResultType() const {
16229         if (!passOrFail.empty()) {
16230             Colour colourGuard(colour);
16231             stream << passOrFail << ":\n";
16232         }
16233     }
16234     void printOriginalExpression() const {
16235         if (result.hasExpression()) {
16236             Colour colourGuard(Colour::OriginalExpression);
16237             stream << " ";
16238             stream << result.getExpressionInMacro();
16239             stream << '\n';
16240         }
16241     }
16242     void printReconstructedExpression() const {
16243         if (result.hasExpandedExpression()) {
16244             stream << "with expansion:\n";
16245             Colour colourGuard(Colour::ReconstructedExpression);
16246             stream << Column(result.getExpandedExpression()).indent(2) << '\n';
16247         }
16248     }
16249     void printMessage() const {
16250         if (!messageLabel.empty())

```

```

16251         stream << messageLabel << ':' << '\n';
16252     for (auto const& msg : messages) {
16253         // If this assertion is a warning ignore any INFO messages
16254         if (printInfoMessages || msg.type != ResultWas::Info)
16255             stream << Column(msg.message).indent(2) << '\n';
16256     }
16257 }
16258 void printSourceInfo() const {
16259     Colour colourGuard(Colour::FileName);
16260     stream << result.getSourceInfo() << ": ";
16261 }
16262
16263 std::ostream& stream;
16264 AssertionStats const& stats;
16265 AssertionResult const& result;
16266 Colour::Code colour;
16267 std::string passOrFail;
16268 std::string messageLabel;
16269 std::string message;
16270 std::vector<MessageInfo> messages;
16271 bool printInfoMessages;
16272 };
16273
16274 std::size_t makeRatio(std::size_t number, std::size_t total) {
16275     std::size_t ratio = total > 0 ? CATCH_CONFIG_CONSOLE_WIDTH * number / total : 0;
16276     return (ratio == 0 && number > 0) ? 1 : ratio;
16277 }
16278
16279 std::size_t& findMax(std::size_t& i, std::size_t& j, std::size_t& k) {
16280     if (i > j && i > k)
16281         return i;
16282     else if (j > k)
16283         return j;
16284     else
16285         return k;
16286 }
16287
16288 struct ColumnInfo {
16289     enum Justification { Left, Right };
16290     std::string name;
16291     int width;
16292     Justification justification;
16293 };
16294 struct ColumnBreak {};
16295 struct RowBreak {};
16296
16297 class Duration {
16298     enum class Unit {
16299         Auto,
16300         Nanoseconds,
16301         Microseconds,
16302         Milliseconds,
16303         Seconds,
16304         Minutes
16305     };
16306     static const uint64_t s_nanosecondsInAMicrosecond = 1000;
16307     static const uint64_t s_nanosecondsInAMillisecond = 1000 * s_nanosecondsInAMicrosecond;
16308     static const uint64_t s_nanosecondsInASecond = 1000 * s_nanosecondsInAMillisecond;
16309     static const uint64_t s_nanosecondsInAMinute = 60 * s_nanosecondsInASecond;
16310
16311     double m_inNanoseconds;
16312     Unit m_units;
16313
16314 public:
16315     explicit Duration(double inNanoseconds, Unit units = Unit::Auto)
16316         : m_inNanoseconds(inNanoseconds),
16317           m_units(units) {
16318         if (m_units == Unit::Auto) {
16319             if (m_inNanoseconds < s_nanosecondsInAMicrosecond)
16320                 m_units = Unit::Nanoseconds;
16321             else if (m_inNanoseconds < s_nanosecondsInAMillisecond)
16322                 m_units = Unit::Microseconds;
16323             else if (m_inNanoseconds < s_nanosecondsInASecond)
16324                 m_units = Unit::Milliseconds;
16325             else if (m_inNanoseconds < s_nanosecondsInAMinute)
16326                 m_units = Unit::Seconds;
16327             else
16328                 m_units = Unit::Minutes;
16329         }
16330     }
16331
16332     auto value() const -> double {
16333         switch (m_units) {
16334             case Unit::Microseconds:
16335                 return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMicrosecond);
16336             case Unit::Milliseconds:

```



```

16338         return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMillisecond);
16339     case Unit::Seconds:
16340         return m_inNanoseconds / static_cast<double>(s_nanosecondsInASecond);
16341     case Unit::Minutes:
16342         return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMinute);
16343     default:
16344         return m_inNanoseconds;
16345     }
16346 }
16347 auto unitsAsString() const -> std::string {
16348     switch (m_units) {
16349     case Unit::Nanoseconds:
16350         return "ns";
16351     case Unit::Microseconds:
16352         return "us";
16353     case Unit::Milliseconds:
16354         return "ms";
16355     case Unit::Seconds:
16356         return "s";
16357     case Unit::Minutes:
16358         return "m";
16359     default:
16360         return "*** internal error ***";
16361     }
16362 }
16363 }
16364 friend auto operator << (std::ostream& os, Duration const& duration) -> std::ostream& {
16365     return os << duration.value() << ' ' << duration.unitsAsString();
16366 }
16367 };
16368 } // end anon namespace
16369
16370 class TablePrinter {
16371     std::ostream& m_os;
16372     std::vector<ColumnInfo> m_columnInfos;
16373     std::ostringstream m_oss;
16374     int m_currentColumn = -1;
16375     bool m_isOpen = false;
16376
16377 public:
16378     TablePrinter( std::ostream& os, std::vector<ColumnInfo> columnInfos )
16379         : m_os( os ),
16380           m_columnInfos( std::move( columnInfos ) ) {}
16381
16382     auto columnInfos() const -> std::vector<ColumnInfo> const& {
16383         return m_columnInfos;
16384     }
16385
16386     void open() {
16387         if (!m_isOpen) {
16388             m_isOpen = true;
16389             *this << RowBreak();
16390
16391             Columns headerCols;
16392             Spacer spacer(2);
16393             for (auto const& info : m_columnInfos) {
16394                 headerCols += Column(info.name).width(static_cast<std::size_t>(info.width - 2));
16395                 headerCols += spacer;
16396             }
16397             m_os << headerCols << '\n';
16398
16399             m_os << Catch::getLineOfChars<'-'>() << '\n';
16400         }
16401     }
16402     void close() {
16403         if (m_isOpen) {
16404             *this << RowBreak();
16405             m_os << std::endl;
16406             m_isOpen = false;
16407         }
16408     }
16409
16410     template<typename T>
16411     friend TablePrinter& operator << (TablePrinter& tp, T const& value) {
16412         tp.m_oss << value;
16413         return tp;
16414     }
16415
16416     friend TablePrinter& operator << (TablePrinter& tp, ColumnBreak) {
16417         auto colStr = tp.m_oss.str();
16418         const auto strSize = colStr.size();
16419         tp.m_oss.str("");
16420         tp.open();
16421         if (tp.m_currentColumn == static_cast<int>(tp.m_columnInfos.size() - 1)) {
16422             tp.m_currentColumn = -1;
16423             tp.m_os << '\n';
16424         }

```

```

16425         tp.m_currentColumn++;
16426
16427         auto colInfo = tp.m_columnInfos[tp.m_currentColumn];
16428         auto padding = (strSize + 1 < static_cast<std::size_t>(colInfo.width))
16429             ? std::string(colInfo.width - (strSize + 1), ' ')
16430             : std::string();
16431         if (colInfo.justification == ColumnInfo::Left)
16432             tp.m_os << colStr << padding << ' ';
16433         else
16434             tp.m_os << padding << colStr << ' ';
16435         return tp;
16436     }
16437
16438     friend TablePrinter& operator << (TablePrinter& tp, RowBreak) {
16439         if (tp.m_currentColumn > 0) {
16440             tp.m_os << '\n';
16441             tp.m_currentColumn = -1;
16442         }
16443         return tp;
16444     }
16445 };
16446
16447 ConsoleReporter::ConsoleReporter(ReporterConfig const& config)
16448     : StreamingReporterBase(config),
16449       m_tablePrinter(new TablePrinter(config.stream(),
16450                                       [&config]() -> std::vector<ColumnInfo> {
16451                                           if (config.fullConfig()->benchmarkNoAnalysis())
16452                                               {
16453                                                 return{
16454                                                     { "benchmark name",
16455                                                       CATCH_CONFIG_CONSOLE_WIDTH - 43, ColumnInfo::Left },
16456                                                     { "      samples", 14, ColumnInfo::Right },
16457                                                     { "    iterations", 14, ColumnInfo::Right },
16458                                                     { "           mean", 14, ColumnInfo::Right }
16459                                                 };
16460                                                 else
16461                                                 {
16462                                                 return{
16463                                                     { "benchmark name",
16464                                                       CATCH_CONFIG_CONSOLE_WIDTH - 43, ColumnInfo::Left },
16465                                                     { "samples      mean      std dev", 14,
16466                                                       ColumnInfo::Right },
16467                                                     { "iterations  low mean  low std dev",
16468                                                       14, ColumnInfo::Right },
16469                                                     { "estimated   high mean  high std
16470 dev", 14, ColumnInfo::Right }
16471                                                 };
16472                                                 }
16473                                             }()) {}
16474
16475         ConsoleReporter::~ConsoleReporter() = default;
16476
16477         std::string ConsoleReporter::getDescription() {
16478             return "Reports test results as plain lines of text";
16479         }
16480
16481         void ConsoleReporter::noMatchingTestCases(std::string const& spec) {
16482             stream << "No test cases matched '" << spec << "'\n" << std::endl;
16483         }
16484
16485         void ConsoleReporter::reportInvalidArguments(std::string const&arg){
16486             stream << "Invalid Filter: " << arg << std::endl;
16487         }
16488
16489         void ConsoleReporter::assertionStarting(AssertionInfo const&) {}
16490
16491         bool ConsoleReporter::assertionEnded(AssertionStats const& _assertionStats) {
16492             AssertionResult const& result = _assertionStats.assertionResult;
16493
16494             bool includeResults = m_config->includeSuccessfulResults() || !result.isOk();
16495
16496             // Drop out if result was successful but we're not printing them.
16497             if (!includeResults && result.getResultType() != ResultWas::Warning)
16498                 return false;
16499
16500             lazyPrint();
16501
16502             ConsoleAssertionPrinter printer(stream, _assertionStats, includeResults);
16503             printer.print();
16504             stream << std::endl;
16505             return true;
16506         }
16507
16508         void ConsoleReporter::sectionStarting(SectionInfo const& _sectionInfo) {

```

```

16504         m_tablePrinter->close();
16505         m_headerPrinted = false;
16506         StreamingReporterBase::sectionStarting(_sectionInfo);
16507     }
16508     void ConsoleReporter::sectionEnded(SectionStats const& _sectionStats) {
16509         m_tablePrinter->close();
16510         if (_sectionStats.missingAssertions) {
16511             lazyPrint();
16512             Colour colour(Colour::ResultError);
16513             if (m_sectionStack.size() > 1)
16514                 stream << "\nNo assertions in section";
16515             else
16516                 stream << "\nNo assertions in test case";
16517             stream << " '" << _sectionStats.sectionInfo.name << "'\n" << std::endl;
16518         }
16519         double dur = _sectionStats.durationInSeconds;
16520         if (shouldShowDuration(*m_config, dur)) {
16521             stream << getFormattedDuration(dur) << " s: " << _sectionStats.sectionInfo.name << std::endl;
16522         }
16523         if (m_headerPrinted) {
16524             m_headerPrinted = false;
16525         }
16526         StreamingReporterBase::sectionEnded(_sectionStats);
16527     }
16528
16529     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
16530     void ConsoleReporter::benchmarkPreparing(std::string const& name) {
16531         lazyPrintWithoutClosingBenchmarkTable();
16532
16533         auto nameCol = Column(name).width(static_cast<std::size_t>(m_tablePrinter->columnInfos()[0].width
16534 - 2));
16535         bool firstLine = true;
16536         for (auto line : nameCol) {
16537             if (!firstLine)
16538                 (*m_tablePrinter) << ColumnBreak() << ColumnBreak() << ColumnBreak();
16539             else
16540                 firstLine = false;
16541
16542             (*m_tablePrinter) << line << ColumnBreak();
16543         }
16544     }
16545
16546     void ConsoleReporter::benchmarkStarting(BenchmarkInfo const& info) {
16547         (*m_tablePrinter) << info.samples << ColumnBreak()
16548         << info.iterations << ColumnBreak();
16549         if (!m_config->benchmarkNoAnalysis())
16550             (*m_tablePrinter) << Duration(info.estimatedDuration) << ColumnBreak();
16551     }
16552     void ConsoleReporter::benchmarkEnded(BenchmarkStats<> const& stats) {
16553         if (m_config->benchmarkNoAnalysis())
16554             {
16555                 (*m_tablePrinter) << Duration(stats.mean.point.count()) << ColumnBreak();
16556             }
16557         else
16558             {
16559                 (*m_tablePrinter) << ColumnBreak()
16560                 << Duration(stats.mean.point.count()) << ColumnBreak()
16561                 << Duration(stats.mean.lower_bound.count()) << ColumnBreak()
16562                 << Duration(stats.mean.upper_bound.count()) << ColumnBreak()
16563                 << Duration(stats.standardDeviation.point.count()) << ColumnBreak()
16564                 << Duration(stats.standardDeviation.lower_bound.count()) << ColumnBreak()
16565                 << Duration(stats.standardDeviation.upper_bound.count()) << ColumnBreak() << ColumnBreak() <<
16566                 ColumnBreak() << ColumnBreak() << ColumnBreak();
16567             }
16568
16569     void ConsoleReporter::benchmarkFailed(std::string const& error) {
16570         Colour colour(Colour::Red);
16571         (*m_tablePrinter)
16572             << "Benchmark failed (" << error << ')'
16573             << ColumnBreak() << RowBreak();
16574     }
16575     #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
16576
16577     void ConsoleReporter::testCaseEnded(TestCaseStats const& _testCaseStats) {
16578         m_tablePrinter->close();
16579         StreamingReporterBase::testCaseEnded(_testCaseStats);
16580         m_headerPrinted = false;
16581     }
16582     void ConsoleReporter::testGroupEnded(TestGroupStats const& _testGroupStats) {
16583         if (currentGroupInfo.used) {
16584             printSummaryDivider();
16585             stream << "Summary for group '" << _testGroupStats.groupInfo.name << "':\n";
16586             printTotals(_testGroupStats.totals);
16587             stream << '\n' << std::endl;
16588         }

```

```

16589     StreamingReporterBase::testGroupEnded(_testGroupStats);
16590 }
16591 void ConsoleReporter::testRunEnded(TestRunStats const& _testRunStats) {
16592     printTotalsDivider(_testRunStats.totals);
16593     printTotals(_testRunStats.totals);
16594     stream < std::endl;
16595     StreamingReporterBase::testRunEnded(_testRunStats);
16596 }
16597 void ConsoleReporter::testRunStarting(TestRunInfo const& _testInfo) {
16598     StreamingReporterBase::testRunStarting(_testInfo);
16599     printTestFilters();
16600 }
16601
16602 void ConsoleReporter::lazyPrint() {
16603
16604     m_tablePrinter->close();
16605     lazyPrintWithoutClosingBenchmarkTable();
16606 }
16607
16608 void ConsoleReporter::lazyPrintWithoutClosingBenchmarkTable() {
16609
16610     if (!currentTestRunInfo.used)
16611         lazyPrintRunInfo();
16612     if (!currentGroupInfo.used)
16613         lazyPrintGroupInfo();
16614
16615     if (!m_headerPrinted) {
16616         printTestCaseAndSectionHeader();
16617         m_headerPrinted = true;
16618     }
16619 }
16620 void ConsoleReporter::lazyPrintRunInfo() {
16621     stream < '\n' < getLineOfChars<'~'>() < '\n';
16622     Colour colour(Colour::SecondaryText);
16623     stream < currentTestRunInfo->name
16624             < " is a Catch v" < libraryVersion() < " host application.\n"
16625             < "Run with -? for options\n\n";
16626
16627     if (m_config->rngSeed() != 0)
16628         stream < "Randomness seeded to: " < m_config->rngSeed() < "\n\n";
16629
16630     currentTestRunInfo.used = true;
16631 }
16632 void ConsoleReporter::lazyPrintGroupInfo() {
16633     if (!currentGroupInfo->name.empty() && currentGroupInfo->groupsCounts > 1) {
16634         printClosedHeader("Group: " + currentGroupInfo->name);
16635         currentGroupInfo.used = true;
16636     }
16637 }
16638 void ConsoleReporter::printTestCaseAndSectionHeader() {
16639     assert(!m_sectionStack.empty());
16640     printOpenHeader(currentTestCaseInfo->name);
16641
16642     if (m_sectionStack.size() > 1) {
16643         Colour colourGuard(Colour::Headers);
16644
16645         auto
16646             it = m_sectionStack.begin() + 1, // Skip first section (test case)
16647             itEnd = m_sectionStack.end();
16648         for (; it != itEnd; ++it)
16649             printHeaderString(it->name, 2);
16650     }
16651
16652     SourceLineInfo lineInfo = m_sectionStack.back().lineInfo;
16653
16654     stream < getLineOfChars<'-'>() < '\n';
16655     Colour colourGuard(Colour::FileName);
16656     stream < lineInfo < '\n';
16657     stream < getLineOfChars<'.'>() < '\n' < std::endl;
16658 }
16659
16660 void ConsoleReporter::printClosedHeader(std::string const& _name) {
16661     printOpenHeader(_name);
16662     stream < getLineOfChars<'.'>() < '\n';
16663 }
16664 void ConsoleReporter::printOpenHeader(std::string const& _name) {
16665     stream < getLineOfChars<'-'>() < '\n';
16666     {
16667         Colour colourGuard(Colour::Headers);
16668         printHeaderString(_name);
16669     }
16670 }
16671
16672 // if string has a : in first line will set indent to follow it on
16673 // subsequent lines
16674 void ConsoleReporter::printHeaderString(std::string const& _string, std::size_t indent) {
16675     std::size_t i = _string.find(": ");

```

```

16676         if (i != std::string::npos)
16677             i += 2;
16678         else
16679             i = 0;
16680         stream << Column(_string).indent(indent + i).initialIndent(indent) << '\n';
16681     }
16682
16683     struct SummaryColumn {
16684
16685         SummaryColumn( std::string _label, Colour::Code _colour )
16686             : label( std::move( _label ) ),
16687               colour( _colour ) {}
16688         SummaryColumn addRow( std::size_t count ) {
16689             ReusableStringStream rss;
16690             rss << count;
16691             std::string row = rss.str();
16692             for (auto& oldRow : rows) {
16693                 while (oldRow.size() < row.size())
16694                     oldRow = ' ' + oldRow;
16695                 while (oldRow.size() > row.size())
16696                     row = ' ' + row;
16697             }
16698             rows.push_back(row);
16699             return *this;
16700         }
16701
16702         std::string label;
16703         Colour::Code colour;
16704         std::vector<std::string> rows;
16705     };
16706
16707 void ConsoleReporter::printTotals( Totals const& totals ) {
16708     if (totals.testCases.total() == 0) {
16709         stream << Colour(Colour::Warning) << "No tests ran\n";
16710     } else if (totals.assertions.total() > 0 && totals.testCases.allPassed()) {
16711         stream << Colour(Colour::ResultSuccess) << "All tests passed";
16712         stream << " ("
16713             << pluralise(totals.assertions.passed, "assertion") << " in "
16714             << pluralise(totals.testCases.passed, "test case") << ') '
16715             << '\n';
16716     } else {
16717
16718         std::vector<SummaryColumn> columns;
16719         columns.push_back(SummaryColumn("", Colour::None)
16720             .addRow(totals.testCases.total())
16721             .addRow(totals.assertions.total()));
16722         columns.push_back(SummaryColumn("passed", Colour::Success)
16723             .addRow(totals.testCases.passed)
16724             .addRow(totals.assertions.passed));
16725         columns.push_back(SummaryColumn("failed", Colour::ResultError)
16726             .addRow(totals.testCases.failed)
16727             .addRow(totals.assertions.failed));
16728         columns.push_back(SummaryColumn("failed as expected", Colour::ResultExpectedFailure)
16729             .addRow(totals.testCases.failedButOk)
16730             .addRow(totals.assertions.failedButOk));
16731
16732         printSummaryRow("test cases", columns, 0);
16733         printSummaryRow("assertions", columns, 1);
16734     }
16735 }
16736
16737 void ConsoleReporter::printSummaryRow(std::string const& label, std::vector<SummaryColumn> const&
cols, std::size_t row) {
16738     for (auto col : cols) {
16739         std::string value = col.rows[row];
16740         if (col.label.empty()) {
16741             stream << label << ": ";
16742             if (value != "0") {
16743                 stream << value;
16744             } else
16745                 stream << Colour(Colour::Warning) << "- none -";
16746         } else if (value != "0") {
16747             stream << Colour(Colour::LightGrey) << " | ";
16748             stream << Colour(col.colour)
16749                 << value << ' ' << col.label;
16750         }
16751     }
16752     stream << '\n';
16753 }
16754
16755 void ConsoleReporter::printTotalsDivider(Totals const& totals) {
16756     if (totals.testCases.total() > 0) {
16757         std::size_t failedRatio = makeRatio(totals.testCases.failed, totals.testCases.total());
16758         std::size_t failedButOkRatio = makeRatio(totals.testCases.failedButOk,
totals.testCases.total());
16759         std::size_t passedRatio = makeRatio(totals.testCases.passed, totals.testCases.total());
16760         while (failedRatio + failedButOkRatio + passedRatio < CATCH_CONFIG_CONSOLE_WIDTH - 1)

```

```

16761         findMax(failedRatio, failedButOkRatio, passedRatio)++;
16762     while (failedRatio + failedButOkRatio + passedRatio > CATCH_CONFIG_CONSOLE_WIDTH - 1)
16763         findMax(failedRatio, failedButOkRatio, passedRatio)--;
16764
16765     stream << Colour(Colour::Error) << std::string(failedRatio, '=');
16766     stream << Colour(Colour::ResultExpectedFailure) << std::string(failedButOkRatio, '=');
16767     if (totals.testCases.allPassed())
16768         stream << Colour(Colour::ResultSuccess) << std::string(passedRatio, '=');
16769     else
16770         stream << Colour(Colour::Success) << std::string(passedRatio, '=');
16771 } else {
16772     stream << Colour(Colour::Warning) << std::string(CATCH_CONFIG_CONSOLE_WIDTH - 1, '=');
16773 }
16774 stream << '\n';
16775 }
16776 void ConsoleReporter::printSummaryDivider() {
16777     stream << getLineOfChars<'-'>() << '\n';
16778 }
16779
16780 void ConsoleReporter::printTestFilters() {
16781     if (m_config->testSpec().hasFilters()) {
16782         Colour guard(Colour::BrightYellow);
16783         stream << "Filters: " << serializeFilters(m_config->getTestsOrTags()) << '\n';
16784     }
16785 }
16786
16787 CATCH_REGISTER_REPORTER("console", ConsoleReporter)
16788
16789 } // end namespace Catch
16790
16791 #if defined(_MSC_VER)
16792 #pragma warning(pop)
16793 #endif
16794
16795 #if defined(__clang__)
16796 #pragma clang diagnostic pop
16797 #endif
16798 // end catch_reporter_console.cpp
16799 // start catch_reporter_junit.cpp
16800
16801 #include <cassert>
16802 #include <sstream>
16803 #include <ctime>
16804 #include <algorithm>
16805 #include <iomanip>
16806
16807 namespace Catch {
16808     namespace {
16809         std::string getCurrentTimestamp() {
16810             // Beware, this is not reentrant because of backward compatibility issues
16811             // Also, UTC only, again because of backward compatibility issues (%z is C++11)
16812             time_t rawtime;
16813             std::time(&rawtime);
16814             auto const timeStampSize = sizeof("2017-01-16T17:06:45Z");
16815
16816 #ifdef _MSC_VER
16817             std::tm timeInfo = {};
16818             gmtime_s(&timeInfo, &rawtime);
16819 #else
16820             std::tm* timeInfo;
16821             timeInfo = std::gmtime(&rawtime);
16822 #endif
16823
16824             char timeStamp[timeStampSize];
16825             const char * const fmt = "%Y-%m-%dT%H:%M:%SZ";
16826
16827 #ifdef _MSC_VER
16828             std::strftime(timeStamp, timeStampSize, fmt, &timeInfo);
16829 #else
16830             std::strftime(timeStamp, timeStampSize, fmt, timeInfo);
16831 #endif
16832             return std::string(timeStamp, timeStampSize-1);
16833         }
16834     }
16835
16836     std::string fileNameTag(const std::vector<std::string> &tags) {
16837         auto it = std::find_if(begin(tags),
16838                               end(tags),
16839                               [](const std::string &tag) { return tag.front() == '#'; });
16840         if (it != tags.end())
16841             return it->substr(1);
16842         return std::string();
16843     }
16844
16845     // Formats the duration in seconds to 3 decimal places.
16846     // This is done because some genius defined Maven Surefire schema
16847     // in a way that only accepts 3 decimal places, and tools like

```

```

16848         // Jenkins use that schema for validation JUnit reporter output.
16849         std::string formatDuration( double seconds ) {
16850             ReusableStringStream rss;
16851             rss << std::fixed << std::setprecision( 3 ) << seconds;
16852             return rss.str();
16853         }
16854     } // anonymous namespace
16855
16856     JUnitReporter::JUnitReporter( ReporterConfig const& _config )
16857         : CumulativeReporterBase( _config ),
16858           xml( _config.stream() )
16859     {
16860         m_reporterPrefs.shouldRedirectStdOut = true;
16861         m_reporterPrefs.shouldReportAllAssertions = true;
16862     }
16863
16864     JUnitReporter::~JUnitReporter() {}
16865
16866     std::string JUnitReporter::getDescription() {
16867         return "Reports test results in an XML format that looks like Ant's junitreport target";
16868     }
16869
16870     void JUnitReporter::noMatchingTestCases( std::string const& /*spec*/ ) {}
16871
16872     void JUnitReporter::testRunStarting( TestRunInfo const& runInfo ) {
16873         CumulativeReporterBase::testRunStarting( runInfo );
16874         xml.startElement( "testsuites" );
16875     }
16876
16877     void JUnitReporter::testGroupStarting( GroupInfo const& groupInfo ) {
16878         suiteTimer.start();
16879         stdOutForSuite.clear();
16880         stdErrForSuite.clear();
16881         unexpectedExceptions = 0;
16882         CumulativeReporterBase::testGroupStarting( groupInfo );
16883     }
16884
16885     void JUnitReporter::testCaseStarting( TestCaseInfo const& testCaseInfo ) {
16886         m_okToFail = testCaseInfo.okToFail();
16887     }
16888
16889     bool JUnitReporter::assertionEnded( AssertionStats const& assertionStats ) {
16890         if( assertionStats.assertionResult.getResultType() == ResultWas::ThrewException && !m_okToFail )
16891             unexpectedExceptions++;
16892         return CumulativeReporterBase::assertionEnded( assertionStats );
16893     }
16894
16895     void JUnitReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
16896         stdOutForSuite += testCaseStats.stdOut;
16897         stdErrForSuite += testCaseStats.stdErr;
16898         CumulativeReporterBase::testCaseEnded( testCaseStats );
16899     }
16900
16901     void JUnitReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
16902         double suiteTime = suiteTimer.getElapsedSeconds();
16903         CumulativeReporterBase::testGroupEnded( testGroupStats );
16904         writeGroup( *m_testGroups.back(), suiteTime );
16905     }
16906
16907     void JUnitReporter::testRunEndedCumulative() {
16908         xml.endElement();
16909     }
16910
16911     void JUnitReporter::writeGroup( TestGroupNode const& groupNode, double suiteTime ) {
16912         XmlWriter::ScopedElement e = xml.scopedElement( "testsuite" );
16913
16914         TestGroupStats const& stats = groupNode.value;
16915         xml.writeAttribute( "name", stats.groupInfo.name );
16916         xml.writeAttribute( "errors", unexpectedExceptions );
16917         xml.writeAttribute( "failures", stats.totals.assertions.failed-unexpectedExceptions );
16918         xml.writeAttribute( "tests", stats.totals.assertions.total() );
16919         xml.writeAttribute( "hostname", "tbd" ); // TBD
16920         if( m_config->showDurations() == ShowDurations::Never )
16921             xml.writeAttribute( "time", "" );
16922         else
16923             xml.writeAttribute( "time", formatDuration( suiteTime ) );
16924         xml.writeAttribute( "timestamp", getCurrentTimestamp() );
16925
16926         // Write properties if there are any
16927         if( m_config->hasTestFilters() || m_config->rngSeed() != 0 ) {
16928             auto properties = xml.scopedElement( "properties" );
16929             if( m_config->hasTestFilters() ) {
16930                 xml.scopedElement( "property" )
16931                     .writeAttribute( "name", "filters" )
16932                     .writeAttribute( "value", serializeFilters( m_config->getTestsOrTags() ) );
16933             }

```

```

16934         }
16935         if (m_config->rngSeed() != 0) {
16936             xml.scopedElement("property")
16937                 .writeAttribute("name", "random-seed")
16938                 .writeAttribute("value", m_config->rngSeed());
16939         }
16940     }
16941
16942     // Write test cases
16943     for( auto const& child : groupNode.children )
16944         writeTestCase( *child );
16945
16946     xml.scopedElement( "system-out" ).writeText( trim( stdoutForSuite ), XmlFormatting::Newline );
16947     xml.scopedElement( "system-err" ).writeText( trim( stderrForSuite ), XmlFormatting::Newline );
16948 }
16949
16950 void JUnitReporter::writeTestCase( TestCaseNode const& testCaseNode ) {
16951     TestCaseStats const& stats = testCaseNode.value;
16952
16953     // All test cases have exactly one section - which represents the
16954     // test case itself. That section may have 0-n nested sections
16955     assert( testCaseNode.children.size() == 1 );
16956     SectionNode const& rootSection = *testCaseNode.children.front();
16957
16958     std::string className = stats.testInfo.className;
16959
16960     if( className.empty() ) {
16961         className = fileNameTag(stats.testInfo.tags);
16962         if ( className.empty() )
16963             className = "global";
16964     }
16965
16966     if ( !m_config->name().empty() )
16967         className = m_config->name() + "." + className;
16968
16969     writeSection( className, "", rootSection, stats.testInfo.okToFail() );
16970 }
16971
16972 void JUnitReporter::writeSection( std::string const& className,
16973                                 std::string const& rootName,
16974                                 SectionNode const& sectionNode,
16975                                 bool testOkToFail ) {
16976     std::string name = trim( sectionNode.stats.sectionInfo.name );
16977     if( !rootName.empty() )
16978         name = rootName + '/' + name;
16979
16980     if( !sectionNode.assertions.empty() ||
16981         !sectionNode.stdout.empty() ||
16982         !sectionNode.stderr.empty() ) {
16983         XmlWriter::ScopedElement e = xml.scopedElement( "testcase" );
16984         if( className.empty() ) {
16985             xml.writeAttribute( "classname", name );
16986             xml.writeAttribute( "name", "root" );
16987         }
16988         else {
16989             xml.writeAttribute( "classname", className );
16990             xml.writeAttribute( "name", name );
16991         }
16992         xml.writeAttribute( "time", formatDuration( sectionNode.stats.durationInSeconds ) );
16993         // This is not ideal, but it should be enough to mimic gtest's
16994         // junit output.
16995         // Ideally the JUnit reporter would also handle `skipTest`
16996         // events and write those out appropriately.
16997         xml.writeAttribute( "status", "run" );
16998
16999         if (sectionNode.stats.assertions.failedButOk) {
17000             xml.scopedElement("skipped")
17001                 .writeAttribute("message", "TEST_CASE tagged with !mayfail");
17002         }
17003
17004         writeAssertions( sectionNode );
17005
17006         if( !sectionNode.stdout.empty() )
17007             xml.scopedElement( "system-out" ).writeText( trim( sectionNode.stdout ),
17008                                                         XmlFormatting::Newline );
17009         if( !sectionNode.stderr.empty() )
17010             xml.scopedElement( "system-err" ).writeText( trim( sectionNode.stderr ),
17011                                                         XmlFormatting::Newline );
17012     }
17013     for( auto const& childNode : sectionNode.childSections )
17014         if( className.empty() )
17015             writeSection( name, "", *childNode, testOkToFail );
17016         else
17017             writeSection( className, name, *childNode, testOkToFail );
17018 }
17019
17020 void JUnitReporter::writeAssertions( SectionNode const& sectionNode ) {

```



```

17019         for( auto const& assertion : sectionNode.assertions )
17020             writeAssertion( assertion );
17021     }
17022
17023     void JunitReporter::writeAssertion( AssertionStats const& stats ) {
17024         AssertionResult const& result = stats.assertionResult;
17025         if( !result.isOk() ) {
17026             std::string elementName;
17027             switch( result.getResultType() ) {
17028                 case ResultWas::ThrewException:
17029                 case ResultWas::FatalErrorCondition:
17030                     elementName = "error";
17031                     break;
17032                 case ResultWas::ExplicitFailure:
17033                 case ResultWas::ExpressionFailed:
17034                 case ResultWas::DidntThrowException:
17035                     elementName = "failure";
17036                     break;
17037
17038                 // We should never see these here:
17039                 case ResultWas::Info:
17040                 case ResultWas::Warning:
17041                 case ResultWas::Ok:
17042                 case ResultWas::Unknown:
17043                 case ResultWas::FailureBit:
17044                 case ResultWas::Exception:
17045                     elementName = "internalError";
17046                     break;
17047             }
17048
17049             XmlWriter::ScopedElement e = xml.scopedElement( elementName );
17050
17051             xml.writeAttribute( "message", result.getExpression() );
17052             xml.writeAttribute( "type", result.getTestMacroName() );
17053
17054             ReusableStringStream rss;
17055             if (stats.totals.assertions.total() > 0) {
17056                 rss << "FAILED" << ":\n";
17057                 if (result.hasExpression()) {
17058                     rss << " ";
17059                     rss << result.getExpressionInMacro();
17060                     rss << '\n';
17061                 }
17062                 if (result.hasExpandedExpression()) {
17063                     rss << "with expansion:\n";
17064                     rss << Column(result.getExpandedExpression()).indent(2) << '\n';
17065                 }
17066             } else {
17067                 rss << '\n';
17068             }
17069
17070             if( !result.getMessage().empty() )
17071                 rss << result.getMessage() << '\n';
17072             for( auto const& msg : stats.infoMessages )
17073                 if( msg.type == ResultWas::Info )
17074                     rss << msg.message << '\n';
17075
17076             rss << "at " << result.getSourceInfo();
17077             xml.writeText( rss.str(), XmlFormatting::Newline );
17078         }
17079     }
17080
17081     CATCH_REGISTER_REPORTER( "junit", JunitReporter )
17082
17083 } // end namespace Catch
17084 // end catch_reporter_junit.cpp
17085 // start catch_reporter_listening.cpp
17086
17087 #include <cassert>
17088
17089 namespace Catch {
17090
17091     ListeningReporter::ListeningReporter() {
17092         // We will assume that listeners will always want all assertions
17093         m_preferences.shouldReportAllAssertions = true;
17094     }
17095
17096     void ListeningReporter::addListener( IStreamingReporterPtr&& listener ) {
17097         m_listeners.push_back( std::move( listener ) );
17098     }
17099
17100     void ListeningReporter::addReporter( IStreamingReporterPtr&& reporter ) {
17101         assert(!m_reporter && "Listening reporter can wrap only 1 real reporter");
17102         m_reporter = std::move( reporter );
17103         m_preferences.shouldRedirectStdOut = m_reporter->getPreferences().shouldRedirectStdOut;
17104     }
17105

```

```

17106 ReporterPreferences ListeningReporter::getPreferences() const {
17107     return m_preferences;
17108 }
17109
17110 std::set<Verbosity> ListeningReporter::getSupportedVerbsities() {
17111     return std::set<Verbosity>{};
17112 }
17113
17114 void ListeningReporter::noMatchingTestCases( std::string const& spec ) {
17115     for ( auto const& listener : m_listeners ) {
17116         listener->noMatchingTestCases( spec );
17117     }
17118     m_reporter->noMatchingTestCases( spec );
17119 }
17120
17121 void ListeningReporter::reportInvalidArguments( std::string const& arg ) {
17122     for ( auto const& listener : m_listeners ) {
17123         listener->reportInvalidArguments( arg );
17124     }
17125     m_reporter->reportInvalidArguments( arg );
17126 }
17127
17128 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
17129 void ListeningReporter::benchmarkPreparing( std::string const& name ) {
17130     for ( auto const& listener : m_listeners ) {
17131         listener->benchmarkPreparing( name );
17132     }
17133     m_reporter->benchmarkPreparing( name );
17134 }
17135 void ListeningReporter::benchmarkStarting( BenchmarkInfo const& benchmarkInfo ) {
17136     for ( auto const& listener : m_listeners ) {
17137         listener->benchmarkStarting( benchmarkInfo );
17138     }
17139     m_reporter->benchmarkStarting( benchmarkInfo );
17140 }
17141 void ListeningReporter::benchmarkEnded( BenchmarkStats<> const& benchmarkStats ) {
17142     for ( auto const& listener : m_listeners ) {
17143         listener->benchmarkEnded( benchmarkStats );
17144     }
17145     m_reporter->benchmarkEnded( benchmarkStats );
17146 }
17147
17148 void ListeningReporter::benchmarkFailed( std::string const& error ) {
17149     for ( auto const& listener : m_listeners ) {
17150         listener->benchmarkFailed( error );
17151     }
17152     m_reporter->benchmarkFailed( error );
17153 }
17154 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
17155
17156 void ListeningReporter::testRunStarting( TestRunInfo const& testRunInfo ) {
17157     for ( auto const& listener : m_listeners ) {
17158         listener->testRunStarting( testRunInfo );
17159     }
17160     m_reporter->testRunStarting( testRunInfo );
17161 }
17162
17163 void ListeningReporter::testGroupStarting( GroupInfo const& groupInfo ) {
17164     for ( auto const& listener : m_listeners ) {
17165         listener->testGroupStarting( groupInfo );
17166     }
17167     m_reporter->testGroupStarting( groupInfo );
17168 }
17169
17170 void ListeningReporter::testCaseStarting( TestCaseInfo const& testInfo ) {
17171     for ( auto const& listener : m_listeners ) {
17172         listener->testCaseStarting( testInfo );
17173     }
17174     m_reporter->testCaseStarting( testInfo );
17175 }
17176
17177 void ListeningReporter::sectionStarting( SectionInfo const& sectionInfo ) {
17178     for ( auto const& listener : m_listeners ) {
17179         listener->sectionStarting( sectionInfo );
17180     }
17181     m_reporter->sectionStarting( sectionInfo );
17182 }
17183
17184 void ListeningReporter::assertionStarting( AssertionInfo const& assertionInfo ) {
17185     for ( auto const& listener : m_listeners ) {
17186         listener->assertionStarting( assertionInfo );
17187     }
17188     m_reporter->assertionStarting( assertionInfo );
17189 }
17190
17191 // The return value indicates if the messages buffer should be cleared:
17192 bool ListeningReporter::assertionEnded( AssertionStats const& assertionStats ) {

```

```

17193         for( auto const& listener : m_listeners ) {
17194             static_cast<void>( listener->assertionEnded( assertionStats ) );
17195         }
17196         return m_reporter->assertionEnded( assertionStats );
17197     }
17198
17199     void ListeningReporter::sectionEnded( SectionStats const& sectionStats ) {
17200         for ( auto const& listener : m_listeners ) {
17201             listener->sectionEnded( sectionStats );
17202         }
17203         m_reporter->sectionEnded( sectionStats );
17204     }
17205
17206     void ListeningReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
17207         for ( auto const& listener : m_listeners ) {
17208             listener->testCaseEnded( testCaseStats );
17209         }
17210         m_reporter->testCaseEnded( testCaseStats );
17211     }
17212
17213     void ListeningReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
17214         for ( auto const& listener : m_listeners ) {
17215             listener->testGroupEnded( testGroupStats );
17216         }
17217         m_reporter->testGroupEnded( testGroupStats );
17218     }
17219
17220     void ListeningReporter::testRunEnded( TestRunStats const& testRunStats ) {
17221         for ( auto const& listener : m_listeners ) {
17222             listener->testRunEnded( testRunStats );
17223         }
17224         m_reporter->testRunEnded( testRunStats );
17225     }
17226
17227     void ListeningReporter::skipTest( TestCaseInfo const& testInfo ) {
17228         for ( auto const& listener : m_listeners ) {
17229             listener->skipTest( testInfo );
17230         }
17231         m_reporter->skipTest( testInfo );
17232     }
17233
17234     bool ListeningReporter::isMulti() const {
17235         return true;
17236     }
17237
17238 } // end namespace Catch
17239 // end catch_reporter_listening.cpp
17240 // start catch_reporter_xml.cpp
17241
17242 #if defined(_MSC_VER)
17243 #pragma warning(push)
17244 #pragma warning(disable:4061) // Not all labels are EXPLICITLY handled in switch
17245                             // Note that 4062 (not all labels are handled
17246                             // and default is missing) is enabled
17247 #endif
17248
17249 namespace Catch {
17250     XmlReporter::XmlReporter( ReporterConfig const& _config )
17251         :   StreamingReporterBase( _config ),
17252             m_xml(_config.stream())
17253     {
17254         m_reporterPrefs.shouldRedirectStdOut = true;
17255         m_reporterPrefs.shouldReportAllAssertions = true;
17256     }
17257
17258     XmlReporter::~XmlReporter() = default;
17259
17260     std::string XmlReporter::getDescription() {
17261         return "Reports test results as an XML document";
17262     }
17263
17264     std::string XmlReporter::getStylesheetRef() const {
17265         return std::string();
17266     }
17267
17268     void XmlReporter::writeSourceInfo( SourceLineInfo const& sourceInfo ) {
17269         m_xml
17270             .writeAttribute( "filename", sourceInfo.file )
17271             .writeAttribute( "line", sourceInfo.line );
17272     }
17273
17274     void XmlReporter::noMatchingTestCases( std::string const& s ) {
17275         StreamingReporterBase::noMatchingTestCases( s );
17276     }
17277
17278     void XmlReporter::testRunStarting( TestRunInfo const& testInfo ) {
17279         StreamingReporterBase::testRunStarting( testInfo );

```

```

17280         std::string stylesheetRef = getStylesheetRef();
17281         if ( !stylesheetRef.empty() )
17282             m_xml.writeStylesheetRef( stylesheetRef );
17283         m_xml.startElement( "Catch" );
17284         if ( !m_config->name().empty() )
17285             m_xml.writeAttribute( "name", m_config->name() );
17286         if ( m_config->testSpec().hasFilters() )
17287             m_xml.writeAttribute( "filters", serializeFilters( m_config->getTestsOrTags() ) );
17288         if ( m_config->rngSeed() != 0 )
17289             m_xml.scopedElement( "Randomness" )
17290                 .writeAttribute( "seed", m_config->rngSeed() );
17291     }
17292
17293     void XmlReporter::testGroupStarting( GroupInfo const& groupInfo ) {
17294         StreamingReporterBase::testGroupStarting( groupInfo );
17295         m_xml.startElement( "Group" );
17296         .writeAttribute( "name", groupInfo.name );
17297     }
17298
17299     void XmlReporter::testCaseStarting( TestCaseInfo const& testInfo ) {
17300         StreamingReporterBase::testCaseStarting( testInfo );
17301         m_xml.startElement( "TestCase" )
17302             .writeAttribute( "name", trim( testInfo.name ) )
17303             .writeAttribute( "description", testInfo.description )
17304             .writeAttribute( "tags", testInfo.tagsAsString() );
17305
17306         writeSourceInfo( testInfo.lineInfo );
17307
17308         if ( m_config->showDurations() == ShowDurations::Always )
17309             m_testCaseTimer.start();
17310         m_xml.ensureTagClosed();
17311     }
17312
17313     void XmlReporter::sectionStarting( SectionInfo const& sectionInfo ) {
17314         StreamingReporterBase::sectionStarting( sectionInfo );
17315         if ( m_sectionDepth++ > 0 ) {
17316             m_xml.startElement( "Section" )
17317                 .writeAttribute( "name", trim( sectionInfo.name ) );
17318             writeSourceInfo( sectionInfo.lineInfo );
17319             m_xml.ensureTagClosed();
17320         }
17321     }
17322
17323     void XmlReporter::assertionStarting( AssertionInfo const& ) { }
17324
17325     bool XmlReporter::assertionEnded( AssertionStats const& assertionStats ) {
17326
17327         AssertionResult const& result = assertionStats.assertionResult;
17328
17329         bool includeResults = m_config->includeSuccessfulResults() || !result.isOk();
17330
17331         if ( includeResults || result.getResultType() == ResultWas::Warning ) {
17332             // Print any info messages in <Info> tags.
17333             for( auto const& msg : assertionStats.infoMessages ) {
17334                 if ( msg.type == ResultWas::Info && includeResults ) {
17335                     m_xml.scopedElement( "Info" )
17336                         .writeText( msg.message );
17337                 } else if ( msg.type == ResultWas::Warning ) {
17338                     m_xml.scopedElement( "Warning" )
17339                         .writeText( msg.message );
17340                 }
17341             }
17342         }
17343
17344         // Drop out if result was successful but we're not printing them.
17345         if ( !includeResults && result.getResultType() != ResultWas::Warning )
17346             return true;
17347
17348         // Print the expression if there is one.
17349         if ( result.hasExpression() ) {
17350             m_xml.startElement( "Expression" )
17351                 .writeAttribute( "success", result.succeeded() )
17352                 .writeAttribute( "type", result.getTestMacroName() );
17353
17354             writeSourceInfo( result.getSourceInfo() );
17355
17356             m_xml.scopedElement( "Original" )
17357                 .writeText( result.getExpression() );
17358             m_xml.scopedElement( "Expanded" )
17359                 .writeText( result.getExpandedExpression() );
17360         }
17361
17362         // And... Print a result applicable to each result type.
17363         switch( result.getResultType() ) {
17364             case ResultWas::ThrowException:
17365                 m_xml.startElement( "Exception" );
17366                 writeSourceInfo( result.getSourceInfo() );

```

```

17367         m_xml.writeText( result.getMessage() );
17368         m_xml.endElement();
17369         break;
17370     case ResultWas::FatalErrorCondition:
17371         m_xml.startElement( "FatalErrorCondition" );
17372         writeSourceInfo( result.getSourceInfo() );
17373         m_xml.writeText( result.getMessage() );
17374         m_xml.endElement();
17375         break;
17376     case ResultWas::Info:
17377         m_xml.scopedElement( "Info" )
17378             .writeText( result.getMessage() );
17379         break;
17380     case ResultWas::Warning:
17381         // Warning will already have been written
17382         break;
17383     case ResultWas::ExplicitFailure:
17384         m_xml.startElement( "Failure" );
17385         writeSourceInfo( result.getSourceInfo() );
17386         m_xml.writeText( result.getMessage() );
17387         m_xml.endElement();
17388         break;
17389     default:
17390         break;
17391 }
17392
17393 if( result.hasExpression() )
17394     m_xml.endElement();
17395
17396 return true;
17397 }
17398
17399 void XmlReporter::sectionEnded( SectionStats const& sectionStats ) {
17400     StreamingReporterBase::sectionEnded( sectionStats );
17401     if( --m_sectionDepth > 0 ) {
17402         XmlWriter::ScopedElement e = m_xml.scopedElement( "OverallResults" );
17403         e.writeAttribute( "successes", sectionStats.assertions.passed );
17404         e.writeAttribute( "failures", sectionStats.assertions.failed );
17405         e.writeAttribute( "expectedFailures", sectionStats.assertions.failedButOk );
17406
17407         if ( m_config->showDurations() == ShowDurations::Always )
17408             e.writeAttribute( "durationInSeconds", sectionStats.durationInSeconds );
17409
17410         m_xml.endElement();
17411     }
17412 }
17413
17414 void XmlReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
17415     StreamingReporterBase::testCaseEnded( testCaseStats );
17416     XmlWriter::ScopedElement e = m_xml.scopedElement( "OverallResult" );
17417     e.writeAttribute( "success", testCaseStats.totals.assertions.allOk() );
17418
17419     if ( m_config->showDurations() == ShowDurations::Always )
17420         e.writeAttribute( "durationInSeconds", m_testCaseTimer.getElapsedSeconds() );
17421
17422     if( !testCaseStats.stdOut.empty() )
17423         m_xml.scopedElement( "StdOut" ).writeText( trim( testCaseStats.stdOut ),
17424             XmlFormatting::Newline );
17425     if( !testCaseStats.stdErr.empty() )
17426         m_xml.scopedElement( "StdErr" ).writeText( trim( testCaseStats.stdErr ),
17427             XmlFormatting::Newline );
17428
17429     m_xml.endElement();
17430 }
17431
17432 void XmlReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
17433     StreamingReporterBase::testGroupEnded( testGroupStats );
17434     // TODO: Check testGroupStats.aborting and act accordingly.
17435     m_xml.scopedElement( "OverallResults" )
17436         .writeAttribute( "successes", testGroupStats.totals.assertions.passed )
17437         .writeAttribute( "failures", testGroupStats.totals.assertions.failed )
17438         .writeAttribute( "expectedFailures", testGroupStats.totals.assertions.failedButOk );
17439     m_xml.scopedElement( "OverallResultsCases" )
17440         .writeAttribute( "successes", testGroupStats.totals.testCases.passed )
17441         .writeAttribute( "failures", testGroupStats.totals.testCases.failed )
17442         .writeAttribute( "expectedFailures", testGroupStats.totals.testCases.failedButOk );
17443     m_xml.endElement();
17444 }
17445
17446 void XmlReporter::testRunEnded( TestRunStats const& testRunStats ) {
17447     StreamingReporterBase::testRunEnded( testRunStats );
17448     m_xml.scopedElement( "OverallResults" )
17449         .writeAttribute( "successes", testRunStats.totals.assertions.passed )
17450         .writeAttribute( "failures", testRunStats.totals.assertions.failed )
17451         .writeAttribute( "expectedFailures", testRunStats.totals.assertions.failedButOk );
17452     m_xml.scopedElement( "OverallResultsCases" )
17453         .writeAttribute( "successes", testRunStats.totals.testCases.passed )

```

```

17452         .writeAttribute( "failures", testRunStats.totals.testCases.failed )
17453         .writeAttribute( "expectedFailures", testRunStats.totals.testCases.failedButOk );
17454     m_xml.endElement();
17455 }
17456
17457 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
17458 void XmlReporter::benchmarkPreparing(std::string const& name) {
17459     m_xml.startElement("BenchmarkResults")
17460         .writeAttribute("name", name);
17461 }
17462
17463 void XmlReporter::benchmarkStarting(BenchmarkInfo const &info) {
17464     m_xml.writeAttribute("samples", info.samples)
17465         .writeAttribute("resamples", info.resamples)
17466         .writeAttribute("iterations", info.iterations)
17467         .writeAttribute("clockResolution", info.clockResolution)
17468         .writeAttribute("estimatedDuration", info.estimatedDuration)
17469         .writeComment("All values in nano seconds");
17470 }
17471
17472 void XmlReporter::benchmarkEnded(BenchmarkStats<> const& benchmarkStats) {
17473     m_xml.startElement("mean")
17474         .writeAttribute("value", benchmarkStats.mean.point.count())
17475         .writeAttribute("lowerBound", benchmarkStats.mean.lower_bound.count())
17476         .writeAttribute("upperBound", benchmarkStats.mean.upper_bound.count())
17477         .writeAttribute("ci", benchmarkStats.mean.confidence_interval);
17478     m_xml.endElement();
17479     m_xml.startElement("standardDeviation")
17480         .writeAttribute("value", benchmarkStats.standardDeviation.point.count())
17481         .writeAttribute("lowerBound", benchmarkStats.standardDeviation.lower_bound.count())
17482         .writeAttribute("upperBound", benchmarkStats.standardDeviation.upper_bound.count())
17483         .writeAttribute("ci", benchmarkStats.standardDeviation.confidence_interval);
17484     m_xml.endElement();
17485     m_xml.startElement("outliers")
17486         .writeAttribute("variance", benchmarkStats.outlierVariance)
17487         .writeAttribute("lowMild", benchmarkStats.outliers.low_mild)
17488         .writeAttribute("lowSevere", benchmarkStats.outliers.low_severe)
17489         .writeAttribute("highMild", benchmarkStats.outliers.high_mild)
17490         .writeAttribute("highSevere", benchmarkStats.outliers.high_severe);
17491     m_xml.endElement();
17492     m_xml.endElement();
17493 }
17494
17495 void XmlReporter::benchmarkFailed(std::string const &error) {
17496     m_xml.scopedElement("failed")
17497         .writeAttribute("message", error);
17498     m_xml.endElement();
17499 }
17500 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
17501
17502 CATCH_REGISTER_REPORTER( "xml", XmlReporter )
17503
17504 } // end namespace Catch
17505
17506 #if defined(_MSC_VER)
17507 #pragma warning(pop)
17508 #endif
17509 // end catch_reporter_xml.cpp
17510
17511 namespace Catch {
17512     LeakDetector leakDetector;
17513 }
17514
17515 #ifdef __clang__
17516 #pragma clang diagnostic pop
17517 #endif
17518
17519 // end catch_impl.hpp
17520 #endif
17521
17522 #ifndef CATCH_CONFIG_MAIN
17523 // start catch_default_main.hpp
17524
17525 #ifndef __OBJC__
17526 #ifndef CATCH_INTERNAL_CDECL
17527 #ifndef _MSC_VER
17528 #define CATCH_INTERNAL_CDECL __cdecl
17529 #else
17530 #define CATCH_INTERNAL_CDECL
17531 #endif
17532 #endif
17533 #endif
17534
17535 #if defined(CATCH_CONFIG_WCHAR) && defined(CATCH_PLATFORM_WINDOWS) && defined(_UNICODE) &&
!defined(DO_NOT_USE_WMAIN)
17536 // Standard C/C++ Win32 Unicode wmain entry point
17537 extern "C" int CATCH_INTERNAL_CDECL wmain (int argc, wchar_t * argv[], wchar_t * []) {

```

```

17538 #else
17539 // Standard C/C++ main entry point
17540 int CATCH_INTERNAL_CDECL main( int argc, char * argv[] ) {
17541 #endif
17542
17543     return Catch::Session().run( argc, argv );
17544 }
17545
17546 #else // __OBJC__
17547 // Objective-C entry point
17548 int main( int argc, char * const argv[] ) {
17549 #if !CATCH_ARC_ENABLED
17550     NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
17551 #endif
17552
17553     Catch::registerTestMethods();
17554     int result = Catch::Session().run( argc, (char**)argv );
17555
17556 #if !CATCH_ARC_ENABLED
17557     [pool drain];
17558 #endif
17559
17560     return result;
17561 }
17562 #endif // __OBJC__
17563
17564 // end catch_default_main.hpp
17565 #endif
17566
17567 #if !defined(CATCH_CONFIG_IMPL_ONLY)
17568
17569 #ifdef CLARA_CONFIG_MAIN_NOT_DEFINED
17570 #undef CLARA_CONFIG_MAIN
17571 #endif
17572
17573 #if !defined(CATCH_CONFIG_DISABLE)
17574 // If this config identifier is defined then all CATCH macros are prefixed with CATCH_
17575 #define CATCH_REQUIRE( ... ) INTERNAL_CATCH_TEST( "CATCH_REQUIRE", Catch::ResultDisposition::Normal, __VA_ARGS__ )
17576 #define CATCH_REQUIRE_FALSE( ... ) INTERNAL_CATCH_TEST( "CATCH_REQUIRE_FALSE", Catch::ResultDisposition::Normal | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
17577 #define CATCH_REQUIRE_THROWS( ... ) INTERNAL_CATCH_THROWS( "CATCH_REQUIRE_THROWS", Catch::ResultDisposition::Normal, __VA_ARGS__ )
17578 #define CATCH_REQUIRE_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS( "CATCH_REQUIRE_THROWS_AS", exceptionType, Catch::ResultDisposition::Normal, expr )
17579 #define CATCH_REQUIRE_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "CATCH_REQUIRE_THROWS_WITH", Catch::ResultDisposition::Normal, matcher, expr )
17580 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17581 #define CATCH_REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES( "CATCH_REQUIRE_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::Normal, matcher, expr )
17582 #endif
17583 #define CATCH_REQUIRE_NO_THROW( ... ) INTERNAL_CATCH_NO_THROW( "CATCH_REQUIRE_NO_THROW", Catch::ResultDisposition::Normal, __VA_ARGS__ )
17584 #define CATCH_CHECK( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK", Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17585 #define CATCH_CHECK_FALSE( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK_FALSE", Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
17586 #define CATCH_CHECKED_IF( ... ) INTERNAL_CATCH_IF( "CATCH_CHECKED_IF", Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17587 #define CATCH_CHECKED_ELSE( ... ) INTERNAL_CATCH_ELSE( "CATCH_CHECKED_ELSE", Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17588 #define CATCH_CHECK_NOFAIL( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK_NOFAIL", Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::SuppressFail, __VA_ARGS__ )
17589 #define CATCH_CHECK_THROWS( ... ) INTERNAL_CATCH_THROWS( "CATCH_CHECK_THROWS", Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17590 #define CATCH_CHECK_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS( "CATCH_CHECK_THROWS_AS", exceptionType, Catch::ResultDisposition::ContinueOnFailure, expr )
17591 #define CATCH_CHECK_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "CATCH_CHECK_THROWS_WITH", Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
17592 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17593 #define CATCH_CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES( "CATCH_CHECK_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
17594 #endif
17595 #define CATCH_CHECK_NO_THROW( ... ) INTERNAL_CATCH_NO_THROW( "CATCH_CHECK_NO_THROW", Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17596 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17597 #define CATCH_CHECK_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CATCH_CHECK_THAT", matcher, Catch::ResultDisposition::ContinueOnFailure, arg )
17598 #endif

```

```
17607
17608 #define CATCH_REQUIRE_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CATCH_REQUIRE_THAT", matcher,
Catch::ResultDisposition::Normal, arg )
17609 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17610
17611 #define CATCH_INFO( msg ) INTERNAL_CATCH_INFO( "CATCH_INFO", msg )
17612 #define CATCH_UNSCOPED_INFO( msg ) INTERNAL_CATCH_UNSCOPED_INFO( "CATCH_UNSCOPED_INFO", msg )
17613 #define CATCH_WARN( msg ) INTERNAL_CATCH_MSG( "CATCH_WARN", Catch::ResultWas::Warning,
Catch::ResultDisposition::ContinueOnFailure, msg )
17614 #define CATCH_CAPTURE( ... ) INTERNAL_CATCH_CAPTURE( INTERNAL_CATCH_UNIQUE_NAME(capturer),
"CATCH_CAPTURE",__VA_ARGS__ )
17615
17616 #define CATCH_TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE( __VA_ARGS__ )
17617 #define CATCH_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className,
__VA_ARGS__ )
17618 #define CATCH_METHOD_AS_TEST_CASE( method, ... ) INTERNAL_CATCH_METHOD_AS_TEST_CASE( method,
__VA_ARGS__ )
17619 #define CATCH_REGISTER_TEST_CASE( Function, ... ) INTERNAL_CATCH_REGISTER_TESTCASE( Function,
__VA_ARGS__ )
17620 #define CATCH_SECTION( ... ) INTERNAL_CATCH_SECTION( __VA_ARGS__ )
17621 #define CATCH_DYNAMIC_SECTION( ... ) INTERNAL_CATCH_DYNAMIC_SECTION( __VA_ARGS__ )
17622 #define CATCH_FAIL( ... ) INTERNAL_CATCH_MSG( "CATCH_FAIL", Catch::ResultWas::ExplicitFailure,
Catch::ResultDisposition::Normal, __VA_ARGS__ )
17623 #define CATCH_FAIL_CHECK( ... ) INTERNAL_CATCH_MSG( "CATCH_FAIL_CHECK",
Catch::ResultWas::ExplicitFailure, Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17624 #define CATCH_SUCCEED( ... ) INTERNAL_CATCH_MSG( "CATCH_SUCCEED", Catch::ResultWas::Ok,
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17625
17626 #define CATCH_ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE()
17627
17628 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
17629 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17630 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ )
17631 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD(
className, __VA_ARGS__ )
17632 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17633 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__
)
17634 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(
__VA_ARGS__ )
17635 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... )
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ )
17636 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... )
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17637 #else
17638 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ ) )
17639 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ ) )
17640 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
17641 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
17642 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ ) )
17643 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( __VA_ARGS__ ) )
17644 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
17645 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
17646 #endif
17647
17648 #if !defined(CATCH_CONFIG_RUNTIME_STATIC_REQUIRE)
17649 #define CATCH_STATIC_REQUIRE( ... ) static_assert( __VA_ARGS__ , #__VA_ARGS__ );
CATCH_SUCCEED( #__VA_ARGS__ )
17650 #define CATCH_STATIC_REQUIRE_FALSE( ... ) static_assert( !(__VA_ARGS__), "!(" #__VA_ARGS__ ")" );
CATCH_SUCCEED( #__VA_ARGS__ )
17651 #else
17652 #define CATCH_STATIC_REQUIRE( ... ) CATCH_REQUIRE( __VA_ARGS__ )
17653 #define CATCH_STATIC_REQUIRE_FALSE( ... ) CATCH_REQUIRE_FALSE( __VA_ARGS__ )
17654 #endif
17655
17656 // "BDD-style" convenience wrappers
17657 #define CATCH_SCENARIO( ... ) CATCH_TEST_CASE( "Scenario: " __VA_ARGS__ )
17658 #define CATCH_SCENARIO_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, "Scenario:
" __VA_ARGS__ )
17659 #define CATCH_GIVEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " Given: " « desc )
17660 #define CATCH_AND_GIVEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( "And given: " « desc )
17661 #define CATCH_WHEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " When: " « desc )
17662 #define CATCH_AND_WHEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " And when: " « desc )
17663 #define CATCH_THEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " Then: " « desc )
17664 #define CATCH_AND_THEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " And: " « desc )
17665
17666 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
17667 #define CATCH_BENCHMARK(...) \
```



```

17668     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__))
17669 #define CATCH_BENCHMARK_ADVANCED(name) \
17670     INTERNAL_CATCH_BENCHMARK_ADVANCED(INTERNAL_CATCH_UNIQUE_NAME(C_A_T_C_H_B_E_N_C_H_), name)
17671 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
17672
17673 // If CATCH_CONFIG_PREFIX_ALL is not defined then the CATCH_ prefix is not required
17674 #else
17675
17676 #define REQUIRE( ... ) INTERNAL_CATCH_TEST( "REQUIRE", Catch::ResultDisposition::Normal, __VA_ARGS__ )
17677 #define REQUIRE_FALSE( ... ) INTERNAL_CATCH_TEST( "REQUIRE_FALSE", Catch::ResultDisposition::Normal |
Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
17678
17679 #define REQUIRE_THROWS( ... ) INTERNAL_CATCH_THROWS( "REQUIRE_THROWS",
Catch::ResultDisposition::Normal, __VA_ARGS__ )
17680 #define REQUIRE_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS( "REQUIRE_THROWS_AS",
exceptionType, Catch::ResultDisposition::Normal, expr )
17681 #define REQUIRE_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "REQUIRE_THROWS_WITH",
Catch::ResultDisposition::Normal, matcher, expr )
17682 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17683 #define REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
"REQUIRE_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::Normal, matcher, expr )
17684 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17685 #define REQUIRE_NO_THROW( ... ) INTERNAL_CATCH_NO_THROW( "REQUIRE_NO_THROW",
Catch::ResultDisposition::Normal, __VA_ARGS__ )
17686
17687 #define CHECK( ... ) INTERNAL_CATCH_TEST( "CHECK", Catch::ResultDisposition::ContinueOnFailure,
__VA_ARGS__ )
17688 #define CHECK_FALSE( ... ) INTERNAL_CATCH_TEST( "CHECK_FALSE",
Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
17689 #define CHECKED_IF( ... ) INTERNAL_CATCH_IF( "CHECKED_IF",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17690 #define CHECKED_ELSE( ... ) INTERNAL_CATCH_ELSE( "CHECKED_ELSE",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17691 #define CHECK_NO_FAIL( ... ) INTERNAL_CATCH_TEST( "CHECK_NO_FAIL",
Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::SuppressFail, __VA_ARGS__ )
17692
17693 #define CHECK_THROWS( ... ) INTERNAL_CATCH_THROWS( "CHECK_THROWS",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17694 #define CHECK_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS( "CHECK_THROWS_AS",
exceptionType, Catch::ResultDisposition::ContinueOnFailure, expr )
17695 #define CHECK_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "CHECK_THROWS_WITH",
Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
17696 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17697 #define CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
"CHECK_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
17698 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17699 #define CHECK_NO_THROW( ... ) INTERNAL_CATCH_NO_THROW( "CHECK_NO_THROW",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17700
17701 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17702 #define CHECK_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CHECK_THAT", matcher,
Catch::ResultDisposition::ContinueOnFailure, arg )
17703
17704 #define REQUIRE_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "REQUIRE_THAT", matcher,
Catch::ResultDisposition::Normal, arg )
17705 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17706
17707 #define INFO( msg ) INTERNAL_CATCH_INFO( "INFO", msg )
17708 #define UNSCOPED_INFO( msg ) INTERNAL_CATCH_UNSCOPED_INFO( "UNSCOPED_INFO", msg )
17709 #define WARN( msg ) INTERNAL_CATCH_MSG( "WARN", Catch::ResultWas::Warning,
Catch::ResultDisposition::ContinueOnFailure, msg )
17710 #define CAPTURE( ... ) INTERNAL_CATCH_CAPTURE( INTERNAL_CATCH_UNIQUE_NAME(capturer),
"CAPTURE", __VA_ARGS__ )
17711
17712 #define TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE( __VA_ARGS__ )
17713 #define TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, __VA_ARGS__ )
17714 #define METHOD_AS_TEST_CASE( method, ... ) INTERNAL_CATCH_METHOD_AS_TEST_CASE( method, __VA_ARGS__ )
17715 #define REGISTER_TEST_CASE( Function, ... ) INTERNAL_CATCH_REGISTER_TESTCASE( Function, __VA_ARGS__ )
17716 #define SECTION( ... ) INTERNAL_CATCH_SECTION( __VA_ARGS__ )
17717 #define DYNAMIC_SECTION( ... ) INTERNAL_CATCH_DYNAMIC_SECTION( __VA_ARGS__ )
17718 #define FAIL( ... ) INTERNAL_CATCH_MSG( "FAIL", Catch::ResultWas::ExplicitFailure,
Catch::ResultDisposition::Normal, __VA_ARGS__ )
17719 #define FAIL_CHECK( ... ) INTERNAL_CATCH_MSG( "FAIL_CHECK", Catch::ResultWas::ExplicitFailure,
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17720 #define SUCCEED( ... ) INTERNAL_CATCH_MSG( "SUCCEED", Catch::ResultWas::Ok,
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
17721 #define ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE()
17722
17723 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
17724 #define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17725 #define TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ )
17726 #define TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD(
className, __VA_ARGS__ )
17727 #define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG(
className, __VA_ARGS__ )

```

```

17728 #define TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ )
17729 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(
__VA_ARGS__ )
17730 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... )
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ )
17731 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... )
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
17732 #define TEMPLATE_LIST_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE(__VA_ARGS__)
17733 #define TEMPLATE_LIST_TEST_CASE_METHOD( className, ... )
INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD( className, __VA_ARGS__ )
17734 #else
17735 #define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE(
__VA_ARGS__ ) )
17736 #define TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ ) )
17737 #define TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
17738 #define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
17739 #define TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ ) )
17740 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( __VA_ARGS__ ) )
17741 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
17742 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
17743 #define TEMPLATE_LIST_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE( __VA_ARGS__ ) )
17744 #define TEMPLATE_LIST_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD( 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(
#__VA_ARGS__ )
17749 #define STATIC_REQUIRE_FALSE( ... ) static_assert( !(__VA_ARGS__), "!(" #__VA_ARGS__ ")" ); SUCCEED(
"!(" #__VA_ARGS__ ")" )
17750 #else
17751 #define STATIC_REQUIRE( ... ) REQUIRE( __VA_ARGS__ )
17752 #define STATIC_REQUIRE_FALSE( ... ) REQUIRE_FALSE( __VA_ARGS__ )
17753 #endif
17754
17755 #endif
17756
17757 #define CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION( signature )
17758
17759 // "BDD-style" convenience wrappers
17760 #define SCENARIO( ... ) TEST_CASE( "Scenario: " __VA_ARGS__ )
17761 #define SCENARIO_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, "Scenario: "
__VA_ARGS__ )
17762
17763 #define GIVEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " Given: " « desc )
17764 #define AND_GIVEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( "And given: " « desc )
17765 #define WHEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " When: " « desc )
17766 #define AND_WHEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " And when: " « desc )
17767 #define THEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " Then: " « desc )
17768 #define AND_THEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( " And: " « desc )
17769
17770 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
17771 #define BENCHMARK(...) \
17772 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__,))
17773 #define BENCHMARK_ADVANCED(name) \
17774 INTERNAL_CATCH_BENCHMARK_ADVANCED(INTERNAL_CATCH_UNIQUE_NAME(C_A_T_C_H_B_E_N_C_H_), name)
17775 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
17776
17777 using Catch::Detail::Approx;
17778
17779 #else // CATCH_CONFIG_DISABLE
17780
17781 // If this config identifier is defined then all CATCH macros are prefixed with CATCH_
17782 #ifdef CATCH_CONFIG_PREFIX_ALL
17783 #define CATCH_REQUIRE( ... ) (void)(0)
17784 #define CATCH_REQUIRE_FALSE( ... ) (void)(0)
17785 #define CATCH_REQUIRE_THROWS( ... ) (void)(0)
17786 #define CATCH_REQUIRE_THROWS_AS( expr, exceptionType ) (void)(0)
17787 #define CATCH_REQUIRE_THROWS_WITH( expr, matcher ) (void)(0)
17788 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17789 #define CATCH_REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
17790 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17791 #define CATCH_REQUIRE_NO_THROW( ... ) (void)(0)
17792
17793 #define CATCH_CHECK( ... ) (void)(0)
17794 #define CATCH_CHECK_FALSE( ... ) (void)(0)

```

```

17798 #define CATCH_CHECKED_IF( ... )    if ( __VA_ARGS__ )
17799 #define CATCH_CHECKED_ELSE( ... )  if (!( __VA_ARGS__ ))
17800 #define CATCH_CHECK_NOFAIL( ... )  (void) (0)
17801
17802 #define CATCH_CHECK_THROWS( ... )  (void) (0)
17803 #define CATCH_CHECK_THROWS_AS( expr, exceptionType ) (void) (0)
17804 #define CATCH_CHECK_THROWS_WITH( expr, matcher )      (void) (0)
17805 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17806 #define CATCH_CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) (void) (0)
17807 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17808 #define CATCH_CHECK_NOTHROW( ... ) (void) (0)
17809
17810 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17811 #define CATCH_CHECK_THAT( arg, matcher )    (void) (0)
17812
17813 #define CATCH_REQUIRE_THAT( arg, matcher ) (void) (0)
17814 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17815
17816 #define CATCH_INFO( msg )          (void) (0)
17817 #define CATCH_UNSCOPED_INFO( msg ) (void) (0)
17818 #define CATCH_WARN( msg )          (void) (0)
17819 #define CATCH_CAPTURE( msg )       (void) (0)
17820
17821 #define CATCH_TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_S_T_ ))
17822 #define CATCH_TEST_CASE_METHOD( className, ... )
INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ))
17823 #define CATCH_METHOD_AS_TEST_CASE( method, ... )
17824 #define CATCH_REGISTER_TEST_CASE( Function, ... ) (void) (0)
17825 #define CATCH_SECTION( ... )
17826 #define CATCH_DYNAMIC_SECTION( ... )
17827 #define CATCH_FAIL( ... ) (void) (0)
17828 #define CATCH_FAIL_CHECK( ... ) (void) (0)
17829 #define CATCH_SUCCEED( ... ) (void) (0)
17830
17831 #define CATCH_ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_S_T_ ))
17832
17833 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
17834 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION( __VA_ARGS__ )
17835 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION( __VA_ARGS__ )
17836 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION( className, __VA_ARGS__ )
17837 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION( className, __VA_ARGS__ )
17838 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17839 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17840 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
className, __VA_ARGS__ )
17841 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
className, __VA_ARGS__ )
17842 #else
17843 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION( __VA_ARGS__ ) )
17844 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION( __VA_ARGS__ ) )
17845 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION( className, __VA_ARGS__ ) )
17846 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION( className, __VA_ARGS__ ) )
17847 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17848 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
17849 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
className, __VA_ARGS__ )
17850 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
className, __VA_ARGS__ )
17851 #endif
17852
17853 // "BDD-style" convenience wrappers
17854 #define CATCH_SCENARIO( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_S_T_ ))
17855 #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 )
17856 #define CATCH_GIVEN( desc )
17857 #define CATCH_AND_GIVEN( desc )
17858 #define CATCH_WHEN( desc )
17859 #define CATCH_AND_WHEN( desc )
17860 #define CATCH_THEN( desc )
17861 #define CATCH_AND_THEN( desc )
17862
17863 #define CATCH_STATIC_REQUIRE( ... )          (void) (0)
17864 #define CATCH_STATIC_REQUIRE_FALSE( ... ) (void) (0)
17865
17866 // If CATCH_CONFIG_PREFIX_ALL is not defined then the CATCH_ prefix is not required
17867 #else

```

```
17868
17869 #define REQUIRE( ... ) (void)(0)
17870 #define REQUIRE_FALSE( ... ) (void)(0)
17871
17872 #define REQUIRE_THROWS( ... ) (void)(0)
17873 #define REQUIRE_THROWS_AS( expr, exceptionType ) (void)(0)
17874 #define REQUIRE_THROWS_WITH( expr, matcher ) (void)(0)
17875 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17876 #define REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
17877 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17878 #define REQUIRE_NOTHROW( ... ) (void)(0)
17879
17880 #define CHECK( ... ) (void)(0)
17881 #define CHECK_FALSE( ... ) (void)(0)
17882 #define CHECKED_IF( ... ) if ( __VA_ARGS__ )
17883 #define CHECKED_ELSE( ... ) if (!( __VA_ARGS__ ))
17884 #define CHECK_NOFAIL( ... ) (void)(0)
17885
17886 #define CHECK_THROWS( ... ) (void)(0)
17887 #define CHECK_THROWS_AS( expr, exceptionType ) (void)(0)
17888 #define CHECK_THROWS_WITH( expr, matcher ) (void)(0)
17889 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17890 #define CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
17891 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17892 #define CHECK_NOTHROW( ... ) (void)(0)
17893
17894 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
17895 #define CHECK_THAT( arg, matcher ) (void)(0)
17896
17897 #define REQUIRE_THAT( arg, matcher ) (void)(0)
17898 #endif // CATCH_CONFIG_DISABLE_MATCHERS
17899
17900 #define INFO( msg ) (void)(0)
17901 #define UNSCOPED_INFO( msg ) (void)(0)
17902 #define WARN( msg ) (void)(0)
17903 #define CAPTURE( ... ) (void)(0)
17904
17905 #define TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_S_T_ ))
17906 #define TEST_CASE_METHOD( className, ... )
INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ))
17907 #define METHOD_AS_TEST_CASE( method, ... )
17908 #define REGISTER_TEST_CASE( Function, ... ) (void)(0)
17909 #define SECTION( ... )
17910 #define DYNAMIC_SECTION( ... )
17911 #define FAIL( ... ) (void)(0)
17912 #define FAIL_CHECK( ... ) (void)(0)
17913 #define SUCCEED( ... ) (void)(0)
17914 #define ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_S_T_ ))
17915
17916 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
17917 #define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__)
17918 #define TEMPLATE_TEST_CASE_SIG( ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__)
17919 #define TEMPLATE_TEST_CASE_METHOD( className, ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__)
17920 #define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ )
17921 #define TEMPLATE_PRODUCT_TEST_CASE( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
17922 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
17923 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
__VA_ARGS__ )
17924 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
__VA_ARGS__ )
17925 #else
17926 #define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__) )
17927 #define TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__) )
17928 #define TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__ ) )
17929 #define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ ) )
17930 #define TEMPLATE_PRODUCT_TEST_CASE( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
17931 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
17932 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
__VA_ARGS__ )
17933 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
__VA_ARGS__ )
17934 #endif
17935
17936 #define STATIC_REQUIRE( ... ) (void)(0)
17937 #define STATIC_REQUIRE_FALSE( ... ) (void)(0)
17938
17939 #endif
17940
```

```

17941 #define CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION_NO_REG(
INTERNAL_CATCH_UNIQUE_NAME( catch_internal_ExceptionTranslator ), signature )
17942
17943 // "BDD-style" convenience wrappers
17944 #define SCENARIO( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_S_T_ ) )
17945 #define SCENARIO_METHOD( className, ... )
INTERNAL_CATCH_TESTCASE_METHOD_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ),
className )
17946
17947 #define GIVEN( desc )
17948 #define AND_GIVEN( desc )
17949 #define WHEN( desc )
17950 #define AND_WHEN( desc )
17951 #define THEN( desc )
17952 #define AND_THEN( desc )
17953
17954 using Catch::Detail::Approx;
17955
17956 #endif
17957
17958 #endif // ! CATCH_CONFIG_IMPL_ONLY
17959
17960 // start catch_reenable_warnings.h
17961
17962
17963 #ifdef __clang__
17964 #   ifdef __ICC // icpc defines the __clang__ macro
17965 #       pragma warning(pop)
17966 #   else
17967 #       pragma clang diagnostic pop
17968 #   endif
17969 #elif defined __GNUC__
17970 #   pragma GCC diagnostic pop
17971 #endif
17972
17973 // end catch_reenable_warnings.h
17974 // end catch.hpp
17975 #endif // TWOBLUECUBES_SINGLE_INCLUDE_CATCH_HPP_INCLUDED
17976 /*
17977  * Catch v2.13.10
17978  * Generated: 2022-10-16 11:01:23.452308
17979  * -----
17980  * This file has been merged from multiple headers. Please don't edit it directly
17981  * Copyright (c) 2022 Two Blue Cubes Ltd. All rights reserved.
17982  *
17983  * Distributed under the Boost Software License, Version 1.0. (See accompanying
17984  * file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
17985  */
17986 #ifndef TWOBLUECUBES_SINGLE_INCLUDE_CATCH_HPP_INCLUDED
17987 #define TWOBLUECUBES_SINGLE_INCLUDE_CATCH_HPP_INCLUDED
17988 // start catch.hpp
17989
17990
17991 #define CATCH_VERSION_MAJOR 2
17992 #define CATCH_VERSION_MINOR 13
17993 #define CATCH_VERSION_PATCH 10
17994
17995 #ifdef __clang__
17996 #   pragma clang system_header
17997 #elif defined __GNUC__
17998 #   pragma GCC system_header
17999 #endif
18000
18001 // start catch_suppress_warnings.h
18002
18003 #ifdef __clang__
18004 #   ifdef __ICC // icpc defines the __clang__ macro
18005 #       pragma warning(push)
18006 #       pragma warning(disable: 161 1682)
18007 #   else // __ICC
18008 #       pragma clang diagnostic push
18009 #       pragma clang diagnostic ignored "-Wpadded"
18010 #       pragma clang diagnostic ignored "-Wswitch-enum"
18011 #       pragma clang diagnostic ignored "-Wcovered-switch-default"
18012 #   endif
18013 #elif defined __GNUC__
18014 // Because REQUIREs trigger GCC's -Wparentheses, and because still
18015 // supported version of g++ have only buggy support for _Pragmas,
18016 // Wparentheses have to be suppressed globally.
18017 #   pragma GCC diagnostic ignored "-Wparentheses" // See #674 for details
18018
18019 #   pragma GCC diagnostic push
18020 #   pragma GCC diagnostic ignored "-Wunused-variable"
18021 #   pragma GCC diagnostic ignored "-Wpadded"
18022 #endif
18023 // end catch_suppress_warnings.h

```

```

18024 #if defined(CATCH_CONFIG_MAIN) || defined(CATCH_CONFIG_RUNNER)
18025 #   define CATCH_IMPL
18026 #   define CATCH_CONFIG_ALL_PARTS
18027 #endif
18028
18029 // In the impl file, we want to have access to all parts of the headers
18030 // Can also be used to sanely support PCHs
18031 #if defined(CATCH_CONFIG_ALL_PARTS)
18032 #   define CATCH_CONFIG_EXTERNAL_INTERFACES
18033 #   if defined(CATCH_CONFIG_DISABLE_MATCHERS)
18034 #       undef CATCH_CONFIG_DISABLE_MATCHERS
18035 #   endif
18036 #   if !defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
18037 #       define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
18038 #   endif
18039 #endif
18040
18041 #if !defined(CATCH_CONFIG_IMPL_ONLY)
18042 // start catch_platform.h
18043
18044 // See e.g.:
18045 // https://opensource.apple.com/source/CarbonHeaders/CarbonHeaders-18.1/TargetConditionals.h.auto.html
18046 #ifdef __APPLE__
18047 #   include <TargetConditionals.h>
18048 #   if (defined(TARGET_OS_OSX) && TARGET_OS_OSX == 1) || \
18049       (defined(TARGET_OS_MAC) && TARGET_OS_MAC == 1)
18050 #       define CATCH_PLATFORM_MAC
18051 #   elif (defined(TARGET_OS_IPHONE) && TARGET_OS_IPHONE == 1)
18052 #       define CATCH_PLATFORM_IPHONE
18053 #   endif
18054
18055 #elif defined(linux) || defined(__linux) || defined(__linux__)
18056 #   define CATCH_PLATFORM_LINUX
18057
18058 #elif defined(WIN32) || defined(__WIN32__) || defined(_WIN32) || defined(_MSC_VER) ||
18059       defined(__MINGW32__)
18060 #   define CATCH_PLATFORM_WINDOWS
18061 #endif
18062 // end catch_platform.h
18063
18064 #ifdef CATCH_IMPL
18065 #   ifndef CLARA_CONFIG_MAIN
18066 #       define CLARA_CONFIG_MAIN_NOT_DEFINED
18067 #       define CLARA_CONFIG_MAIN
18068 #   endif
18069 #endif
18070
18071 // start catch_user_interfaces.h
18072
18073 namespace Catch {
18074     unsigned int rngSeed();
18075 }
18076
18077 // end catch_user_interfaces.h
18078 // start catch_tag_alias_autoregistrar.h
18079
18080 // start catch_common.h
18081
18082 // start catch_compiler_capabilities.h
18083
18084 // Detect a number of compiler features - by compiler
18085 // The following features are defined:
18086 //
18087 // CATCH_CONFIG_COUNTER : is the __COUNTER__ macro supported?
18088 // CATCH_CONFIG_WINDOWS_SEH : is Windows SEH supported?
18089 // CATCH_CONFIG_POSIX_SIGNALS : are POSIX signals supported?
18090 // CATCH_CONFIG_DISABLE_EXCEPTIONS : Are exceptions enabled?
18091 // *****
18092 // Note to maintainers: if new toggles are added please document them
18093 // in configuration.md, too
18094 // *****
18095
18096 // In general each macro has a _NO_<feature name> form
18097 // (e.g. CATCH_CONFIG_NO_POSIX_SIGNALS) which disables the feature.
18098 // Many features, at point of detection, define an _INTERNAL_ macro, so they
18099 // can be combined, en-mass, with the _NO_ forms later.
18100
18101 #ifdef __cplusplus
18102
18103 #   if (__cplusplus >= 201402L) || (defined(_MSVC_LANG) && _MSVC_LANG >= 201402L)
18104 #       define CATCH_CPP14_OR_GREATER
18105 #   endif
18106
18107 #   if (__cplusplus >= 201703L) || (defined(_MSVC_LANG) && _MSVC_LANG >= 201703L)
18108 #       define CATCH_CPP17_OR_GREATER
18109 #   endif
18110

```

```

18110
18111 #endif
18112
18113 // Only GCC compiler should be used in this block, so other compilers trying to
18114 // mask themselves as GCC should be ignored.
18115 #if defined(__GNUC__) && !defined(__clang__) && !defined(__ICC) && !defined(__CUDACC__) &&
!defined(__LCC__)
18116 #   define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION _Pragma( "GCC diagnostic push" )
18117 #   define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION _Pragma( "GCC diagnostic pop" )
18118
18119 #   define CATCH_INTERNAL_IGNORE_BUT_WARN(...) (void)__builtin_constant_p(__VA_ARGS__)
18120
18121 #endif
18122
18123 #if defined(__clang__)
18124
18125 #   define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION _Pragma( "clang diagnostic push" )
18126 #   define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION _Pragma( "clang diagnostic pop" )
18127
18128 // As of this writing, IBM XL's implementation of __builtin_constant_p has a bug
18129 // which results in calls to destructors being emitted for each temporary,
18130 // without a matching initialization. In practice, this can result in something
18131 // like `std::string::~string` being called on an uninitialized value.
18132 //
18133 // For example, this code will likely segfault under IBM XL:
18134 // ```
18135 // REQUIRE(std::string("12") + "34" == "1234")
18136 // ```
18137 //
18138 // Therefore, `CATCH_INTERNAL_IGNORE_BUT_WARN` is not implemented.
18139 #   if !defined(__ibmxl__) && !defined(__CUDACC__)
18140 #       define CATCH_INTERNAL_IGNORE_BUT_WARN(...) (void)__builtin_constant_p(__VA_ARGS__) /*
NOLINT(cppcoreguidelines-pro-type-vararg, hicpp-vararg) */
18141 #   endif
18142
18143 #   define CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
18144     _Pragma( "clang diagnostic ignored \\"-Wexit-time-destructors\\" ) \
18145     _Pragma( "clang diagnostic ignored \\"-Wglobal-constructors\\" )
18146
18147 #   define CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS \
18148     _Pragma( "clang diagnostic ignored \\"-Wparentheses\\" )
18149
18150 #   define CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
18151     _Pragma( "clang diagnostic ignored \\"-Wunused-variable\\" )
18152
18153 #   define CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
18154     _Pragma( "clang diagnostic ignored \\"-Wgnu-zero-variadic-macro-arguments\\" )
18155
18156 #   define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
18157     _Pragma( "clang diagnostic ignored \\"-Wunused-template\\" )
18158
18159 #endif // __clang__
18160
18161 // Assume that non-Windows platforms support posix signals by default
18162 #if !defined(CATCH_PLATFORM_WINDOWS)
18163 #define CATCH_INTERNAL_CONFIG_POSIX_SIGNALS
18164 #endif
18165 #endif
18166
18167 // We know some environments not to support full POSIX signals
18168 #if defined(__CYGWIN__) || defined(__QNX__) || defined(__EMSCRIPTEN__) || defined(__DJGPP__)
18169 #define CATCH_INTERNAL_CONFIG_NO_POSIX_SIGNALS
18170 #endif
18171 #endif
18172
18173 #ifdef __OS400__
18174 #define CATCH_INTERNAL_CONFIG_NO_POSIX_SIGNALS
18175 #define CATCH_CONFIG_COLOUR_NONE
18176 #endif
18177
18178 // Android somehow still does not support std::to_string
18179 #if defined(__ANDROID__)
18180 #define CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING
18181 #define CATCH_INTERNAL_CONFIG_ANDROID_LOGWRITE
18182 #endif
18183 #endif
18184
18185 // Not all Windows environments support SEH properly
18186 #if defined(__MINGW32__)
18187 #define CATCH_INTERNAL_CONFIG_NO_WINDOWS_SEH
18188 #endif
18189 #endif
18190
18191 // PS4
18192 #if defined(__ORBIS__)
18193 #define CATCH_INTERNAL_CONFIG_NO_NEW_CAPTURE
18194 #endif
18195 #endif
18196
18197 // Cygwin
18198 #ifdef __CYGWIN__
18199 #endif
18200

```



```

18201 // Required for some versions of Cygwin to declare gettimeofday
18202 // see: http://stackoverflow.com/questions/36901803/gettimeofday-not-declared-in-this-scope-cygwin
18203 #   define _BSD_SOURCE
18204 // some versions of cygwin (most) do not support std::to_string. Use the libstd check.
18205 // https://gcc.gnu.org/onlinedocs/gcc-4.8.2/libstdc++/api/a01053_source.html line 2812-2813
18206 # if !((__cplusplus >= 201103L) && defined(_GLIBCXX_USE_C99) \
18207        && !defined(_GLIBCXX_HAVE_BROKEN_VSWPRINTF))
18208
18209 #     define CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING
18210
18211 # endif
18212 #endif // __CYGWIN__
18213
18215 // Visual C++
18216 #if defined(_MSC_VER)
18217
18218 // Universal Windows platform does not support SEH
18219 // Or console colours (or console at all...)
18220 #   if defined(WINAPI_FAMILY) && (WINAPI_FAMILY == WINAPI_FAMILY_APP)
18221 #       define CATCH_CONFIG_COLOUR_NONE
18222 #   else
18223 #       define CATCH_INTERNAL_CONFIG_WINDOWS_SEH
18224 #   endif
18225
18226 #   if !defined(__clang__) // Handle Clang masquerading for msvc
18227
18228 // MSVC traditional preprocessor needs some workaround for __VA_ARGS__
18229 // _MSVC_TRADITIONAL == 0 means new conformant preprocessor
18230 // _MSVC_TRADITIONAL == 1 means old traditional non-conformant preprocessor
18231 #       if !defined(_MSVC_TRADITIONAL) || (defined(_MSVC_TRADITIONAL) && _MSVC_TRADITIONAL)
18232 #           define CATCH_INTERNAL_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
18233 #       endif // _MSVC_TRADITIONAL
18234
18235 // Only do this if we're not using clang on Windows, which uses `diagnostic push` & `diagnostic pop`
18236 #       define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION __pragma( warning(push) )
18237 #       define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION __pragma( warning(pop) )
18238 #   endif // __clang__
18239
18240 #endif // _MSC_VER
18241
18242 #if defined(_REENTRANT) || defined(_MSC_VER)
18243 // Enable async processing, as -pthread is specified or no additional linking is required
18244 #   define CATCH_INTERNAL_CONFIG_USE_ASYNC
18245 #endif // _MSC_VER
18246
18248 // Check if we are compiled with -fno-exceptions or equivalent
18249 #if defined(__EXCEPTIONS) || defined(__cpp_exceptions) || defined(_CPPUNWIND)
18250 #   define CATCH_INTERNAL_CONFIG_EXCEPTIONS_ENABLED
18251 #endif
18252
18254 // DJGPP
18255 #ifdef __DJGPP__
18256 #   define CATCH_INTERNAL_CONFIG_NO_WCHAR
18257 #endif // __DJGPP__
18258
18260 // Embarcadero C++Build
18261 #if defined(_BORLANDC_)
18262     #define CATCH_INTERNAL_CONFIG_POLYFILL_ISNAN
18263 #endif
18264
18266
18267 // Use of __COUNTER__ is suppressed during code analysis in
18268 // CLion/AppCode 2017.2.x and former, because __COUNTER__ is not properly
18269 // handled by it.
18270 // Otherwise all supported compilers support COUNTER macro,
18271 // but user still might want to turn it off
18272 #if ( !defined(__JETBRAINS_IDE__) || __JETBRAINS_IDE__ >= 20170300L )
18273     #define CATCH_INTERNAL_CONFIG_COUNTER
18274 #endif
18275
18277
18278 // RTX is a special version of Windows that is real time.
18279 // This means that it is detected as Windows, but does not provide
18280 // the same set of capabilities as real Windows does.
18281 #if defined(UNDER_RTSS) || defined(RTX64_BUILD)
18282     #define CATCH_INTERNAL_CONFIG_NO_WINDOWS_SEH
18283     #define CATCH_INTERNAL_CONFIG_NO_ASYNC
18284     #define CATCH_CONFIG_COLOUR_NONE
18285 #endif
18286
18287 #if !defined(_GLIBCXX_USE_C99_MATH_TR1)
18288 #define CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER
18289 #endif
18290
18291 // Various stdlib support checks that require __has_include
18292 #if defined(__has_include)
18293     // Check if string_view is available and usable

```



```

18294 #if __has_include(<string_view>) && defined(CATCH_CPP17_OR_GREATER)
18295 #   define CATCH_INTERNAL_CONFIG_CPP17_STRING_VIEW
18296 #endif
18297
18298 // Check if optional is available and usable
18299 # if __has_include(<optional>) && defined(CATCH_CPP17_OR_GREATER)
18300 #   define CATCH_INTERNAL_CONFIG_CPP17_OPTIONAL
18301 # endif // __has_include(<optional>) && defined(CATCH_CPP17_OR_GREATER)
18302
18303 // Check if byte is available and usable
18304 # if __has_include(<cstdint>) && defined(CATCH_CPP17_OR_GREATER)
18305 #   include <cstdint>
18306 #   if defined(__cpp_lib_byte) && (__cpp_lib_byte > 0)
18307 #       define CATCH_INTERNAL_CONFIG_CPP17_BYTE
18308 #   endif
18309 # endif // __has_include(<cstdint>) && defined(CATCH_CPP17_OR_GREATER)
18310
18311 // Check if variant is available and usable
18312 # if __has_include(<variant>) && defined(CATCH_CPP17_OR_GREATER)
18313 #   if defined(__clang__) && (__clang_major__ < 8)
18314 #       // work around clang bug with libstdc++ https://bugs.llvm.org/show\_bug.cgi?id=31852
18315 #       // fix should be in clang 8, workaround in libstdc++ 8.2
18316 #       include <ciso646>
18317 #       if defined(__GLIBCXX__) && defined(_GLIBCXX_RELEASE) && (_GLIBCXX_RELEASE < 9)
18318 #           define CATCH_CONFIG_NO_CPP17_VARIANT
18319 #       else
18320 #           define CATCH_INTERNAL_CONFIG_CPP17_VARIANT
18321 #       endif // defined(__GLIBCXX__) && defined(_GLIBCXX_RELEASE) && (_GLIBCXX_RELEASE < 9)
18322 #   else
18323 #       define CATCH_INTERNAL_CONFIG_CPP17_VARIANT
18324 #   endif // defined(__clang__) && (__clang_major__ < 8)
18325 # endif // __has_include(<variant>) && defined(CATCH_CPP17_OR_GREATER)
18326 #endif // defined(__has_include)
18327
18328 #if defined(CATCH_INTERNAL_CONFIG_COUNTER) && !defined(CATCH_CONFIG_NO_COUNTER) &&
!defined(CATCH_CONFIG_COUNTER)
18329 #   define CATCH_CONFIG_COUNTER
18330 #endif
18331 #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)
18332 #   define CATCH_CONFIG_WINDOWS_SEH
18333 #endif
18334 // This is set by default, because we assume that unix compilers are posix-signal-compatible by
default.
18335 #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)
18336 #   define CATCH_CONFIG_POSIX_SIGNALS
18337 #endif
18338 // This is set by default, because we assume that compilers with no wchar_t support are just rare
exceptions.
18339 #if !defined(CATCH_INTERNAL_CONFIG_NO_WCHAR) && !defined(CATCH_CONFIG_NO_WCHAR) &&
!defined(CATCH_CONFIG_WCHAR)
18340 #   define CATCH_CONFIG_WCHAR
18341 #endif
18342
18343 #if !defined(CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING) && !defined(CATCH_CONFIG_NO_CPP11_TO_STRING) &&
!defined(CATCH_CONFIG_CPP11_TO_STRING)
18344 #   define CATCH_CONFIG_CPP11_TO_STRING
18345 #endif
18346
18347 #if defined(CATCH_INTERNAL_CONFIG_CPP17_OPTIONAL) && !defined(CATCH_CONFIG_NO_CPP17_OPTIONAL) &&
!defined(CATCH_CONFIG_CPP17_OPTIONAL)
18348 #   define CATCH_CONFIG_CPP17_OPTIONAL
18349 #endif
18350
18351 #if defined(CATCH_INTERNAL_CONFIG_CPP17_STRING_VIEW) && !defined(CATCH_CONFIG_NO_CPP17_STRING_VIEW) &&
!defined(CATCH_CONFIG_CPP17_STRING_VIEW)
18352 #   define CATCH_CONFIG_CPP17_STRING_VIEW
18353 #endif
18354
18355 #if defined(CATCH_INTERNAL_CONFIG_CPP17_VARIANT) && !defined(CATCH_CONFIG_NO_CPP17_VARIANT) &&
!defined(CATCH_CONFIG_CPP17_VARIANT)
18356 #   define CATCH_CONFIG_CPP17_VARIANT
18357 #endif
18358
18359 #if defined(CATCH_INTERNAL_CONFIG_CPP17_BYTE) && !defined(CATCH_CONFIG_NO_CPP17_BYTE) &&
!defined(CATCH_CONFIG_CPP17_BYTE)
18360 #   define CATCH_CONFIG_CPP17_BYTE
18361 #endif
18362
18363 #if defined(CATCH_CONFIG_EXPERIMENTAL_REDIRECT)
18364 #   define CATCH_INTERNAL_CONFIG_NEW_CAPTURE
18365 #endif
18366
18367 #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)
18368 #   define CATCH_CONFIG_NEW_CAPTURE

```

```

18369 #endif
18370
18371 #if !defined(CATCH_INTERNAL_CONFIG_EXCEPTIONS_ENABLED) && !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
18372 # define CATCH_CONFIG_DISABLE_EXCEPTIONS
18373 #endif
18374
18375 #if defined(CATCH_INTERNAL_CONFIG_POLYFILL_ISNAN) && !defined(CATCH_CONFIG_NO_POLYFILL_ISNAN) &&
!defined(CATCH_CONFIG_POLYFILL_ISNAN)
18376 # define CATCH_CONFIG_POLYFILL_ISNAN
18377 #endif
18378
18379 #if defined(CATCH_INTERNAL_CONFIG_USE_ASYNC) && !defined(CATCH_INTERNAL_CONFIG_NO_ASYNC) &&
!defined(CATCH_CONFIG_NO_USE_ASYNC) && !defined(CATCH_CONFIG_USE_ASYNC)
18380 # define CATCH_CONFIG_USE_ASYNC
18381 #endif
18382
18383 #if defined(CATCH_INTERNAL_CONFIG_ANDROID_LOGWRITE) && !defined(CATCH_CONFIG_NO_ANDROID_LOGWRITE) &&
!defined(CATCH_CONFIG_ANDROID_LOGWRITE)
18384 # define CATCH_CONFIG_ANDROID_LOGWRITE
18385 #endif
18386
18387 #if defined(CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER) && !defined(CATCH_CONFIG_NO_GLOBAL_NEXTAFTER) &&
!defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
18388 # define CATCH_CONFIG_GLOBAL_NEXTAFTER
18389 #endif
18390
18391 // Even if we do not think the compiler has that warning, we still have
18392 // to provide a macro that can be used by the code.
18393 #if !defined(CATCH_INTERNAL_START_WARNINGS_SUPPRESSION)
18394 # define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION
18395 #endif
18396 #if !defined(CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION)
18397 # define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
18398 #endif
18399 #if !defined(CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS)
18400 # define CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS
18401 #endif
18402 #if !defined(CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS)
18403 # define CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS
18404 #endif
18405 #if !defined(CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS)
18406 # define CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS
18407 #endif
18408 #if !defined(CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS)
18409 # define CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS
18410 #endif
18411
18412 // The goal of this macro is to avoid evaluation of the arguments, but
18413 // still have the compiler warn on problems inside...
18414 #if !defined(CATCH_INTERNAL_IGNORE_BUT_WARN)
18415 # define CATCH_INTERNAL_IGNORE_BUT_WARN(...)
18416 #endif
18417
18418 #if defined(__APPLE__) && defined(__apple_build_version__) && (__clang_major__ < 10)
18419 # undef CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
18420 #elif defined(__clang__) && (__clang_major__ < 5)
18421 # undef CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
18422 #endif
18423
18424 #if !defined(CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS)
18425 # define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
18426 #endif
18427
18428 #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
18429 #define CATCH_TRY if ((true))
18430 #define CATCH_CATCH_ALL if ((false))
18431 #define CATCH_CATCH_ANON(type) if ((false))
18432 #else
18433 #define CATCH_TRY try
18434 #define CATCH_CATCH_ALL catch (...)
18435 #define CATCH_CATCH_ANON(type) catch (type)
18436 #endif
18437
18438 #if defined(CATCH_INTERNAL_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR) &&
!defined(CATCH_CONFIG_NO_TRADITIONAL_MSVC_PREPROCESSOR) &&
!defined(CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR)
18439 #define CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
18440 #endif
18441
18442 // end catch_compiler_capabilities.h
18443 #define INTERNAL_CATCH_UNIQUE_NAME_LINE2( name, line ) name##line
18444 #define INTERNAL_CATCH_UNIQUE_NAME_LINE( name, line ) INTERNAL_CATCH_UNIQUE_NAME_LINE2( name, line )
18445 #ifdef CATCH_CONFIG_COUNTER
18446 # define INTERNAL_CATCH_UNIQUE_NAME( name ) INTERNAL_CATCH_UNIQUE_NAME_LINE( name, __COUNTER__ )
18447 #else
18448 # define INTERNAL_CATCH_UNIQUE_NAME( name ) INTERNAL_CATCH_UNIQUE_NAME_LINE( name, __LINE__ )
18449 #endif

```

```

18450
18451 #include <iosfwd>
18452 #include <string>
18453 #include <cstdint>
18454
18455 // We need a dummy global operator« so we can bring it into Catch namespace later
18456 struct Catch_global_namespace_dummy {};
18457 std::ostream& operator«(std::ostream&, Catch_global_namespace_dummy);
18458
18459 namespace Catch {
18460
18461     struct CaseSensitive { enum Choice {
18462         Yes,
18463         No
18464     }; };
18465
18466     class NonCopyable {
18467     public:
18468         NonCopyable( NonCopyable const& )           = delete;
18469         NonCopyable( NonCopyable && )               = delete;
18470         NonCopyable& operator = ( NonCopyable const& ) = delete;
18471         NonCopyable& operator = ( NonCopyable && )     = delete;
18472
18473     protected:
18474         NonCopyable();
18475         virtual ~NonCopyable();
18476     };
18477
18478     struct SourceLineInfo {
18479     public:
18480         SourceLineInfo() = delete;
18481         SourceLineInfo( char const* _file, std::size_t _line ) noexcept
18482             :   file( _file ),
18483                 line( _line )
18484         {}
18485
18486         SourceLineInfo( SourceLineInfo const& other )           = default;
18487         SourceLineInfo& operator = ( SourceLineInfo const& )   = default;
18488         SourceLineInfo( SourceLineInfo&& )                     noexcept = default;
18489         SourceLineInfo& operator = ( SourceLineInfo&& )         noexcept = default;
18490
18491         bool empty() const noexcept { return file[0] == '\0'; }
18492         bool operator == ( SourceLineInfo const& other ) const noexcept;
18493         bool operator < ( SourceLineInfo const& other ) const noexcept;
18494
18495         char const* file;
18496         std::size_t line;
18497     };
18498
18499     std::ostream& operator « ( std::ostream& os, SourceLineInfo const& info );
18500
18501     // Bring in operator« from global namespace into Catch namespace
18502     // This is necessary because the overload of operator« above makes
18503     // lookup stop at namespace Catch
18504     using ::operator«;
18505
18506     // Use this in variadic streaming macros to allow
18507     // » +StreamEndStop
18508     // as well as
18509     // » stuff +StreamEndStop
18510     struct StreamEndStop {
18511     public:
18512         std::string operator+() const;
18513     };
18514
18515     template<typename T>
18516     T const& operator + ( T const& value, StreamEndStop ) {
18517         return value;
18518     }
18519
18520 #define CATCH_INTERNAL_LINEINFO \
18521     ::Catch::SourceLineInfo( __FILE__, static_cast<std::size_t>( __LINE__ ) )
18522
18523 // end catch_common.h
18524 namespace Catch {
18525
18526     struct RegistrarForTagAliases {
18527     public:
18528         RegistrarForTagAliases( char const* alias, char const* tag, SourceLineInfo const& lineInfo );
18529     };
18530
18531 #define CATCH_REGISTER_TAG_ALIAS( alias, spec ) \
18532     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
18533     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
18534     namespace{ Catch::RegistrarForTagAliases INTERNAL_CATCH_UNIQUE_NAME( AutoRegisterTagAlias )( \
18535         alias, spec, CATCH_INTERNAL_LINEINFO ); } \
18536     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
18537

```

```

18536 // end catch_tag_alias_autoregistrar.h
18537 // start catch_test_registry.h
18538
18539 // start catch_interfaces_testcase.h
18540
18541 #include <vector>
18542
18543 namespace Catch {
18544
18545     class TestSpec;
18546
18547     struct ITestInvoker {
18548         virtual void invoke () const = 0;
18549         virtual ~ITestInvoker();
18550     };
18551
18552     class TestCase;
18553     struct IConfig;
18554
18555     struct ITestCaseRegistry {
18556         virtual ~ITestCaseRegistry();
18557         virtual std::vector<TestCase> const& getAllTests() const = 0;
18558         virtual std::vector<TestCase> const& getAllTestsSorted( IConfig const& config ) const = 0;
18559     };
18560
18561     bool isThrowSafe( TestCase const& testCase, IConfig const& config );
18562     bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config );
18563     std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
testSpec, IConfig const& config );
18564     std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config );
18565 }
18566
18567 // end catch_interfaces_testcase.h
18568 // start catch_stringref.h
18569
18570 #include <cstddef>
18571 #include <string>
18572 #include <iosfwd>
18573 #include <cassert>
18574
18575 namespace Catch {
18576
18577     class StringRef {
18578     public:
18579         using size_type = std::size_t;
18580         using const_iterator = const char*;
18581
18582     private:
18583         static constexpr char const* s_empty = "";
18584
18585         char const* m_start = s_empty;
18586         size_type m_size = 0;
18587
18588     public: // construction
18589         constexpr StringRef() noexcept = default;
18590
18591         StringRef( char const* rawChars ) noexcept;
18592
18593         constexpr StringRef( char const* rawChars, size_type size ) noexcept
18594         :   m_start( rawChars ),
18595             m_size( size )
18596         {}
18597
18598         StringRef( std::string const& stdString ) noexcept
18599         :   m_start( stdString.c_str() ),
18600             m_size( stdString.size() )
18601         {}
18602
18603         explicit operator std::string() const {
18604             return std::string(m_start, m_size);
18605         }
18606
18607     public: // operators
18608         auto operator == ( StringRef const& other ) const noexcept -> bool;
18609         auto operator != ( StringRef const& other ) const noexcept -> bool {
18610             return !(*this == other);
18611         }
18612
18613         auto operator[] ( size_type index ) const noexcept -> char {
18614             assert(index < m_size);
18615             return m_start[index];
18616         }
18617
18618     public: // named queries
18619         constexpr auto empty() const noexcept -> bool {
18620             return m_size == 0;
18621         }

```

```

18625     }
18626     constexpr auto size() const noexcept -> size_type {
18627         return m_size;
18628     }
18629
18630     // Returns the current start pointer. If the StringRef is not
18631     // null-terminated, throws std::domain_exception
18632     auto c_str() const -> char const*;
18633
18634     public: // substrings and searches
18635         // Returns a substring of [start, start + length).
18636         // If start + length > size(), then the substring is [start, size()).
18637         // If start > size(), then the substring is empty.
18638         auto substr( size_type start, size_type length ) const noexcept -> StringRef;
18639
18640         // Returns the current start pointer. May not be null-terminated.
18641         auto data() const noexcept -> char const*;
18642
18643         constexpr auto isNullTerminated() const noexcept -> bool {
18644             return m_start[m_size] == '\0';
18645         }
18646
18647     public: // iterators
18648         constexpr const_iterator begin() const { return m_start; }
18649         constexpr const_iterator end() const { return m_start + m_size; }
18650     };
18651
18652     auto operator += ( std::string& lhs, StringRef const& sr ) -> std::string&;
18653     auto operator << ( std::ostream& os, StringRef const& sr ) -> std::ostream&;
18654
18655     constexpr auto operator "" _sr( char const* rawChars, std::size_t size ) noexcept -> StringRef {
18656         return StringRef( rawChars, size );
18657     }
18658 } // namespace Catch
18659
18660 constexpr auto operator "" _catch_sr( char const* rawChars, std::size_t size ) noexcept ->
    Catch::StringRef {
18661     return Catch::StringRef( rawChars, size );
18662 }
18663
18664 // end catch_stringref.h
18665 // start catch_preprocessor.hpp
18666
18667 #define CATCH_RECURSION_LEVEL0(...) __VA_ARGS__
18668 #define CATCH_RECURSION_LEVEL1(...)
18669     CATCH_RECURSION_LEVEL0(CATCH_RECURSION_LEVEL0(CATCH_RECURSION_LEVEL0(__VA_ARGS__)))
18670 #define CATCH_RECURSION_LEVEL2(...)
18671     CATCH_RECURSION_LEVEL1(CATCH_RECURSION_LEVEL1(CATCH_RECURSION_LEVEL1(__VA_ARGS__)))
18672 #define CATCH_RECURSION_LEVEL3(...)
18673     CATCH_RECURSION_LEVEL2(CATCH_RECURSION_LEVEL2(CATCH_RECURSION_LEVEL2(__VA_ARGS__)))
18674 #define CATCH_RECURSION_LEVEL4(...)
18675     CATCH_RECURSION_LEVEL3(CATCH_RECURSION_LEVEL3(CATCH_RECURSION_LEVEL3(__VA_ARGS__)))
18676 #define CATCH_RECURSION_LEVEL5(...)
18677     CATCH_RECURSION_LEVEL4(CATCH_RECURSION_LEVEL4(CATCH_RECURSION_LEVEL4(__VA_ARGS__)))
18678
18679 #ifdef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
18680 #define INTERNAL_CATCH_EXPAND_VARGS(...) __VA_ARGS__
18681 // MSVC needs more evaluations
18682 #define CATCH_RECURSION_LEVEL6(...)
18683     CATCH_RECURSION_LEVEL5(CATCH_RECURSION_LEVEL5(CATCH_RECURSION_LEVEL5(__VA_ARGS__)))
18684 #define CATCH_RECURSE(...) CATCH_RECURSION_LEVEL6(CATCH_RECURSION_LEVEL6(__VA_ARGS__))
18685 #else
18686 #define CATCH_RECURSE(...) CATCH_RECURSION_LEVEL5(__VA_ARGS__)
18687 #endif
18688
18689 #define CATCH_REC_END(...)
18690 #define CATCH_REC_OUT
18691
18692 #define CATCH_EMPTY()
18693 #define CATCH_DEFER(id) id CATCH_EMPTY()
18694
18695 #define CATCH_REC_GET_END2() 0, CATCH_REC_END
18696 #define CATCH_REC_GET_END1(...) CATCH_REC_GET_END2
18697 #define CATCH_REC_GET_END(...) CATCH_REC_GET_END1
18698 #define CATCH_REC_NEXT0(test, next, ...) next CATCH_REC_OUT
18699 #define CATCH_REC_NEXT1(test, next) CATCH_DEFER ( CATCH_REC_NEXT0 ) ( test, next, 0)
18700 #define CATCH_REC_NEXT(test, next) CATCH_REC_NEXT1(CATCH_REC_GET_END test, next)
18701
18702 #define CATCH_REC_LIST0(f, x, peek, ...) , f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1) )
18703     ( f, peek, __VA_ARGS__ )
18704 #define CATCH_REC_LIST1(f, x, peek, ...) , f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST0) )
18705     ( f, peek, __VA_ARGS__ )
18706 #define CATCH_REC_LIST2(f, x, peek, ...) f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1) )
18707     ( f, peek, __VA_ARGS__ )
18708
18709 #define CATCH_REC_LIST0_UD(f, userdata, x, peek, ...) , f(userdata, x) CATCH_DEFER (

```

```

    CATCH_REC_NEXT(peek, CATCH_REC_LIST1_UD) ) ( f, userdata, peek, __VA_ARGS__ )
18702 #define CATCH_REC_LIST1_UD(f, userdata, x, peek, ...) , f(userdata, x) CATCH_DEFER (
    CATCH_REC_NEXT(peek, CATCH_REC_LIST0_UD) ) ( f, userdata, peek, __VA_ARGS__ )
18703 #define CATCH_REC_LIST2_UD(f, userdata, x, peek, ...) f(userdata, x) CATCH_DEFER (
    CATCH_REC_NEXT(peek, CATCH_REC_LIST1_UD) ) ( f, userdata, peek, __VA_ARGS__ )
18704
18705 // Applies the function macro `f` to each of the remaining parameters, inserts commas between the
    results,
18706 // and passes userdata as the first parameter to each invocation,
18707 // e.g. CATCH_REC_LIST_UD(f, x, a, b, c) evaluates to f(x, a), f(x, b), f(x, c)
18708 #define CATCH_REC_LIST_UD(f, userdata, ...) CATCH_RECURSE(CATCH_REC_LIST2_UD(f, userdata, __VA_ARGS__,
    )()(), ()()(), ()()(), 0))
18709
18710 #define CATCH_REC_LIST(f, ...) CATCH_RECURSE(CATCH_REC_LIST2(f, __VA_ARGS__, ()()(), ()()(), ()()(),
    0))
18711
18712 #define INTERNAL_CATCH_EXPAND1(param) INTERNAL_CATCH_EXPAND2(param)
18713 #define INTERNAL_CATCH_EXPAND2(...) INTERNAL_CATCH_NO## __VA_ARGS__
18714 #define INTERNAL_CATCH_DEF(...) INTERNAL_CATCH_DEF __VA_ARGS__
18715 #define INTERNAL_CATCH_NOINTERNAL_CATCH_DEF
18716 #define INTERNAL_CATCH_STRINGIZE(...) INTERNAL_CATCH_STRINGIZE2(__VA_ARGS__)
18717 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
18718 #define INTERNAL_CATCH_STRINGIZE2(...) #__VA_ARGS__
18719 #define INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS(param)
    INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_REMOVE_PARENS(param))
18720 #else
18721 // MSVC is adding extra space and needs another indirection to expand
    INTERNAL_CATCH_NOINTERNAL_CATCH_DEF
18722 #define INTERNAL_CATCH_STRINGIZE2(...) INTERNAL_CATCH_STRINGIZE3(__VA_ARGS__)
18723 #define INTERNAL_CATCH_STRINGIZE3(...) #__VA_ARGS__
18724 #define INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS(param)
    (INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_REMOVE_PARENS(param)) + 1)
18725 #endif
18726
18727 #define INTERNAL_CATCH_MAKE_NAMESPACE2(...) ns_##__VA_ARGS__
18728 #define INTERNAL_CATCH_MAKE_NAMESPACE(name) INTERNAL_CATCH_MAKE_NAMESPACE2(name)
18729
18730 #define INTERNAL_CATCH_REMOVE_PARENS(...) INTERNAL_CATCH_EXPAND1(INTERNAL_CATCH_DEF __VA_ARGS__)
18731
18732 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
18733 #define INTERNAL_CATCH_MAKE_TYPE_LIST2(...)
    decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS_GEN(__VA_ARGS__)>())
18734 #define INTERNAL_CATCH_MAKE_TYPE_LIST(...)
    INTERNAL_CATCH_MAKE_TYPE_LIST2(INTERNAL_CATCH_REMOVE_PARENS(__VA_ARGS__))
18735 #else
18736 #define INTERNAL_CATCH_MAKE_TYPE_LIST2(...)
    INTERNAL_CATCH_EXPAND_VARS(decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS_GEN(__VA_ARGS__)>()))
18737 #define INTERNAL_CATCH_MAKE_TYPE_LIST(...)
    INTERNAL_CATCH_EXPAND_VARS(INTERNAL_CATCH_MAKE_TYPE_LIST2(INTERNAL_CATCH_REMOVE_PARENS(__VA_ARGS__)))
18738 #endif
18739
18740 #define INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(...) \
18741     CATCH_REC_LIST(INTERNAL_CATCH_MAKE_TYPE_LIST, __VA_ARGS__)
18742
18743 #define INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_0) INTERNAL_CATCH_REMOVE_PARENS(_0)
18744 #define INTERNAL_CATCH_REMOVE_PARENS_2_ARG(_0, _1) INTERNAL_CATCH_REMOVE_PARENS(_0),
    INTERNAL_CATCH_REMOVE_PARENS(_1)
18745 #define INTERNAL_CATCH_REMOVE_PARENS_3_ARG(_0, _1, _2) INTERNAL_CATCH_REMOVE_PARENS(_0),
    INTERNAL_CATCH_REMOVE_PARENS(_1, _2)
18746 #define INTERNAL_CATCH_REMOVE_PARENS_4_ARG(_0, _1, _2, _3) INTERNAL_CATCH_REMOVE_PARENS(_0),
    INTERNAL_CATCH_REMOVE_PARENS(_1, _2, _3)
18747 #define INTERNAL_CATCH_REMOVE_PARENS_5_ARG(_0, _1, _2, _3, _4) INTERNAL_CATCH_REMOVE_PARENS(_0),
    INTERNAL_CATCH_REMOVE_PARENS(_1, _2, _3, _4)
18748 #define INTERNAL_CATCH_REMOVE_PARENS_6_ARG(_0, _1, _2, _3, _4, _5) INTERNAL_CATCH_REMOVE_PARENS(_0),
    INTERNAL_CATCH_REMOVE_PARENS(_1, _2, _3, _4, _5)
18749 #define INTERNAL_CATCH_REMOVE_PARENS_7_ARG(_0, _1, _2, _3, _4, _5, _6)
    INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS(_1, _2, _3, _4, _5, _6)
18750 #define INTERNAL_CATCH_REMOVE_PARENS_8_ARG(_0, _1, _2, _3, _4, _5, _6, _7)
    INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS(_1, _2, _3, _4, _5, _6, _7)
18751 #define INTERNAL_CATCH_REMOVE_PARENS_9_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8)
    INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS(_1, _2, _3, _4, _5, _6, _7, _8)
18752 #define INTERNAL_CATCH_REMOVE_PARENS_10_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9)
    INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS(_1, _2, _3, _4, _5, _6, _7, _8,
    _9)
18753 #define INTERNAL_CATCH_REMOVE_PARENS_11_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10)
    INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS(_1, _2, _3, _4, _5, _6, _7, _8,
    _9, _10)
18754
18755 #define INTERNAL_CATCH_VA_NARGS_IMPL(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, N, ...) N
18756
18757 #define INTERNAL_CATCH_TYPE_GEN\
18758     template<typename...> struct TypeList {};\
18759     template<typename...Ts>\
18760     constexpr auto get_wrapper() noexcept -> TypeList<Ts...> { return {}; }\
18761     template<template<typename...> class...> struct TemplateTypeList{};\
18762     template<template<typename...> class...Cs>\
18763     constexpr auto get_wrapper() noexcept -> TemplateTypeList<Cs...> { return {}; }\

```

```

18764     template<typename...>\
18765     struct append;\
18766     template<typename...>\
18767     struct rewrap;\
18768     template<template<typename...> class, typename...>\
18769     struct create;\
18770     template<template<typename...> class, typename>\
18771     struct convert;\
18772     \
18773     template<typename T> \
18774     struct append<T> { using type = T; };\
18775     template< template<typename...> class L1, typename...E1, template<typename...> class L2,
typename...E2, typename...Rest>\
18776     struct append<L1<E1...>, L2<E2...>, Rest...> { using type = typename append<L1<E1...,E2...>,
Rest...>::type; };\
18777     template< template<typename...> class L1, typename...E1, typename...Rest>\
18778     struct append<L1<E1...>, TypeList<mpl_::na>, Rest...> { using type = L1<E1...>; };\
18779     \
18780     template< template<typename...> class Container, template<typename...> class List,
typename...elems>\
18781     struct rewrap<TemplateTypeList<Container>, List<elems...> { using type =
TypeList<Container<elems...>; };\
18782     template< template<typename...> class Container, template<typename...> class List, class...Elems,
typename...Elements>\
18783     struct rewrap<TemplateTypeList<Container>, List<Elems...>, Elements...> { using type = typename
append<TypeList<Container<Elems...>, typename rewrap<TemplateTypeList<Container>,
Elements...>::type>::type; };\
18784     \
18785     template<template <typename...> class Final, template< typename...> class...Containers,
typename...Types>\
18786     struct create<Final, TemplateTypeList<Containers...>, TypeList<Types...> { using type = typename
append<Final<>, typename rewrap<TemplateTypeList<Containers>, Types...>::type...>::type; };\
18787     template<typename...> class Final, template <typename...> class List, typename...Ts>\
18788     struct convert<Final, List<Ts...> { using type = typename append<Final<>,TypeList<Ts>...>::type;
};
18789
18790 #define INTERNAL_CATCH_NTTP_1(signature, ...)\
18791     template<INTERNAL_CATCH_REMOVE_PARENS(signature)> struct Nttp{};\
18792     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
18793     constexpr auto get_wrapper() noexcept -> Nttp<__VA_ARGS__> { return {}; } \
18794     template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class...> struct
NttpTemplateTypeList{};\
18795     template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class...Cs>\
18796     constexpr auto get_wrapper() noexcept -> NttpTemplateTypeList<Cs...> { return {}; } \
18797     \
18798     template< template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class Container,
template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class List,
INTERNAL_CATCH_REMOVE_PARENS(signature)>\
18799     struct rewrap<NttpTemplateTypeList<Container>, List<__VA_ARGS__> { using type =
TypeList<Container<__VA_ARGS__>; };\
18800     template< template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class Container,
template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class List, INTERNAL_CATCH_REMOVE_PARENS(signature),
typename...Elements>\
18801     struct rewrap<NttpTemplateTypeList<Container>, List<__VA_ARGS__>, Elements...> { using type =
typename append<TypeList<Container<__VA_ARGS__>, typename rewrap<NttpTemplateTypeList<Container>,
Elements...>::type>::type; };\
18802     template<template <typename...> class Final, template<INTERNAL_CATCH_REMOVE_PARENS(signature)>
class...Containers, typename...Types>\
18803     struct create<Final, NttpTemplateTypeList<Containers...>, TypeList<Types...> { using type =
typename append<Final<>, typename rewrap<NttpTemplateTypeList<Containers>, Types...>::type...>::type;
};
18804
18805 #define INTERNAL_CATCH_DECLARE_SIG_TEST0(TestName)
18806 #define INTERNAL_CATCH_DECLARE_SIG_TEST1(TestName, signature)\
18807     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
18808     static void TestName()
18809 #define INTERNAL_CATCH_DECLARE_SIG_TEST_X(TestName, signature, ...)\
18810     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
18811     static void TestName()
18812
18813 #define INTERNAL_CATCH_DEFINE_SIG_TEST0(TestName)
18814 #define INTERNAL_CATCH_DEFINE_SIG_TEST1(TestName, signature)\
18815     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
18816     static void TestName()
18817 #define INTERNAL_CATCH_DEFINE_SIG_TEST_X(TestName, signature,...)\
18818     template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
18819     static void TestName()
18820
18821 #define INTERNAL_CATCH_NTTP_REGISTER0(TestFunc, signature)\
18822     template<typename Type>\
18823     void reg_test(TypeList<Type>, Catch::NameAndTags nameAndTags)\
18824     {\
18825         Catch::AutoReg( Catch::makeTestInvoker(&TestFunc<Type>), CATCH_INTERNAL_LINEINFO,
Catch::StringRef(), nameAndTags);\
18826     }
18827
18828 #define INTERNAL_CATCH_NTTP_REGISTER(TestFunc, signature, ...)\

```



```

INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_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__)
18879 #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, __VA_ARGS__)
18880 #else
18881 #define INTERNAL_CATCH_NTTP_0(signature)
18882 #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__))
18883 #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_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0)(TestName,
__VA_ARGS__))
18884 #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_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__))
18885 #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__))
18886 #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__))
18887 #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__))
18888 #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__))
18889 #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, __VA_ARGS__))
18890 #endif
18891
18892 // end catch_preprocessor.hpp
18893 // start catch_meta.hpp
18894
18895
18896 #include <type_traits>
18897
18898 namespace Catch {
18899     template<typename T>
18900     struct always_false : std::false_type {};
18901
18902     template<typename> struct true_given : std::true_type {};
18903     struct is_callable_tester {
18904         template<typename Fun, typename... Args>
18905         true_given<decltype(std::declval<Fun>() (std::declval<Args>()...))> static test(int);
18906         template<typename...>
18907         std::false_type static test(...);
18908     };
18909
18910     template<typename T>
18911     struct is_callable;
18912
18913     template<typename Fun, typename... Args>
18914     struct is_callable<Fun(Args...)> : decltype(is_callable_tester::test<Fun, Args...>(0)) {};
18915
18916 #if defined(__cpp_lib_is_invocable) && __cpp_lib_is_invocable >= 201703
18917     // std::result_of is deprecated in C++17 and removed in C++20. Hence, it is

```

```

18918 // replaced with std::invoke_result here.
18919 template <typename Func, typename... U>
18920 using FunctionReturnType = std::remove_reference_t<std::remove_cv_t<std::invoke_result_t<Func,
    U...>>>;
18921 #else
18922 // Keep ::type here because we still support C++11
18923 template <typename Func, typename... U>
18924 using FunctionReturnType = typename std::remove_reference<typename std::remove_cv<typename
    std::result_of<Func(U...)>::type>::type>::type;
18925 #endif
18926
18927 } // namespace Catch
18928
18929 namespace mpl_{
18930     struct na;
18931 }
18932
18933 // end catch_meta.hpp
18934 namespace Catch {
18935
18936 template<typename C>
18937 class TestInvokerAsMethod : public ITestInvoker {
18938     void (C::*m_testAsMethod)();
18939 public:
18940     TestInvokerAsMethod( void (C::*testAsMethod)() ) noexcept : m_testAsMethod( testAsMethod ) {}
18941
18942     void invoke() const override {
18943         C obj;
18944         (obj.*m_testAsMethod)();
18945     }
18946 };
18947
18948 auto makeTestInvoker( void(*testAsFunction)() ) noexcept -> ITestInvoker*;
18949
18950 template<typename C>
18951 auto makeTestInvoker( void (C::*testAsMethod)() ) noexcept -> ITestInvoker* {
18952     return new(std::nothrow) TestInvokerAsMethod<C>( testAsMethod );
18953 }
18954
18955 struct NameAndTags {
18956     NameAndTags( StringRef const& name_ = StringRef(), StringRef const& tags_ = StringRef() )
18957     noexcept;
18958     StringRef name;
18959     StringRef tags;
18960 };
18961
18962 struct AutoReg : NonCopyable {
18963     AutoReg( ITestInvoker* invoker, SourceLineInfo const& lineInfo, StringRef const& classOrMethod,
18964         NameAndTags const& nameAndTags ) noexcept;
18965     ~AutoReg();
18966 };
18967
18968 } // end namespace Catch
18969
18970 #if defined(CATCH_CONFIG_DISABLE)
18971 #define INTERNAL_CATCH_TESTCASE_NO_REGISTRATION( TestName, ... ) \
18972     static void TestName()
18973 #define INTERNAL_CATCH_TESTCASE_METHOD_NO_REGISTRATION( TestName, ClassName, ... ) \
18974     namespace{ \
18975         struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName) { \
18976             void test(); \
18977         }; \
18978         void TestName::test()
18979 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION_2( TestName, TestFunc, Name, Tags, \
18980     Signature, ... ) \
18981     INTERNAL_CATCH_DEFINE_SIG_TEST(TestFunc, INTERNAL_CATCH_REMOVE_PARENS(Signature))
18982 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2( TestNameClass, TestName, \
18983     ClassName, Name, Tags, Signature, ... ) \
18984     namespace{ \
18985         namespace INTERNAL_CATCH_MAKE_NAMESPACED(TestName) { \
18986             INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD(TestName, ClassName, \
18987                 INTERNAL_CATCH_REMOVE_PARENS(Signature)); \
18988         } \
18989         INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, INTERNAL_CATCH_REMOVE_PARENS(Signature))
18990
18991 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
18992 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(Name, Tags, ...) \
18993     INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME( \
18994         C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( \
18995         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, __VA_ARGS__ )
18996 #else
18997 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(Name, Tags, ...) \
18998     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION_2( \
18999         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( \
19000         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, __VA_ARGS__ ) )

```

```

18994     #endif
18995
18996     #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
18997         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(Name, Tags, Signature, ...) \
18998             INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME(
18999                 C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
19000                 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__ )
19001     #else
19002         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(Name, Tags, Signature, ...) \
19003             INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION_2(
19004                 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(
19005                 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__ ) )
19006     #endif
19007
19008     #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
19009         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION( ClassName, Name, Tags,... )
19010     \
19011         INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME(
19012             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(
19013             C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), ClassName, Name, Tags, typename T, __VA_ARGS__ )
19014     #else
19015         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION( ClassName, Name, Tags,... )
19016     \
19017         INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2(
19018             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_ ),
19019             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,
19020             __VA_ARGS__ ) )
19021     #endif
19022
19023     #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
19024         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION( ClassName, Name, Tags,
19025             Signature, ... ) \
19026             INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME(
19027                 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(
19028                 C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), ClassName, Name, Tags, Signature, __VA_ARGS__ )
19029     #else
19030         #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION( ClassName, Name, Tags,
19031             Signature, ... ) \
19032             INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2(
19033                 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_ ),
19034                 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,
19035                 __VA_ARGS__ ) )
19036     #endif
19037 #endif
19038
19039 #define INTERNAL_CATCH_TESTCASE2( TestName, ... ) \
19040     static void TestName(); \
19041     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
19042     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
19043     namespace{ Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker(
19044         &TestName ), CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ __VA_ARGS__ } ); } /*
19045     NOLINT */ \
19046     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
19047     static void TestName()
19048 #define INTERNAL_CATCH_TESTCASE( ... ) \
19049     INTERNAL_CATCH_TESTCASE2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ), __VA_ARGS__ )
19050
19051 #define INTERNAL_CATCH_METHOD_AS_TEST_CASE( QualifiedMethod, ... ) \
19052     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
19053     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
19054     namespace{ Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker(
19055         &QualifiedMethod ), CATCH_INTERNAL_LINEINFO, "&" #QualifiedMethod, Catch::NameAndTags{ __VA_ARGS__ }
19056     ); } /* NOLINT */ \
19057     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
19058
19059 #define INTERNAL_CATCH_TEST_CASE_METHOD2( TestName, ClassName, ... )\
19060     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
19061     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
19062     namespace{ \
19063         struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName) { \
19064             void test(); \
19065         }; \
19066         Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar ) ( Catch::makeTestInvoker(
19067             &TestName::test ), CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ __VA_ARGS__ } ); /* NOLINT
19068     */ \
19069     } \
19070     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
19071     void TestName::test()
19072 #define INTERNAL_CATCH_TEST_CASE_METHOD( ClassName, ... ) \
19073     INTERNAL_CATCH_TEST_CASE_METHOD2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ), ClassName,
19074     __VA_ARGS__ )
19075
19076 #define INTERNAL_CATCH_REGISTER_TESTCASE( Function, ... ) \
19077     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
19078     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
19079     Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker( Function
19080     ), CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ __VA_ARGS__ } ); /* NOLINT */ \

```

```

19059     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
19060
19061     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( TestName, TestFunc, Name, Tags, Signature, ... )\
19062     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
19063     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
19064     CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
19065     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
19066     INTERNAL_CATCH_DECLARE_SIG_TEST( TestFunc, INTERNAL_CATCH_REMOVE_PARENS( Signature ) );\
19067     namespace {\
19068     namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestName ) {\
19069         INTERNAL_CATCH_TYPE_GEN\
19070         INTERNAL_CATCH_NTTP_GEN( INTERNAL_CATCH_REMOVE_PARENS( Signature ) )\
19071         INTERNAL_CATCH_NTTP_REG_GEN( TestFunc, INTERNAL_CATCH_REMOVE_PARENS( Signature ) )\
19072         template<typename... Types> \
19073         struct TestName {\
19074             TestName() {\
19075                 int index = 0; \
19076                 constexpr char const* tmpl_types[] = \
19077 {CATCH_REC_LIST( INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, __VA_ARGS__ )};\
19078                 using expander = int[];\
19079                 (void)expander{ (reg_test( Types{}, Catch::NameAndTags{ Name " - " + \
std::string( tmpl_types[ index ], Tags ) ), index++ )... }; /* NOLINT */ \
19080             }\
19081             static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = []() {\
19082                 TestName<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES( __VA_ARGS__ )>();\
19083                 return 0;\
19084             }();\
19085         };\
19086     }\
19087     }\
19088     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
19089     INTERNAL_CATCH_DEFINE_SIG_TEST( TestFunc, INTERNAL_CATCH_REMOVE_PARENS( Signature ) )
19090
19091 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
19092     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE( Name, Tags, ... ) \
19093     INTERNAL_CATCH_TEMPLATE_TEST_CASE_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, typename TestType, __VA_ARGS__ )
19094 #else
19095     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE( Name, Tags, ... ) \
19096     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_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, typename TestType, __VA_ARGS__ ) )
19097 #endif
19098
19099 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
19100     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( Name, Tags, Signature, ... ) \
19101     INTERNAL_CATCH_TEMPLATE_TEST_CASE_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__ )
19102 #else
19103     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( Name, Tags, Signature, ... ) \
19104     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_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__ ) )
19105 #endif
19106
19107     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2( TestName, TestFuncName, Name, Tags, Signature, \
TmplTypes, TypesList ) \
19108     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
19109     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
19110     CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
19111     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
19112     template<typename TestType> static void TestFuncName(); \
19113     namespace {\
19114     namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestName ) {\
19115         INTERNAL_CATCH_TYPE_GEN \
19116         INTERNAL_CATCH_NTTP_GEN( INTERNAL_CATCH_REMOVE_PARENS( Signature ) ) \
19117         template<typename... Types> \
19118         struct TestName {\
19119             void reg_tests() {\
19120                 int index = 0; \
19121                 using expander = int[]; \
19122                 constexpr char const* tmpl_types[] = \
19123 {CATCH_REC_LIST( INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS( TmplTypes ) )}; \
19124                 constexpr char const* types_list[] = \
19125 {CATCH_REC_LIST( INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS( TypesList ) )}; \
19126                 constexpr auto num_types = sizeof( types_list ) / sizeof( types_list[0] ); \
19127                 (void)expander{ (Catch::AutoReg( Catch::makeTestInvoker( &TestFuncName<Types> ), \
CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ Name " - " + \
std::string( tmpl_types[ index / num_types ], "<" + std::string( types_list[ index % num_types ] ) + ">", \
Tags ) ), index++ )... }; /* NOLINT */ \
19128             }\
19129             static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = []() {\
19130                 using TestInit = typename create<TestName, \
decltype( get_wrapper<INTERNAL_CATCH_REMOVE_PARENS( TmplTypes )>() ),

```

```

TypeList<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(INTERNAL_CATCH_REMOVE_PARENS(TypesList))>::type; \
19130         TestInit t; \
19131         t.reg_tests(); \
19132         return 0; \
19133     }(); \
19134 } \
19135 } \
19136 CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
19137 template<typename TestType> \
19138 static void TestFuncName() \
19139 \
19140 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
19141     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE(Name, Tags, ...) \
19142     INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(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, typename T, __VA_ARGS__ )
19143 #else
19144     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE(Name, Tags, ...) \
19145     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(
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, typename T, __VA_ARGS__ ) )
19146 #endif
19147 \
19148 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
19149     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(Name, Tags, Signature, ...) \
19150     INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(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__ )
19151 #else
19152     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(Name, Tags, Signature, ...) \
19153     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(
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__ ) )
19154 #endif
19155 \
19156     #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_2( TestName, TestFunc, Name, Tags, TmplList ) \
19157     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
19158     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
19159     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
19160     template<typename TestType> static void TestFunc(); \
19161     namespace { \
19162     namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestName ) { \
19163     INTERNAL_CATCH_TYPE_GEN \
19164     template<typename... Types> \
19165     struct TestName { \
19166         void reg_tests() { \
19167             int index = 0; \
19168             using expander = int[]; \
19169             (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestFunc<Types> ),
CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ Name " - " +
std::string(INTERNAL_CATCH_STRINGIZE(TmplList)) + " - " + std::to_string(index), Tags } ), index++)...
}; /* NOLINT */ \
19170         } \
19171     }; \
19172     static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = []() { \
19173         using TestInit = typename convert<TestName, TmplList>::type; \
19174         TestInit t; \
19175         t.reg_tests(); \
19176         return 0; \
19177     }(); \
19178     } \
19179     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
19180     template<typename TestType> \
19181     static void TestFunc() \
19182     \
19183     #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE(Name, Tags, TmplList) \
19184     INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_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, TmplList )
19185 \
19186     #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2( TestNameClass, TestName, ClassName, Name,
Tags, Signature, ... ) \
19187     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
19188     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
19189     CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
19190     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
19191     namespace { \
19192     namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestName ) { \
19193     INTERNAL_CATCH_TYPE_GEN \
19194     INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS(Signature)) \
19195     INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD( TestName, ClassName,
INTERNAL_CATCH_REMOVE_PARENS(Signature)); \
19196     INTERNAL_CATCH_NTTP_REG_METHOD_GEN( TestName, INTERNAL_CATCH_REMOVE_PARENS(Signature)) \
19197     template<typename... Types> \
19198     struct TestNameClass { \
19199         TestNameClass() { \
19200             int index = 0; \

```

```

19201         constexpr char const* tmpl_types[] =
19202 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, __VA_ARGS__)};\
19203         using expander = int[];\
19204         (void)expander{(reg_test(Types{}, #ClassName, Catch::NameAndTags{ Name " - " +
std::string(tmpl_types[index]), Tags } ), index++)... };\
19205     }\
19206     static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
19207         TestNameClass<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(__VA_ARGS__)>();\
19208         return 0;\
19209     }();\
19210 }\
19211 }\
19212 CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
19213 INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, INTERNAL_CATCH_REMOVE_PARENS(Signature))
19214
19215 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
19216 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( ClassName, Name, Tags,... ) \
19217     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__ )
19218 #else
19219 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( ClassName, Name, Tags,... ) \
19220     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__ ) )
19221 #endif
19222
19223 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
19224 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature, ... ) \
19225     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__ )
19226 #else
19227 #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature, ... ) \
19228     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__ ) )
19229 #endif
19230
19231 #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2( TestNameClass, TestName, ClassName,
Name, Tags, Signature, TmplTypes, TypesList)\
19232     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
19233     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
19234     CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
19235     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
19236     template<typename TestType> \
19237     struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName <TestType>) { \
19238         void test();\
19239     };\
19240     namespace {\
19241         namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestNameClass ) {\
19242             INTERNAL_CATCH_TYPE_GEN \
19243             INTERNAL_CATCH_NTTP_GEN( INTERNAL_CATCH_REMOVE_PARENS( Signature ) )\
19244             template<typename... Types>\
19245             struct TestNameClass{\
19246                 void reg_tests(){\
19247                     int index = 0;\
19248                     using expander = int[];\
19249                     constexpr char const* tmpl_types[] =
19250 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TmplTypes))};\
19251                     constexpr char const* types_list[] =
19252 {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TypesList))};\
19253                     constexpr auto num_types = sizeof(types_list) / sizeof(types_list[0]);\
19254                     (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestName<Types>::test ),
CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ Name " - " + std::string(tmpl_types[index /
num_types]) + "<" + std::string(types_list[index % num_types]) + ">", Tags } ), index++)... };\
19255                     NOLINT */ \
19256                 }\
19257             };\
19258             static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
19259                 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;\
19260                 TestInit t;\
19261                 t.reg_tests();\
19262                 return 0;\
19263             }(); \
19264         }\
19265     }\
19266     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
19267     template<typename TestType> \
19268     void TestName<TestType>::test()
19269 #endif
19270 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR

```

```

19268     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( ClassName, Name, Tags, ... )\
19269     INTERNAL_CATCH_TEMPLATE_PRODUCT_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_ ), 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, Name, Tags, typename T, __VA_ARGS__ )
19270 #else
19271     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( ClassName, Name, Tags, ... )\
19272     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_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_ ), 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, Name, Tags, typename T, __VA_ARGS__ ) )
19273 #endif
19274
19275 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
19276     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature,
... )\
19277     INTERNAL_CATCH_TEMPLATE_PRODUCT_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_ ), 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, Name, Tags, Signature, __VA_ARGS__ )
19278 #else
19279     #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature,
... )\
19280     INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_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_ ), 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, Name, Tags, Signature, __VA_ARGS__ ) )
19281 #endif
19282
19283     #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_2( TestNameClass, TestName, ClassName, Name,
Tags, TmplList) \
19284     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
19285     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
19286     CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
19287     template<typename TestType> \
19288     struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName <TestType>) { \
19289         void test(); \
19290     }; \
19291     namespace { \
19292     namespace INTERNAL_CATCH_MAKE_NAMESPACE( TestName ) { \
19293         INTERNAL_CATCH_TYPE_GEN \
19294         template<typename... Types> \
19295         struct TestNameClass { \
19296             void reg_tests() { \
19297                 int index = 0; \
19298                 using expander = int[]; \
19299                 (void)expander{ (Catch::AutoReg( Catch::makeTestInvoker( &TestName<Types>::test ),
CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ Name " - " +
std::string(INTERNAL_CATCH_STRINGIZE(TmplList)) + " - " + std::to_string(index), Tags } ), index++)...
}; /* NOLINT */ \
19300             } \
19301         }; \
19302         static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = []() { \
19303             using TestInit = typename convert<TestNameClass, TmplList>::type; \
19304             TestInit t; \
19305             t.reg_tests(); \
19306             return 0; \
19307         }(); \
19308     } \
19309     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
19310     template<typename TestType> \
19311     void TestName<TestType>::test()
19312
19313 #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD(ClassName, Name, Tags, TmplList) \
19314     INTERNAL_CATCH_TEMPLATE_LIST_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_ ), 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, Name, Tags, TmplList )
19315
19316 // end catch_test_registry.h
19317 // start catch_capture.hpp
19318
19319 // start catch_assertionhandler.h
19320
19321 // start catch_assertioninfo.h
19322
19323 // start catch_result_type.h
19324
19325 namespace Catch {
19326
19327     // ResultWas::OfType enum
19328     struct ResultWas { enum OfType {
19329         Unknown = -1,
19330         Ok = 0,
19331         Info = 1,
19332         Warning = 2,
19333
19334         FailureBit = 0x10,
19335
19336         ExpressionFailed = FailureBit | 1,
19337         ExplicitFailure = FailureBit | 2,
19338     };

```



```

19339         Exception = 0x100 | FailureBit,
19340
19341         ThrewException = Exception | 1,
19342         DidntThrowException = Exception | 2,
19343
19344         FatalErrorCondition = 0x200 | FailureBit
19345     }; };
19346
19347     bool isOk( ResultWas::OfType resultType );
19348     bool isJustInfo( int flags );
19349
19350     // ResultDisposition::Flags enum
19351     struct ResultDisposition { enum Flags {
19352         Normal = 0x01,
19353
19354         ContinueOnFailure = 0x02,    // Failures fail test, but execution continues
19355         FalseTest = 0x04,            // Prefix expression with !
19356         SuppressFail = 0x08          // Failures are reported but do not fail the test
19357     }; };
19358
19359     ResultDisposition::Flags operator | ( ResultDisposition::Flags lhs, ResultDisposition::Flags rhs
19360 );
19361
19362     bool shouldContinueOnFailure( int flags );
19363     inline bool isFalseTest( int flags ) { return ( flags & ResultDisposition::FalseTest ) != 0; }
19364     bool shouldSuppressFailure( int flags );
19365
19366 } // end namespace Catch
19367
19368 // end catch_result_type.h
19369 namespace Catch {
19370
19371     struct AssertionInfo
19372     {
19373        StringRef macroName;
19374        SourceLineInfo lineInfo;
19375        StringRef capturedExpression;
19376        ResultDisposition::Flags resultDisposition;
19377
19378         // We want to delete this constructor but a compiler bug in 4.8 means
19379         // the struct is then treated as non-aggregate
19380         //AssertionInfo() = delete;
19381     };
19382
19383 } // end namespace Catch
19384
19385 // end catch_assertioninfo.h
19386 // start catch_decomposer.h
19387
19388 // start catch_tostring.h
19389
19390 #include <vector>
19391 #include <cstdint>
19392 #include <type_traits>
19393 #include <string>
19394 // start catch_stream.h
19395
19396 #include <iosfwd>
19397 #include <cstdint>
19398 #include <ostream>
19399
19400 namespace Catch {
19401
19402     std::ostream& cout();
19403     std::ostream& cerr();
19404     std::ostream& clog();
19405
19406     class StringRef;
19407
19408     struct IStream {
19409         virtual ~IStream();
19410         virtual std::ostream& stream() const = 0;
19411     };
19412
19413     auto makeStream( StringRef const& filename ) -> IStream const*;
19414
19415     class ReusableStringStream : NonCopyable {
19416     public:
19417         ReusableStringStream();
19418         ~ReusableStringStream();
19419
19420         auto str() const -> std::string;
19421
19422     private:
19423         template<typename T>

```



```

19425         auto operator « ( T const& value ) -> ReusableStringStream& {
19426             *m_oss « value;
19427             return *this;
19428         }
19429         auto get() -> std::ostream& { return *m_oss; }
19430     };
19431 }
19432
19433 // end catch_stream.h
19434 // start catch_interfaces_enum_values_registry.h
19435
19436 #include <vector>
19437
19438 namespace Catch {
19439
19440     namespace Detail {
19441         struct EnumInfo {
19442            StringRef m_name;
19443             std::vector<std::pair<int, StringRef>> m_values;
19444
19445             ~EnumInfo();
19446
19447             StringRef lookup( int value ) const;
19448         };
19449     } // namespace Detail
19450
19451     struct IMutableEnumValuesRegistry {
19452         virtual ~IMutableEnumValuesRegistry();
19453
19454         virtual Detail::EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums,
19455             std::vector<int> const& values ) = 0;
19456
19457         template<typename E>
19458         Detail::EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums,
19459             std::initializer_list<E> values ) {
19460             static_assert(sizeof(int) >= sizeof(E), "Cannot serialize enum to int");
19461             std::vector<int> intValues;
19462             intValues.reserve( values.size() );
19463             for( auto enumValue : values )
19464                 intValues.push_back( static_cast<int>( enumValue ) );
19465             return registerEnum( enumName, allEnums, intValues );
19466         }
19467     };
19468
19469 // end catch_interfaces_enum_values_registry.h
19470
19471 #ifdef CATCH_CONFIG_CPP17_STRING_VIEW
19472 #include <string_view>
19473 #endif
19474
19475 #ifdef __OBJC__
19476 // start catch_objc_arc.hpp
19477
19478 #import <Foundation/Foundation.h>
19479
19480 #ifdef __has_feature
19481 #define CATCH_ARC_ENABLED __has_feature(objc_arc)
19482 #else
19483 #define CATCH_ARC_ENABLED 0
19484 #endif
19485
19486 void arcSafeRelease( NSObject* obj );
19487 id performOptionalSelector( id obj, SEL sel );
19488
19489 #if !CATCH_ARC_ENABLED
19490 inline void arcSafeRelease( NSObject* obj ) {
19491     [obj release];
19492 }
19493 inline id performOptionalSelector( id obj, SEL sel ) {
19494     if( [obj respondsToSelector: sel] )
19495         return [obj performSelector: sel];
19496     return nil;
19497 }
19498 #define CATCH_UNSAFE_UNRETAINED
19499 #define CATCH_ARC_STRONG
19500 #else
19501 inline void arcSafeRelease( NSObject* ){}
19502 inline id performOptionalSelector( id obj, SEL sel ) {
19503     #ifdef __clang__
19504     #pragma clang diagnostic push
19505     #pragma clang diagnostic ignored "-Warc-performSelector-leaks"
19506     #endif
19507     if( [obj respondsToSelector: sel] )
19508         return [obj performSelector: sel];
19509     #ifdef __clang__

```

```

19510 #pragma clang diagnostic pop
19511 #endif
19512     return nil;
19513 }
19514 #define CATCH_UNSAFE_UNRETAINED __unsafe_unretained
19515 #define CATCH_ARC_STRONG __strong
19516 #endif
19517
19518 // end catch_objc_arc.hpp
19519 #endif
19520
19521 #ifndef _MSC_VER
19522 #pragma warning(push)
19523 #pragma warning(disable:4180) // We attempt to stream a function (address) by const&, which MSVC
    complains about but is harmless
19524 #endif
19525
19526 namespace Catch {
19527     namespace Detail {
19528
19529         extern const std::string unprintableString;
19530
19531         std::string rawMemoryToString( const void *object, std::size_t size );
19532
19533         template<typename T>
19534         std::string rawMemoryToString( const T& object ) {
19535             return rawMemoryToString( &object, sizeof(object) );
19536         }
19537
19538         template<typename T>
19539         class IsStreamInsertable {
19540             template<typename Stream, typename U>
19541             static auto test(int)
19542                 -> decltype(std::declval<Stream&>() << std::declval<U>(), std::true_type());
19543
19544             template<typename, typename>
19545             static auto test(...) -> std::false_type;
19546
19547         public:
19548             static const bool value = decltype(test<std::ostream, const T&>(0))::value;
19549         };
19550
19551         template<typename E>
19552         std::string convertUnknownEnumToString( E e );
19553
19554         template<typename T>
19555         typename std::enable_if<
19556             !std::is_enum<T>::value && !std::is_base_of<std::exception, T>::value,
19557             std::string>::type convertUnstreamable( T const& ) {
19558             return Detail::unprintableString;
19559         }
19560
19561         template<typename T>
19562         typename std::enable_if<
19563             !std::is_enum<T>::value && std::is_base_of<std::exception, T>::value,
19564             std::string>::type convertUnstreamable( T const& ex ) {
19565             return ex.what();
19566         }
19567
19568         template<typename T>
19569         typename std::enable_if<
19570             std::is_enum<T>::value
19571             , std::string>::type convertUnstreamable( T const& value ) {
19572             return convertUnknownEnumToString( value );
19573         }
19574
19575         #if defined(_MANAGED)
19576         template<typename T>
19577         std::string clrReferenceToString( T^ ref ) {
19578             if (ref == nullptr)
19579                 return std::string("null");
19580             auto bytes = System::Text::Encoding::UTF8->GetBytes(ref->ToString());
19581             cli::pin_ptr<System::Byte> p = &bytes[0];
19582             return std::string(reinterpret_cast<char const *>(p), bytes->Length);
19583         }
19584     #endif
19585
19586     } // namespace Detail
19587
19588     // If we decide for C++14, change these to enable_if_ts
19589     template<typename T, typename = void>
19590     struct StringMaker {
19591         template<typename Fake = T>
19592         static
19593             typename std::enable_if<:Catch::Detail::IsStreamInsertable<Fake>::value, std::string>::type
19594             convert(const Fake& value) {
19595                 ReusableStringStream rss;
19596                 // NB: call using the function-like syntax to avoid ambiguity with

```

```

19597         // user-defined templated operator« under clang.
19598         rss.operator«(value);
19599         return rss.str();
19600     }
19601
19602     template <typename Fake = T>
19603     static
19604     typename std::enable_if<!::Catch::Detail::IsStreamInsertable<Fake>::value, std::string>::type
19605     convert( const Fake& value ) {
19606     #if !defined(CATCH_CONFIG_FALLBACK_STRINGIFIER)
19607         return Detail::convertUnstreamable(value);
19608     #else
19609         return CATCH_CONFIG_FALLBACK_STRINGIFIER(value);
19610     #endif
19611     }
19612 };
19613
19614 namespace Detail {
19615
19616     // This function dispatches all stringification requests inside of Catch.
19617     // Should be preferably called fully qualified, like ::Catch::Detail::stringify
19618     template <typename T>
19619     std::string stringify(const T& e) {
19620         return ::Catch::StringMaker<typename std::remove_cv<typename
std::remove_reference<T>::type>::type>::convert(e);
19621     }
19622
19623     template<typename E>
19624     std::string convertUnknownEnumToString( E e ) {
19625         return ::Catch::Detail::stringify(static_cast<typename std::underlying_type<E>::type>(e));
19626     }
19627
19628     #if defined(_MANAGED)
19629     template <typename T>
19630     std::string stringify( T^ e ) {
19631         return ::Catch::StringMaker<T^>::convert(e);
19632     }
19633     #endif
19634
19635     } // namespace Detail
19636
19637     // Some predefined specializations
19638
19639     template<>
19640     struct StringMaker<std::string> {
19641         static std::string convert(const std::string& str);
19642     };
19643
19644     #ifdef CATCH_CONFIG_CPP17_STRING_VIEW
19645     template<>
19646     struct StringMaker<std::string_view> {
19647         static std::string convert(std::string_view str);
19648     };
19649     #endif
19650
19651     template<>
19652     struct StringMaker<char const *> {
19653         static std::string convert(char const * str);
19654     };
19655     template<>
19656     struct StringMaker<char *> {
19657         static std::string convert(char * str);
19658     };
19659
19660     #ifdef CATCH_CONFIG_WCHAR
19661     template<>
19662     struct StringMaker<std::wstring> {
19663         static std::string convert(const std::wstring& wstr);
19664     };
19665
19666     #ifdef CATCH_CONFIG_CPP17_STRING_VIEW
19667     template<>
19668     struct StringMaker<std::wstring_view> {
19669         static std::string convert(std::wstring_view str);
19670     };
19671     #endif
19672
19673     template<>
19674     struct StringMaker<wchar_t const *> {
19675         static std::string convert(wchar_t const * str);
19676     };
19677     template<>
19678     struct StringMaker<wchar_t *> {
19679         static std::string convert(wchar_t * str);
19680     };
19681     #endif
19682

```

```

19683 // TBD: Should we use `strlen` to ensure that we don't go out of the buffer,
19684 // while keeping string semantics?
19685 template<int SZ>
19686 struct StringMaker<char[SZ]> {
19687     static std::string convert(char const* str) {
19688         return ::Catch::Detail::stringify(std::string{ str });
19689     }
19690 };
19691 template<int SZ>
19692 struct StringMaker<signed char[SZ]> {
19693     static std::string convert(signed char const* str) {
19694         return ::Catch::Detail::stringify(std::string{ reinterpret_cast<char const*>(str) });
19695     }
19696 };
19697 template<int SZ>
19698 struct StringMaker<unsigned char[SZ]> {
19699     static std::string convert(unsigned char const* str) {
19700         return ::Catch::Detail::stringify(std::string{ reinterpret_cast<char const*>(str) });
19701     }
19702 };
19703
19704 #if defined(CATCH_CONFIG_CPP17_BYTE)
19705 template<>
19706 struct StringMaker<std::byte> {
19707     static std::string convert(std::byte value);
19708 };
19709 #endif // defined(CATCH_CONFIG_CPP17_BYTE)
19710 template<>
19711 struct StringMaker<int> {
19712     static std::string convert(int value);
19713 };
19714 template<>
19715 struct StringMaker<long> {
19716     static std::string convert(long value);
19717 };
19718 template<>
19719 struct StringMaker<long long> {
19720     static std::string convert(long long value);
19721 };
19722 template<>
19723 struct StringMaker<unsigned int> {
19724     static std::string convert(unsigned int value);
19725 };
19726 template<>
19727 struct StringMaker<unsigned long> {
19728     static std::string convert(unsigned long value);
19729 };
19730 template<>
19731 struct StringMaker<unsigned long long> {
19732     static std::string convert(unsigned long long value);
19733 };
19734
19735 template<>
19736 struct StringMaker<bool> {
19737     static std::string convert(bool b);
19738 };
19739
19740 template<>
19741 struct StringMaker<char> {
19742     static std::string convert(char c);
19743 };
19744 template<>
19745 struct StringMaker<signed char> {
19746     static std::string convert(signed char c);
19747 };
19748 template<>
19749 struct StringMaker<unsigned char> {
19750     static std::string convert(unsigned char c);
19751 };
19752
19753 template<>
19754 struct StringMaker<std::nullptr_t> {
19755     static std::string convert(std::nullptr_t);
19756 };
19757
19758 template<>
19759 struct StringMaker<float> {
19760     static std::string convert(float value);
19761     static int precision;
19762 };
19763
19764 template<>
19765 struct StringMaker<double> {
19766     static std::string convert(double value);
19767     static int precision;
19768 };
19769

```

```

19770     template <typename T>
19771     struct StringMaker<T*> {
19772         template <typename U>
19773         static std::string convert(U* p) {
19774             if (p) {
19775                 return ::Catch::Detail::rawMemoryToString(p);
19776             } else {
19777                 return "nullptr";
19778             }
19779         };
19780     };
19781
19782     template <typename R, typename C>
19783     struct StringMaker<R C::*> {
19784         static std::string convert(R C::* p) {
19785             if (p) {
19786                 return ::Catch::Detail::rawMemoryToString(p);
19787             } else {
19788                 return "nullptr";
19789             }
19790         };
19791     };
19792
19793 #if defined(_MANAGED)
19794     template <typename T>
19795     struct StringMaker<T^> {
19796         static std::string convert( T^ ref ) {
19797             return ::Catch::Detail::clrReferenceToString(ref);
19798         };
19799     };
19800 #endif
19801
19802     namespace Detail {
19803         template<typename InputIterator, typename Sentinel = InputIterator>
19804         std::string rangeToString(InputIterator first, Sentinel last) {
19805             ReusableStringStream rss;
19806             rss << "{ ";
19807             if (first != last) {
19808                 rss << ::Catch::Detail::stringify(*first);
19809                 for (++first; first != last; ++first)
19810                     rss << ", " << ::Catch::Detail::stringify(*first);
19811             }
19812             rss << " }";
19813             return rss.str();
19814         }
19815     }
19816
19817 #ifdef __OBJC__
19818     template<>
19819     struct StringMaker<NSString*> {
19820         static std::string convert(NSString * nsstring) {
19821             if (!nsstring)
19822                 return "nil";
19823             return std::string(@"") + [nsstring UTF8String];
19824         };
19825     };
19826     template<>
19827     struct StringMaker<NSObject*> {
19828         static std::string convert(NSObject* nsObject) {
19829             return ::Catch::Detail::stringify([nsObject description]);
19830         };
19831     };
19832
19833     namespace Detail {
19834         inline std::string stringify( NSString* nsstring ) {
19835             return StringMaker<NSString*>::convert( nsstring );
19836         }
19837     }
19838 } // namespace Detail
19839 #endif // __OBJC__
19840
19841 } // namespace Catch
19842
19843 // Separate std-lib types stringification, so it can be selectively enabled
19844 // This means that we do not bring in
19845
19846 #if defined(CATCH_CONFIG_ENABLE_ALL_STRINGMAKERS)
19847 # define CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER
19848 # define CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER
19849 # define CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER
19850 # define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
19851 # define CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER
19852 #endif
19853
19854 // Separate std::pair specialization
19855 #if defined(CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER)
19856 #include <utility>

```

```

19858 namespace Catch {
19859     template<typename T1, typename T2>
19860     struct StringMaker<std::pair<T1, T2> > {
19861         static std::string convert(const std::pair<T1, T2>& pair) {
19862             ReusableStringStream rss;
19863             rss << "{ "
19864                 << ::Catch::Detail::stringify(pair.first)
19865                 << ", "
19866                 << ::Catch::Detail::stringify(pair.second)
19867                 << " }";
19868             return rss.str();
19869         }
19870     };
19871 }
19872 #endif // CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER
19873
19874 #if defined(CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER) && defined(CATCH_CONFIG_CPP17_OPTIONAL)
19875 #include <optional>
19876 namespace Catch {
19877     template<typename T>
19878     struct StringMaker<std::optional<T> > {
19879         static std::string convert(const std::optional<T>& optional) {
19880             ReusableStringStream rss;
19881             if (optional.has_value()) {
19882                 rss << ::Catch::Detail::stringify(*optional);
19883             } else {
19884                 rss << "{ }";
19885             }
19886             return rss.str();
19887         }
19888     };
19889 }
19890 #endif // CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER
19891
19892 // Separate std::tuple specialization
19893 #if defined(CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER)
19894 #include <tuple>
19895 namespace Catch {
19896     namespace Detail {
19897         template<
19898             typename Tuple,
19899             std::size_t N = 0,
19900             bool = (N < std::tuple_size<Tuple>::value)
19901         >
19902         struct TupleElementPrinter {
19903             static void print(const Tuple& tuple, std::ostream& os) {
19904                 os << (N ? ", " : " ")
19905                     << ::Catch::Detail::stringify(std::get<N>(tuple));
19906                 TupleElementPrinter<Tuple, N + 1>::print(tuple, os);
19907             }
19908         };
19909
19910         template<
19911             typename Tuple,
19912             std::size_t N
19913         >
19914         struct TupleElementPrinter<Tuple, N, false> {
19915             static void print(const Tuple&, std::ostream&) {}
19916         };
19917
19918     }
19919
19920     template<typename ...Types>
19921     struct StringMaker<std::tuple<Types...> > {
19922         static std::string convert(const std::tuple<Types...>& tuple) {
19923             ReusableStringStream rss;
19924             rss << '{';
19925             Detail::TupleElementPrinter<std::tuple<Types...>::print(tuple, rss.get());
19926             rss << " }";
19927             return rss.str();
19928         }
19929     };
19930 }
19931 #endif // CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER
19932
19933 #if defined(CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER) && defined(CATCH_CONFIG_CPP17_VARIANT)
19934 #include <variant>
19935 namespace Catch {
19936     template<>
19937     struct StringMaker<std::monostate> {
19938         static std::string convert(const std::monostate&) {
19939             return "{ }";
19940         }
19941     };
19942
19943     template<typename... Elements>
19944     struct StringMaker<std::variant<Elements...> > {

```

```

19945     static std::string convert(const std::variant<Elements...>& variant) {
19946         if (variant.valueless_by_exception()) {
19947             return "{valueless variant}";
19948         } else {
19949             return std::visit(
19950                 [](const auto& value) {
19951                     return ::Catch::Detail::stringify(value);
19952                 },
19953                 variant
19954             );
19955         }
19956     }
19957 };
19958 }
19959 #endif // CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER
19960
19961 namespace Catch {
19962     // Import begin/ end from std here
19963     using std::begin;
19964     using std::end;
19965
19966     namespace detail {
19967         template <typename...>
19968         struct void_type {
19969             using type = void;
19970         };
19971
19972         template <typename T, typename = void>
19973         struct is_range_impl : std::false_type {
19974         };
19975
19976         template <typename T>
19977         struct is_range_impl<T, typename void_type<decltype(begin(std::declval<T>()))>::type> :
19978             std::true_type {
19979         };
19980     } // namespace detail
19981
19982     template <typename T>
19983     struct is_range : detail::is_range_impl<T> {
19984     };
19985
19986     #if defined(_MANAGED) // Managed types are never ranges
19987     template <typename T>
19988     struct is_range<T^> {
19989         static const bool value = false;
19990     };
19991 #endif
19992
19993     template<typename Range>
19994     std::string rangeToString( Range const& range ) {
19995         return ::Catch::Detail::rangeToString( begin( range ), end( range ) );
19996     }
19997
19998     // Handle vector<bool> specially
19999     template<typename Allocator>
20000     std::string rangeToString( std::vector<bool, Allocator> const& v ) {
20001         ReusableStringStream rss;
20002         rss << "{ ";
20003         bool first = true;
20004         for( bool b : v ) {
20005             if( first )
20006                 first = false;
20007             else
20008                 rss << ", ";
20009             rss << ::Catch::Detail::stringify( b );
20010         }
20011         rss << " }";
20012         return rss.str();
20013     }
20014
20015     template<typename R>
20016     struct StringMaker<R, typename std::enable_if<is_range<R>::value &&
20017         !::Catch::Detail::IsStreamInsertable<R>::value>::type> {
20018         static std::string convert( R const& range ) {
20019             return rangeToString( range );
20020         }
20021     };
20022
20023     template <typename T, int SZ>
20024     struct StringMaker<T[SZ]> {
20025         static std::string convert( T const(&arr)[SZ] ) {
20026             return rangeToString(arr);
20027         }
20028     };
20029 } // namespace Catch

```

```

20030 // Separate std::chrono::duration specialization
20031 #if defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
20032 #include <ctime>
20033 #include <ratio>
20034 #include <chrono>
20035
20036 namespace Catch {
20037
20038     template <class Ratio>
20039     struct ratio_string {
20040         static std::string symbol();
20041     };
20042
20043     template <class Ratio>
20044     std::string ratio_string<Ratio>::symbol() {
20045         Catch::ReusableStringStream rss;
20046         rss << '[' << Ratio::num << '/'
20047             << Ratio::den << ']';
20048         return rss.str();
20049     }
20050     template <>
20051     struct ratio_string<std::atto> {
20052         static std::string symbol();
20053     };
20054     template <>
20055     struct ratio_string<std::femto> {
20056         static std::string symbol();
20057     };
20058     template <>
20059     struct ratio_string<std::pico> {
20060         static std::string symbol();
20061     };
20062     template <>
20063     struct ratio_string<std::nano> {
20064         static std::string symbol();
20065     };
20066     template <>
20067     struct ratio_string<std::micro> {
20068         static std::string symbol();
20069     };
20070     template <>
20071     struct ratio_string<std::milli> {
20072         static std::string symbol();
20073     };
20074
20075     // std::chrono::duration specializations
20076     template<typename Value, typename Ratio>
20077     struct StringMaker<std::chrono::duration<Value, Ratio> {
20078         static std::string convert(std::chrono::duration<Value, Ratio> const& duration) {
20079             ReusableStringStream rss;
20080             rss << duration.count() << ' ' << ratio_string<Ratio>::symbol() << 's';
20081             return rss.str();
20082         }
20083     };
20084
20085     template<typename Value>
20086     struct StringMaker<std::chrono::duration<Value, std::ratio<1>> {
20087         static std::string convert(std::chrono::duration<Value, std::ratio<1> const& duration) {
20088             ReusableStringStream rss;
20089             rss << duration.count() << " s";
20090             return rss.str();
20091         }
20092     };
20093
20094     template<typename Value>
20095     struct StringMaker<std::chrono::duration<Value, std::ratio<60>> {
20096         static std::string convert(std::chrono::duration<Value, std::ratio<60> const& duration) {
20097             ReusableStringStream rss;
20098             rss << duration.count() << " m";
20099             return rss.str();
20100         }
20101     };
20102
20103     template<typename Value>
20104     struct StringMaker<std::chrono::duration<Value, std::ratio<3600>> {
20105         static std::string convert(std::chrono::duration<Value, std::ratio<3600> const& duration) {
20106             ReusableStringStream rss;
20107             rss << duration.count() << " h";
20108             return rss.str();
20109         }
20110     };
20111
20112     // std::chrono::time_point specialization
20113     // Generic time_point cannot be specialized, only std::chrono::time_point<system_clock>
20114     template<typename Clock, typename Duration>
20115     struct StringMaker<std::chrono::time_point<Clock, Duration> {
20116         static std::string convert(std::chrono::time_point<Clock, Duration> const& time_point) {
20117             return ::Catch::Detail::stringify(time_point.time_since_epoch()) + " since epoch";
20118         }
20119     };

```



```

20119 // std::chrono::time_point<system_clock> specialization
20120 template<typename Duration>
20121 struct StringMaker<std::chrono::time_point<std::chrono::system_clock, Duration> {
20122     static std::string convert(std::chrono::time_point<std::chrono::system_clock, Duration> const&
time_point) {
20123         auto converted = std::chrono::system_clock::to_time_t(time_point);
20124
20125         #ifdef _MSC_VER
20126             std::tm timeInfo = {};
20127             gmtime_s(&timeInfo, &converted);
20128         #else
20129             std::tm* timeInfo = std::gmtime(&converted);
20130         #endif
20131
20132         auto const timeStampSize = sizeof("2017-01-16T17:06:45Z");
20133         char timeStamp[timeStampSize];
20134         const char * const fmt = "%Y-%m-%dT%H:%M:%SZ";
20135
20136         #ifdef _MSC_VER
20137             std::strftime(timeStamp, timeStampSize, fmt, &timeInfo);
20138         #else
20139             std::strftime(timeStamp, timeStampSize, fmt, timeInfo);
20140         #endif
20141         return std::string(timeStamp);
20142     }
20143 };
20144 }
20145 #endif // CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
20146
20147 #define INTERNAL_CATCH_REGISTER_ENUM( enumName, ... ) \
20148 namespace Catch { \
20149     template<> struct StringMaker<enumName> { \
20150         static std::string convert( enumName value ) { \
20151             static const auto& enumInfo =
::Catch::getMutableRegistryHub().getMutableEnumValuesRegistry().registerEnum( #enumName, #__VA_ARGS__,
{ __VA_ARGS__ } ); \
20152             return static_cast<std::string>(enumInfo.lookup( static_cast<int>( value ) )); \
20153         } \
20154     }; \
20155 }
20156
20157 #define CATCH_REGISTER_ENUM( enumName, ... ) INTERNAL_CATCH_REGISTER_ENUM( enumName, __VA_ARGS__ )
20158
20159 #ifdef _MSC_VER
20160 #pragma warning(pop)
20161 #endif
20162
20163 // end catch_tostring.h
20164 #include <iosfwd>
20165
20166 #ifdef _MSC_VER
20167 #pragma warning(push)
20168 #pragma warning(disable:4389) // '==' : signed/unsigned mismatch
20169 #pragma warning(disable:4018) // more "signed/unsigned mismatch"
20170 #pragma warning(disable:4312) // Converting int to T* using reinterpret_cast (issue on x64 platform)
20171 #pragma warning(disable:4180) // qualifier applied to function type has no meaning
20172 #pragma warning(disable:4800) // Forcing result to true or false
20173 #endif
20174
20175 namespace Catch {
20176
20177     struct ITransientExpression {
20178         auto isBinaryExpression() const -> bool { return m_isBinaryExpression; }
20179         auto getResult() const -> bool { return m_result; }
20180         virtual void streamReconstructedExpression( std::ostream &os ) const = 0;
20181
20182         ITransientExpression( bool isBinaryExpression, bool result )
20183             : m_isBinaryExpression( isBinaryExpression ),
20184               m_result( result )
20185         {}
20186
20187         // We don't actually need a virtual destructor, but many static analysers
20188         // complain if it's not here :-|
20189         virtual ~ITransientExpression();
20190
20191         bool m_isBinaryExpression;
20192         bool m_result;
20193     };
20194
20195     void formatReconstructedExpression( std::ostream &os, std::string const& lhs, StringRef op,
std::string const& rhs );
20196
20197     template<typename LhsT, typename RhsT>
20198     class BinaryExpr : public ITransientExpression {
20199     public:
20200         LhsT m_lhs;
20201         StringRef m_op;

```

```

20202         RhsT m_rhs;
20203
20204         void streamReconstructedExpression( std::ostream &os ) const override {
20205             formatReconstructedExpression
20206                 ( os, Catch::Detail::stringify( m_lhs ), m_op, Catch::Detail::stringify( m_rhs )
20207 );
20208     }
20209 public:
20210     BinaryExpr( bool comparisonResult, LhsT lhs, StringRef op, RhsT rhs )
20211     :     ITransientExpression{ true, comparisonResult },
20212         m_lhs( lhs ),
20213         m_op( op ),
20214         m_rhs( rhs )
20215     {}
20216
20217     template<typename T>
20218     auto operator && ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
20219         static_assert(always_false<T>::value,
20220             "chained comparisons are not supported inside assertions, "
20221             "wrap the expression inside parentheses, or decompose it");
20222     }
20223
20224     template<typename T>
20225     auto operator || ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
20226         static_assert(always_false<T>::value,
20227             "chained comparisons are not supported inside assertions, "
20228             "wrap the expression inside parentheses, or decompose it");
20229     }
20230
20231     template<typename T>
20232     auto operator == ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
20233         static_assert(always_false<T>::value,
20234             "chained comparisons are not supported inside assertions, "
20235             "wrap the expression inside parentheses, or decompose it");
20236     }
20237
20238     template<typename T>
20239     auto operator != ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
20240         static_assert(always_false<T>::value,
20241             "chained comparisons are not supported inside assertions, "
20242             "wrap the expression inside parentheses, or decompose it");
20243     }
20244
20245     template<typename T>
20246     auto operator > ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
20247         static_assert(always_false<T>::value,
20248             "chained comparisons are not supported inside assertions, "
20249             "wrap the expression inside parentheses, or decompose it");
20250     }
20251
20252     template<typename T>
20253     auto operator < ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
20254         static_assert(always_false<T>::value,
20255             "chained comparisons are not supported inside assertions, "
20256             "wrap the expression inside parentheses, or decompose it");
20257     }
20258
20259     template<typename T>
20260     auto operator >= ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
20261         static_assert(always_false<T>::value,
20262             "chained comparisons are not supported inside assertions, "
20263             "wrap the expression inside parentheses, or decompose it");
20264     }
20265
20266     template<typename T>
20267     auto operator <= ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
20268         static_assert(always_false<T>::value,
20269             "chained comparisons are not supported inside assertions, "
20270             "wrap the expression inside parentheses, or decompose it");
20271     }
20272 };
20273
20274 template<typename LhsT>
20275 class UnaryExpr : public ITransientExpression {
20276     LhsT m_lhs;
20277
20278     void streamReconstructedExpression( std::ostream &os ) const override {
20279         os << Catch::Detail::stringify( m_lhs );
20280     }
20281
20282 public:
20283     explicit UnaryExpr( LhsT lhs )
20284     :     ITransientExpression{ false, static_cast<bool>(lhs) },
20285         m_lhs( lhs )
20286     {}
20287 };

```

```

20288
20289 // Specialised comparison functions to handle equality comparisons between ints and pointers (NULL
deduces as an int)
20290 template<typename LhsT, typename RhsT>
20291 auto compareEqual( LhsT const& lhs, RhsT const& rhs ) -> bool { return static_cast<bool>(lhs ==
rhs); }
20292 template<typename T>
20293 auto compareEqual( T* const& lhs, int rhs ) -> bool { return lhs == reinterpret_cast<void const*>(
rhs ); }
20294 template<typename T>
20295 auto compareEqual( T* const& lhs, long rhs ) -> bool { return lhs == reinterpret_cast<void
const*>( rhs ); }
20296 template<typename T>
20297 auto compareEqual( int lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs )
== rhs; }
20298 template<typename T>
20299 auto compareEqual( long lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs )
== rhs; }
20300
20301 template<typename LhsT, typename RhsT>
20302 auto compareNotEqual( LhsT const& lhs, RhsT&& rhs ) -> bool { return static_cast<bool>(lhs !=
rhs); }
20303 template<typename T>
20304 auto compareNotEqual( T* const& lhs, int rhs ) -> bool { return lhs != reinterpret_cast<void
const*>( rhs ); }
20305 template<typename T>
20306 auto compareNotEqual( T* const& lhs, long rhs ) -> bool { return lhs != reinterpret_cast<void
const*>( rhs ); }
20307 template<typename T>
20308 auto compareNotEqual( int lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs
) != rhs; }
20309 template<typename T>
20310 auto compareNotEqual( long lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>(
lhs ) != rhs; }
20311
20312 template<typename LhsT>
20313 class ExprLhs {
20314     LhsT m_lhs;
20315 public:
20316     explicit ExprLhs( LhsT lhs ) : m_lhs( lhs ) {}
20317
20318     template<typename RhsT>
20319     auto operator == ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
20320         return { compareEqual( m_lhs, rhs ), m_lhs, "==", rhs };
20321     }
20322     auto operator == ( bool rhs ) -> BinaryExpr<LhsT, bool> const {
20323         return { m_lhs == rhs, m_lhs, "==", rhs };
20324     }
20325
20326     template<typename RhsT>
20327     auto operator != ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
20328         return { compareNotEqual( m_lhs, rhs ), m_lhs, "!=", rhs };
20329     }
20330     auto operator != ( bool rhs ) -> BinaryExpr<LhsT, bool> const {
20331         return { m_lhs != rhs, m_lhs, "!=", rhs };
20332     }
20333
20334     template<typename RhsT>
20335     auto operator > ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
20336         return { static_cast<bool>(m_lhs > rhs), m_lhs, ">", rhs };
20337     }
20338     template<typename RhsT>
20339     auto operator < ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
20340         return { static_cast<bool>(m_lhs < rhs), m_lhs, "<", rhs };
20341     }
20342     template<typename RhsT>
20343     auto operator >= ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
20344         return { static_cast<bool>(m_lhs >= rhs), m_lhs, ">=", rhs };
20345     }
20346     template<typename RhsT>
20347     auto operator <= ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
20348         return { static_cast<bool>(m_lhs <= rhs), m_lhs, "<=", rhs };
20349     }
20350     template <typename RhsT>
20351     auto operator | (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
20352         return { static_cast<bool>(m_lhs | rhs), m_lhs, "|", rhs };
20353     }
20354     template <typename RhsT>
20355     auto operator & (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
20356         return { static_cast<bool>(m_lhs & rhs), m_lhs, "&", rhs };
20357     }
20358     template <typename RhsT>
20359     auto operator ^ (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
20360         return { static_cast<bool>(m_lhs ^ rhs), m_lhs, "^", rhs };
20361     }
20362
20363     template<typename RhsT>

```

```

20364     auto operator && ( RhsT const& ) -> BinaryExpr<LhsT, RhsT const&> const {
20365         static_assert(always_false<RhsT>::value,
20366             "operator&& is not supported inside assertions, "
20367             "wrap the expression inside parentheses, or decompose it");
20368     }
20369
20370     template<typename RhsT>
20371     auto operator || ( RhsT const& ) -> BinaryExpr<LhsT, RhsT const&> const {
20372         static_assert(always_false<RhsT>::value,
20373             "operator|| is not supported inside assertions, "
20374             "wrap the expression inside parentheses, or decompose it");
20375     }
20376
20377     auto makeUnaryExpr() const -> UnaryExpr<LhsT> {
20378         return UnaryExpr<LhsT>{ m_lhs };
20379     }
20380 };
20381
20382 void handleExpression( ITransientExpression const& expr );
20383
20384 template<typename T>
20385 void handleExpression( ExprLhs<T> const& expr ) {
20386     handleExpression( expr.makeUnaryExpr() );
20387 }
20388
20389 struct Decomposer {
20390     template<typename T>
20391     auto operator <= ( T const& lhs ) -> ExprLhs<T const&> {
20392         return ExprLhs<T const&>{ lhs };
20393     }
20394
20395     auto operator <=( bool value ) -> ExprLhs<bool> {
20396         return ExprLhs<bool>{ value };
20397     }
20398 };
20399
20400 } // end namespace Catch
20401
20402 #ifdef _MSC_VER
20403 #pragma warning(pop)
20404 #endif
20405
20406 // end catch_decomposer.h
20407 // start catch_interfaces_capture.h
20408
20409 #include <string>
20410 #include <chrono>
20411
20412 namespace Catch {
20413
20414     class AssertionResult;
20415     struct AssertionInfo;
20416     struct SectionInfo;
20417     struct SectionEndInfo;
20418     struct MessageInfo;
20419     struct MessageBuilder;
20420     struct Counts;
20421     struct AssertionReaction;
20422     struct SourceLineInfo;
20423
20424     struct ITransientExpression;
20425     struct IGeneratorTracker;
20426
20427 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
20428     struct BenchmarkInfo;
20429     template <typename Duration = std::chrono::duration<double, std::nano>
20430     struct BenchmarkStats;
20431 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
20432
20433     struct IResultCapture {
20434
20435         virtual ~IResultCapture();
20436
20437         virtual bool sectionStarted(      SectionInfo const& sectionInfo,
20438                                     Counts& assertions ) = 0;
20439         virtual void sectionEnded( SectionEndInfo const& endInfo ) = 0;
20440         virtual void sectionEndedEarly( SectionEndInfo const& endInfo ) = 0;
20441
20442         virtual auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo
20443 ) -> IGeneratorTracker& = 0;
20444
20445 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
20446         virtual void benchmarkPreparing( std::string const& name ) = 0;
20447         virtual void benchmarkStarting( BenchmarkInfo const& info ) = 0;
20448         virtual void benchmarkEnded( BenchmarkStats<> const& stats ) = 0;
20449         virtual void benchmarkFailed( std::string const& error ) = 0;
20450 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING

```

```

20450
20451     virtual void pushScopedMessage( MessageInfo const& message ) = 0;
20452     virtual void popScopedMessage( MessageInfo const& message ) = 0;
20453
20454     virtual void emplaceUnscopedMessage( MessageBuilder const& builder ) = 0;
20455
20456     virtual void handleFatalErrorCondition( StringRef message ) = 0;
20457
20458     virtual void handleExpr
20459         ( AssertionInfo const& info,
20460           ITransientExpression const& expr,
20461           AssertionReaction& reaction ) = 0;
20462     virtual void handleMessage
20463         ( AssertionInfo const& info,
20464           ResultWas::OfType resultType,
20465           StringRef const& message,
20466           AssertionReaction& reaction ) = 0;
20467     virtual void handleUnexpectedExceptionNotThrown
20468         ( AssertionInfo const& info,
20469           AssertionReaction& reaction ) = 0;
20470     virtual void handleUnexpectedInflightException
20471         ( AssertionInfo const& info,
20472           std::string const& message,
20473           AssertionReaction& reaction ) = 0;
20474     virtual void handleIncomplete
20475         ( AssertionInfo const& info ) = 0;
20476     virtual void handleNonExpr
20477         ( AssertionInfo const& info,
20478           ResultWas::OfType resultType,
20479           AssertionReaction& reaction ) = 0;
20480
20481     virtual bool lastAssertionPassed() = 0;
20482     virtual void assertionPassed() = 0;
20483
20484     // Deprecated, do not use:
20485     virtual std::string getCurrentTestName() const = 0;
20486     virtual const AssertionResult* getLastResult() const = 0;
20487     virtual void exceptionEarlyReported() = 0;
20488 };
20489
20490 IResultCapture& getResultCapture();
20491 }
20492
20493 // end catch_interfaces_capture.h
20494 namespace Catch {
20495
20496     struct TestFailureException{};
20497     struct AssertionResultData;
20498     struct IResultCapture;
20499     class RunContext;
20500
20501     class LazyExpression {
20502     friend class AssertionHandler;
20503     friend struct AssertionStats;
20504     friend class RunContext;
20505
20506     ITransientExpression const* m_transientExpression = nullptr;
20507     bool m_isNegated;
20508     public:
20509         LazyExpression( bool isNegated );
20510         LazyExpression( LazyExpression const& other );
20511         LazyExpression& operator = ( LazyExpression const& ) = delete;
20512
20513         explicit operator bool() const;
20514
20515         friend auto operator << ( std::ostream& os, LazyExpression const& lazyExpr ) -> std::ostream&;
20516     };
20517
20518     struct AssertionReaction {
20519         bool shouldDebugBreak = false;
20520         bool shouldThrow = false;
20521     };
20522
20523     class AssertionHandler {
20524     AssertionInfo m_assertionInfo;
20525     AssertionReaction m_reaction;
20526     bool m_completed = false;
20527     IResultCapture& m_resultCapture;
20528     public:
20529         AssertionHandler
20530             ( StringRef const& macroName,
20531               SourceLineInfo const& lineInfo,
20532               StringRef capturedExpression,
20533               ResultDisposition::Flags resultDisposition );
20534         ~AssertionHandler() {
20535             if ( !m_completed ) {

```

```

20537         m_resultCapture.handleIncomplete( m_assertionInfo );
20538     }
20539 }
20540
20541 template<typename T>
20542 void handleExpr( ExprLhs<T> const& expr ) {
20543     handleExpr( expr.makeUnaryExpr() );
20544 }
20545 void handleExpr( ITransientExpression const& expr );
20546
20547 void handleMessage(ResultWas::OfType resultType, StringRef const& message);
20548
20549 void handleExceptionThrownAsExpected();
20550 void handleUnexpectedExceptionNotThrown();
20551 void handleExceptionNotThrownAsExpected();
20552 void handleThrowingCallSkipped();
20553 void handleUnexpectedInflightException();
20554
20555 void complete();
20556 void setCompleted();
20557
20558 // query
20559 auto allowThrows() const -> bool;
20560 };
20561
20562 void handleExceptionMatchExpr( AssertionHandler& handler, std::string const& str, StringRef const&
matcherString );
20563
20564 } // namespace Catch
20565
20566 // end catch_assertionhandler.h
20567 // start catch_message.h
20568
20569 #include <string>
20570 #include <vector>
20571
20572 namespace Catch {
20573
20574     struct MessageInfo {
20575         MessageInfo( StringRef const& _macroName,
20576                     SourceLineInfo const& _lineInfo,
20577                     ResultWas::OfType _type );
20578
20579         StringRef macroName;
20580         std::string message;
20581         SourceLineInfo lineInfo;
20582         ResultWas::OfType type;
20583         unsigned int sequence;
20584
20585         bool operator == ( MessageInfo const& other ) const;
20586         bool operator < ( MessageInfo const& other ) const;
20587     private:
20588         static unsigned int globalCount;
20589     };
20590
20591     struct MessageStream {
20592
20593         template<typename T>
20594         MessageStream& operator « ( T const& value ) {
20595             m_stream « value;
20596             return *this;
20597         }
20598
20599         ReusableStringStream m_stream;
20600     };
20601
20602     struct MessageBuilder : MessageStream {
20603         MessageBuilder( StringRef const& macroName,
20604                       SourceLineInfo const& lineInfo,
20605                       ResultWas::OfType type );
20606
20607         template<typename T>
20608         MessageBuilder& operator « ( T const& value ) {
20609             m_stream « value;
20610             return *this;
20611         }
20612
20613         MessageInfo m_info;
20614     };
20615
20616     class ScopedMessage {
20617     public:
20618         explicit ScopedMessage( MessageBuilder const& builder );
20619         ScopedMessage( ScopedMessage& duplicate ) = delete;
20620         ScopedMessage( ScopedMessage&& old );
20621         ~ScopedMessage();
20622     };

```

```

20623     MessageInfo m_info;
20624     bool m_moved;
20625 };
20626
20627     class Capturer {
20628     public:
20629         std::vector<MessageInfo> m_messages;
20630         IResultCapture& m_resultCapture = getResultCapture();
20631         size_t m_captured = 0;
20632     public:
20633         Capturer( StringRef macroName, SourceLineInfo const& lineInfo, ResultWas::OfType resultType,
20634             StringRef names );
20635         ~Capturer();
20636
20637         void captureValue( size_t index, std::string const& value );
20638
20639         template<typename T>
20640         void captureValues( size_t index, T const& value ) {
20641             captureValue( index, Catch::Detail::stringify( value ) );
20642         }
20643
20644         template<typename T, typename... Ts>
20645         void captureValues( size_t index, T const& value, Ts const&... values ) {
20646             captureValue( index, Catch::Detail::stringify(value) );
20647             captureValues( index+1, values... );
20648         }
20649     };
20650
20651 } // end namespace Catch
20652
20653 // end catch_message.h
20654 #if !defined(CATCH_CONFIG_DISABLE)
20655 #if !defined(CATCH_CONFIG_DISABLE_STRINGIFICATION)
20656 #define CATCH_INTERNAL_STRINGIFY(...) #__VA_ARGS__
20657 #else
20658 #define CATCH_INTERNAL_STRINGIFY(...) "Disabled by CATCH_CONFIG_DISABLE_STRINGIFICATION"
20659 #endif
20660 #if defined(CATCH_CONFIG_FAST_COMPILE) || defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
20661 // Another way to speed-up compilation is to omit local try-catch for REQUIRE*
20662 // macros.
20663 #define INTERNAL_CATCH_TRY
20664 #define INTERNAL_CATCH_CATCH( handler )
20665 #else
20666 #define INTERNAL_CATCH_TRY try
20667 #define INTERNAL_CATCH_CATCH( handler ) catch(...) { handler.handleUnexpectedInflightException(); }
20668 #endif
20669 #define INTERNAL_CATCH_REACT( handler ) handler.complete();
20670
20671 #define INTERNAL_CATCH_TEST( macroName, resultDisposition, ... ) \
20672     do { \
20673         CATCH_INTERNAL_IGNORE_BUT_WARN(__VA_ARGS__); \
20674         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
20675             CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition ); \
20676         INTERNAL_CATCH_TRY { \
20677             CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
20678             CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS \
20679             catchAssertionHandler.handleExpr( Catch::Decomposer() <= __VA_ARGS__ ); \
20680             CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
20681         } \
20682         INTERNAL_CATCH_CATCH( catchAssertionHandler ) \
20683         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
20684     } while( (void)0, (false) && static_cast<bool>( !(__VA_ARGS__) ) )
20685
20686 #define INTERNAL_CATCH_IF( macroName, resultDisposition, ... ) \
20687     INTERNAL_CATCH_TEST( macroName, resultDisposition, __VA_ARGS__ ); \
20688     if( Catch::getResultCapture().lastAssertionPassed() )
20689
20690 #define INTERNAL_CATCH_ELSE( macroName, resultDisposition, ... ) \
20691     INTERNAL_CATCH_TEST( macroName, resultDisposition, __VA_ARGS__ ); \
20692     if( !Catch::getResultCapture().lastAssertionPassed() )
20693
20694 #define INTERNAL_CATCH_NO_THROW( macroName, resultDisposition, ... ) \
20695     do { \
20696         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
20697             CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition ); \
20698         try { \
20699             static_cast<void>(__VA_ARGS__); \
20700             catchAssertionHandler.handleExceptionNotThrownAsExpected(); \
20701         } \
20702         catch( ... ) { \
20703             catchAssertionHandler.handleUnexpectedInflightException(); \
20704         } \
20705     } while( false )

```

```

20712         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
20713     } while( false )
20714
20716 #define INTERNAL_CATCH_THROWS( macroName, resultDisposition, ... ) \
20717     do { \
20718         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition); \
20719         if( catchAssertionHandler.allowThrows() ) \
20720             try { \
20721                 static_cast<void>(__VA_ARGS__); \
20722                 catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
20723             } \
20724             catch( ... ) { \
20725                 catchAssertionHandler.handleExceptionThrownAsExpected(); \
20726             } \
20727         else \
20728             catchAssertionHandler.handleThrowingCallSkipped(); \
20729         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
20730     } while( false )
20731
20733 #define INTERNAL_CATCH_THROWS_AS( macroName, exceptionType, resultDisposition, expr ) \
20734     do { \
20735         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
CATCH_INTERNAL_STRINGIFY(expr), " CATCH_INTERNAL_STRINGIFY(exceptionType)", resultDisposition ); \
20736         if( catchAssertionHandler.allowThrows() ) \
20737             try { \
20738                 static_cast<void>(expr); \
20739                 catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
20740             } \
20741             catch( exceptionType const& ) { \
20742                 catchAssertionHandler.handleExceptionThrownAsExpected(); \
20743             } \
20744             catch( ... ) { \
20745                 catchAssertionHandler.handleUnexpectedInflightException(); \
20746             } \
20747         else \
20748             catchAssertionHandler.handleThrowingCallSkipped(); \
20749         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
20750     } while( false )
20751
20753 #define INTERNAL_CATCH_MSG( macroName, messageType, resultDisposition, ... ) \
20754     do { \
20755         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
Catch::StringRef(), resultDisposition ); \
20756         catchAssertionHandler.handleMessage( messageType, ( Catch::MessageStream() << __VA_ARGS__ +
::Catch::StreamEndStop() ).m_stream.str() ); \
20757         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
20758     } while( false )
20759
20761 #define INTERNAL_CATCH_CAPTURE( varName, macroName, ... ) \
20762     auto varName = Catch::Capturer( macroName, CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info,
__VA_ARGS__ ); \
20763     varName.captureValues( 0, __VA_ARGS__ )
20764
20766 #define INTERNAL_CATCH_INFO( macroName, log ) \
20767     Catch::ScopedMessage INTERNAL_CATCH_UNIQUE_NAME( scopedMessage )( Catch::MessageBuilder(
macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info ) << log );
20768
20770 #define INTERNAL_CATCH_UNSCOPED_INFO( macroName, log ) \
20771     Catch::getResultCapture().emplaceUnscopedMessage( Catch::MessageBuilder( macroName##_catch_sr,
CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info ) << log )
20772
20774 // Although this is matcher-based, it can be used with just a string
20775 #define INTERNAL_CATCH_THROWS_STR_MATCHES( macroName, resultDisposition, matcher, ... ) \
20776     do { \
20777         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), " CATCH_INTERNAL_STRINGIFY(matcher)", resultDisposition ); \
20778         if( catchAssertionHandler.allowThrows() ) \
20779             try { \
20780                 static_cast<void>(__VA_ARGS__); \
20781                 catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
20782             } \
20783             catch( ... ) { \
20784                 Catch::handleExceptionMatchExpr( catchAssertionHandler, matcher, #matcher##_catch_sr
); \
20785             } \
20786         else \
20787             catchAssertionHandler.handleThrowingCallSkipped(); \
20788         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
20789     } while( false )
20790
20791 #endif // CATCH_CONFIG_DISABLE
20792
20793 // end catch_capture.hpp
20794 // start catch_section.h
20795
20796 // start catch_section_info.h

```



```

20797
20798 // start catch_totals.h
20799
20800 #include <cstdint>
20801
20802 namespace Catch {
20803
20804     struct Counts {
20805         Counts operator - ( Counts const& other ) const;
20806         Counts& operator += ( Counts const& other );
20807
20808         std::size_t total() const;
20809         bool allPassed() const;
20810         bool allOk() const;
20811
20812         std::size_t passed = 0;
20813         std::size_t failed = 0;
20814         std::size_t failedButOk = 0;
20815     };
20816
20817     struct Totals {
20818
20819         Totals operator - ( Totals const& other ) const;
20820         Totals& operator += ( Totals const& other );
20821
20822         Totals delta( Totals const& prevTotals ) const;
20823
20824         int error = 0;
20825         Counts assertions;
20826         Counts testCases;
20827     };
20828 }
20829
20830 // end catch_totals.h
20831 #include <string>
20832
20833 namespace Catch {
20834
20835     struct SectionInfo {
20836         SectionInfo
20837             ( SourceLineInfo const& _lineInfo,
20838               std::string const& _name );
20839
20840         // Deprecated
20841         SectionInfo
20842             ( SourceLineInfo const& _lineInfo,
20843               std::string const& _name,
20844               std::string const& ) : SectionInfo( _lineInfo, _name ) {}
20845
20846         std::string name;
20847         std::string description; // !Deprecated: this will always be empty
20848         SourceLineInfo lineInfo;
20849     };
20850
20851     struct SectionEndInfo {
20852         SectionInfo sectionInfo;
20853         Counts prevAssertions;
20854         double durationInSeconds;
20855     };
20856
20857 } // end namespace Catch
20858
20859 // end catch_section_info.h
20860 // start catch_timer.h
20861
20862 #include <cstdint>
20863
20864 namespace Catch {
20865
20866     auto getCurrentNanosecondsSinceEpoch() -> uint64_t;
20867     auto getEstimatedClockResolution() -> uint64_t;
20868
20869     class Timer {
20870     public:
20871         uint64_t m_nanoseconds = 0;
20872         void start();
20873         auto getElapsedNanoseconds() const -> uint64_t;
20874         auto getElapsedMicroseconds() const -> uint64_t;
20875         auto getElapsedMilliseconds() const -> unsigned int;
20876         auto getElapsedSeconds() const -> double;
20877     };
20878
20879 } // namespace Catch
20880
20881 // end catch_timer.h
20882 #include <string>
20883

```

```

20884 namespace Catch {
20885
20886     class Section : NonCopyable {
20887     public:
20888         Section( SectionInfo const& info );
20889         ~Section();
20890
20891         // This indicates whether the section should be executed or not
20892         explicit operator bool() const;
20893
20894     private:
20895         SectionInfo m_info;
20896
20897         std::string m_name;
20898         Counts m_assertions;
20899         bool m_sectionIncluded;
20900         Timer m_timer;
20901     };
20902
20903 } // end namespace Catch
20904
20905 #define INTERNAL_CATCH_SECTION( ... ) \
20906     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
20907     CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
20908     if( Catch::Section const& INTERNAL_CATCH_UNIQUE_NAME( catch_internal_Section ) = \
20909         Catch::SectionInfo( CATCH_INTERNAL_LINEINFO, __VA_ARGS__ ) ) \
20910         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
20911
20912 #define INTERNAL_CATCH_DYNAMIC_SECTION( ... ) \
20913     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
20914     CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
20915     if( Catch::Section const& INTERNAL_CATCH_UNIQUE_NAME( catch_internal_Section ) = \
20916         Catch::SectionInfo( CATCH_INTERNAL_LINEINFO, (Catch::ReusableStringStream() << __VA_ARGS__).str() ) ) \
20917         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
20918
20919 // end catch_section.h
20920 // start catch_interfaces_exception.h
20921
20922 // start catch_interfaces_registry_hub.h
20923
20924 #include <string>
20925 #include <memory>
20926
20927 namespace Catch {
20928
20929     class TestCase;
20930     struct ITestCaseRegistry;
20931     struct IExceptionTranslatorRegistry;
20932     struct IExceptionTranslator;
20933     struct IReporterRegistry;
20934     struct IReporterFactory;
20935     struct ITagAliasRegistry;
20936     struct IMutableEnumValuesRegistry;
20937
20938     class StartupExceptionRegistry;
20939
20940     using IReporterFactoryPtr = std::shared_ptr<IReporterFactory>;
20941
20942     struct IRegistryHub {
20943     virtual ~IRegistryHub();
20944
20945     virtual IReporterRegistry const& getReporterRegistry() const = 0;
20946     virtual ITestCaseRegistry const& getTestCaseRegistry() const = 0;
20947     virtual ITagAliasRegistry const& getTagAliasRegistry() const = 0;
20948     virtual IExceptionTranslatorRegistry const& getExceptionTranslatorRegistry() const = 0;
20949     virtual StartupExceptionRegistry const& getStartupExceptionRegistry() const = 0;
20950
20951     struct IMutableRegistryHub {
20952     virtual ~IMutableRegistryHub();
20953     virtual void registerReporter( std::string const& name, IReporterFactoryPtr const& factory ) =
20954     0;
20955     virtual void registerListener( IReporterFactoryPtr const& factory ) = 0;
20956     virtual void registerTest( TestCase const& testInfo ) = 0;
20957     virtual void registerTranslator( const IExceptionTranslator* translator ) = 0;
20958     virtual void registerTagAlias( std::string const& alias, std::string const& tag, \
20959         SourceLineInfo const& lineInfo ) = 0;
20960     virtual void registerStartupException() noexcept = 0;
20961     virtual IMutableEnumValuesRegistry& getMutableEnumValuesRegistry() = 0;
20962     };
20963
20964     IRegistryHub const& getRegistryHub();
20965     IMutableRegistryHub& getMutableRegistryHub();
20966     void cleanUp();
20967     std::string translateActiveException();

```

```

20967 }
20968
20969 // end catch_interfaces_registry_hub.h
20970 #if defined(CATCH_CONFIG_DISABLE)
20971     #define INTERNAL_CATCH_TRANSLATE_EXCEPTION_NO_REG( translatorName, signature) \
20972         static std::string translatorName( signature )
20973 #endif
20974
20975 #include <exception>
20976 #include <string>
20977 #include <vector>
20978
20979 namespace Catch {
20980     using exceptionTranslateFunction = std::string(*)();
20981
20982     struct IExceptionTranslator;
20983     using ExceptionTranslators = std::vector<std::unique_ptr<IExceptionTranslator const>;
20984
20985     struct IExceptionTranslator {
20986         virtual ~IExceptionTranslator();
20987         virtual std::string translate( ExceptionTranslators::const_iterator it,
ExceptionTranslators::const_iterator itEnd ) const = 0;
20988     };
20989
20990     struct IExceptionTranslatorRegistry {
20991         virtual ~IExceptionTranslatorRegistry();
20992
20993         virtual std::string translateActiveException() const = 0;
20994     };
20995
20996     class ExceptionTranslatorRegistrar {
20997     public:
20998         template<typename T>
20999         class ExceptionTranslator : public IExceptionTranslator {
21000         public:
21001             ExceptionTranslator( std::string(*translateFunction)( T& ) )
21002             : m_translateFunction( translateFunction )
21003             {}
21004
21005             std::string translate( ExceptionTranslators::const_iterator it,
ExceptionTranslators::const_iterator itEnd ) const override {
21006                 #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
21007                     return "";
21008                 #else
21009                     try {
21010                         if( it == itEnd )
21011                             std::rethrow_exception(std::current_exception());
21012                         else
21013                             return (*it)->translate( it+1, itEnd );
21014                     }
21015                     catch( T& ex ) {
21016                         return m_translateFunction( ex );
21017                     }
21018                 #endif
21019             }
21020
21021         protected:
21022             std::string(*m_translateFunction)( T& );
21023         };
21024
21025     public:
21026         template<typename T>
21027         ExceptionTranslatorRegistrar( std::string(*translateFunction)( T& ) ) {
21028             getMutableRegistryHub().registerTranslator
21029                 ( new ExceptionTranslator<T>( translateFunction ) );
21030         }
21031     };
21032 }
21033
21034 #define INTERNAL_CATCH_TRANSLATE_EXCEPTION2( translatorName, signature ) \
21035     static std::string translatorName( signature ); \
21036     CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
21037     CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
21038     namespace{ Catch::ExceptionTranslatorRegistrar INTERNAL_CATCH_UNIQUE_NAME( \
catch_internal_ExceptionTranslator )( &translatorName ); } \
21039     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
21040     static std::string translatorName( signature )
21041
21042 #define INTERNAL_CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION2( \
INTERNAL_CATCH_UNIQUE_NAME( catch_internal_ExceptionTranslator ), signature )
21043
21044 // end catch_interfaces_exception.h
21045 // start catch_approx.h
21046
21047 #include <type_traits>
21048
21049 namespace Catch {
21050     namespace Detail {

```

```

21051 namespace Detail {
21052
21053     class Approx {
21054     private:
21055         bool equalityComparisonImpl(double other) const;
21056         // Validates the new margin (margin >= 0)
21057         // out-of-line to avoid including stdexcept in the header
21058         void setMargin(double margin);
21059         // Validates the new epsilon (0 < epsilon < 1)
21060         // out-of-line to avoid including stdexcept in the header
21061         void setEpsilon(double epsilon);
21062
21063     public:
21064         explicit Approx ( double value );
21065
21066         static Approx custom();
21067
21068         Approx operator-() const;
21069
21070         template <typename T, typename = typename std::enable_if<std::is_constructible<double,
T::value>::type>
21071             Approx operator()( T const& value ) const {
21072                 Approx approx( static_cast<double>(value) );
21073                 approx.m_epsilon = m_epsilon;
21074                 approx.m_margin = m_margin;
21075                 approx.m_scale = m_scale;
21076                 return approx;
21077             }
21078
21079         template <typename T, typename = typename std::enable_if<std::is_constructible<double,
T::value>::type>
21080             explicit Approx( T const& value ) : Approx(static_cast<double>(value))
21081             {}
21082
21083         template <typename T, typename = typename std::enable_if<std::is_constructible<double,
T::value>::type>
21084             friend bool operator == ( const T& lhs, Approx const& rhs ) {
21085                 auto lhs_v = static_cast<double>(lhs);
21086                 return rhs.equalityComparisonImpl(lhs_v);
21087             }
21088
21089         template <typename T, typename = typename std::enable_if<std::is_constructible<double,
T::value>::type>
21090             friend bool operator == ( Approx const& lhs, const T& rhs ) {
21091                 return operator==( rhs, lhs );
21092             }
21093
21094         template <typename T, typename = typename std::enable_if<std::is_constructible<double,
T::value>::type>
21095             friend bool operator != ( T const& lhs, Approx const& rhs ) {
21096                 return !operator==( lhs, rhs );
21097             }
21098
21099         template <typename T, typename = typename std::enable_if<std::is_constructible<double,
T::value>::type>
21100             friend bool operator != ( Approx const& lhs, T const& rhs ) {
21101                 return !operator==( rhs, lhs );
21102             }
21103
21104         template <typename T, typename = typename std::enable_if<std::is_constructible<double,
T::value>::type>
21105             friend bool operator <= ( T const& lhs, Approx const& rhs ) {
21106                 return static_cast<double>(lhs) < rhs.m_value || lhs == rhs;
21107             }
21108
21109         template <typename T, typename = typename std::enable_if<std::is_constructible<double,
T::value>::type>
21110             friend bool operator <= ( Approx const& lhs, T const& rhs ) {
21111                 return lhs.m_value < static_cast<double>(rhs) || lhs == rhs;
21112             }
21113
21114         template <typename T, typename = typename std::enable_if<std::is_constructible<double,
T::value>::type>
21115             friend bool operator >= ( T const& lhs, Approx const& rhs ) {
21116                 return static_cast<double>(lhs) > rhs.m_value || lhs == rhs;
21117             }
21118
21119         template <typename T, typename = typename std::enable_if<std::is_constructible<double,
T::value>::type>
21120             friend bool operator >= ( Approx const& lhs, T const& rhs ) {
21121                 return lhs.m_value > static_cast<double>(rhs) || lhs == rhs;
21122             }
21123
21124         template <typename T, typename = typename std::enable_if<std::is_constructible<double,
T::value>::type>
21125             Approx& epsilon( T const& newEpsilon ) {
21126                 double epsilonAsDouble = static_cast<double>(newEpsilon);

```

```

21127         setEpsilon(epsilonAsDouble);
21128         return *this;
21129     }
21130
21131     template <typename T, typename = typename std::enable_if<std::is_constructible<double,
T>::value>::type>
21132     Approx& margin( T const& newMargin ) {
21133         double marginAsDouble = static_cast<double>(newMargin);
21134         setMargin(marginAsDouble);
21135         return *this;
21136     }
21137
21138     template <typename T, typename = typename std::enable_if<std::is_constructible<double,
T>::value>::type>
21139     Approx& scale( T const& newScale ) {
21140         m_scale = static_cast<double>(newScale);
21141         return *this;
21142     }
21143
21144     std::string toString() const;
21145
21146     private:
21147         double m_epsilon;
21148         double m_margin;
21149         double m_scale;
21150         double m_value;
21151     };
21152 } // end namespace Detail
21153
21154 namespace literals {
21155     Detail::Approx operator "" _a(long double val);
21156     Detail::Approx operator "" _a(unsigned long long val);
21157 } // end namespace literals
21158
21159 template<>
21160 struct StringMaker<Catch::Detail::Approx> {
21161     static std::string convert(Catch::Detail::Approx const& value);
21162 };
21163
21164 } // end namespace Catch
21165
21166 // end catch_approx.h
21167 // start catch_string_manip.h
21168
21169 #include <string>
21170 #include <iosfwd>
21171 #include <vector>
21172
21173 namespace Catch {
21174
21175     bool startsWith( std::string const& s, std::string const& prefix );
21176     bool startsWith( std::string const& s, char prefix );
21177     bool endsWith( std::string const& s, std::string const& suffix );
21178     bool endsWith( std::string const& s, char suffix );
21179     bool contains( std::string const& s, std::string const& infix );
21180     void toLowerInPlace( std::string& s );
21181     std::string toLower( std::string const& s );
21182     std::string trim( std::string const& str );
21183     StringRef trim(StringRef ref);
21184
21185     // !!! Be aware, returns refs into original string - make sure original string outlives them
21186     std::vector<StringRef> splitStringRef( StringRef str, char delimiter );
21187     bool replaceInPlace( std::string& str, std::string const& replaceThis, std::string const& withThis );
21188
21189 };
21190
21191 struct pluralise {
21192     pluralise( std::size_t count, std::string const& label );
21193
21194     friend std::ostream& operator << ( std::ostream& os, pluralise const& pluraliser );
21195
21196     std::size_t m_count;
21197     std::string m_label;
21198 };
21199 }
21200
21201 // end catch_string_manip.h
21202 #ifndef CATCH_CONFIG_DISABLE_MATCHERS
21203 // start catch_capture_matchers.h
21204
21205 // start catch_matchers.h
21206
21207 #include <string>
21208 #include <vector>
21209
21210 namespace Catch {
21211     namespace Matchers {
21212         namespace Impl {

```

```

21213
21214     template<typename ArgT> struct MatchAllOf;
21215     template<typename ArgT> struct MatchAnyOf;
21216     template<typename ArgT> struct MatchNotOf;
21217
21218     class MatcherUntypedBase {
21219     public:
21220         MatcherUntypedBase() = default;
21221         MatcherUntypedBase ( MatcherUntypedBase const& ) = default;
21222         MatcherUntypedBase& operator = ( MatcherUntypedBase const& ) = delete;
21223         std::string toString() const;
21224
21225     protected:
21226         virtual ~MatcherUntypedBase();
21227         virtual std::string describe() const = 0;
21228         mutable std::string m_cachedToString;
21229     };
21230
21231 #ifdef __clang__
21232 #     pragma clang diagnostic push
21233 #     pragma clang diagnostic ignored "-Wnon-virtual-dtor"
21234 #endif
21235
21236     template<typename ObjectT>
21237     struct MatcherMethod {
21238         virtual bool match( ObjectT const& arg ) const = 0;
21239     };
21240
21241 #if defined(__OBJC__)
21242     // Hack to fix Catch GH issue #1661. Could use id for generic Object support.
21243     // use of const for Object pointers is very uncommon and under ARC it causes some kind of
21244     // signature mismatch that breaks compilation
21245     template<>
21246     struct MatcherMethod<NSString*> {
21247         virtual bool match( NSString* arg ) const = 0;
21248     };
21249 #endif
21250 #ifdef __clang__
21251 #     pragma clang diagnostic pop
21252 #endif
21253
21254     template<typename T>
21255     struct MatcherBase : MatcherUntypedBase, MatcherMethod<T> {
21256
21257         MatchAllOf<T> operator && ( MatcherBase const& other ) const;
21258         MatchAnyOf<T> operator || ( MatcherBase const& other ) const;
21259         MatchNotOf<T> operator ! ( ) const;
21260     };
21261
21262     template<typename ArgT>
21263     struct MatchAllOf : MatcherBase<ArgT> {
21264         bool match( ArgT const& arg ) const override {
21265             for( auto matcher : m_matchers ) {
21266                 if (!matcher->match(arg))
21267                     return false;
21268             }
21269             return true;
21270         }
21271         std::string describe() const override {
21272             std::string description;
21273             description.reserve( 4 + m_matchers.size()*32 );
21274             description += "( ";
21275             bool first = true;
21276             for( auto matcher : m_matchers ) {
21277                 if( first )
21278                     first = false;
21279                 else
21280                     description += " and ";
21281                 description += matcher->toString();
21282             }
21283             description += " )";
21284             return description;
21285         }
21286
21287         MatchAllOf<ArgT> operator && ( MatcherBase<ArgT> const& other ) {
21288             auto copy(*this);
21289             copy.m_matchers.push_back( &other );
21290             return copy;
21291         }
21292
21293         std::vector<MatcherBase<ArgT> const*> m_matchers;
21294     };
21295     template<typename ArgT>
21296     struct MatchAnyOf : MatcherBase<ArgT> {
21297
21298         bool match( ArgT const& arg ) const override {

```

```

21299         for( auto matcher : m_matchers ) {
21300             if (matcher->match(arg))
21301                 return true;
21302         }
21303         return false;
21304     }
21305     std::string describe() const override {
21306         std::string description;
21307         description.reserve( 4 + m_matchers.size()*32 );
21308         description += "( ";
21309         bool first = true;
21310         for( auto matcher : m_matchers ) {
21311             if( first )
21312                 first = false;
21313             else
21314                 description += " or ";
21315             description += matcher->toString();
21316         }
21317         description += " )";
21318         return description;
21319     }
21320
21321     MatchAnyOf<ArgT> operator || ( MatcherBase<ArgT> const& other ) {
21322         auto copy(*this);
21323         copy.m_matchers.push_back( &other );
21324         return copy;
21325     }
21326
21327     std::vector<MatcherBase<ArgT> const*> m_matchers;
21328 };
21329
21330 template<typename ArgT>
21331 struct MatchNotOf : MatcherBase<ArgT> {
21332
21333     MatchNotOf( MatcherBase<ArgT> const& underlyingMatcher ) : m_underlyingMatcher(
underlyingMatcher ) {}
21334
21335     bool match( ArgT const& arg ) const override {
21336         return !m_underlyingMatcher.match( arg );
21337     }
21338
21339     std::string describe() const override {
21340         return "not " + m_underlyingMatcher.toString();
21341     }
21342     MatcherBase<ArgT> const& m_underlyingMatcher;
21343 };
21344
21345 template<typename T>
21346 MatchAllOf<T> MatcherBase<T>::operator && ( MatcherBase const& other ) const {
21347     return MatchAllOf<T>() && *this && other;
21348 }
21349 template<typename T>
21350 MatchAnyOf<T> MatcherBase<T>::operator || ( MatcherBase const& other ) const {
21351     return MatchAnyOf<T>() || *this || other;
21352 }
21353 template<typename T>
21354 MatchNotOf<T> MatcherBase<T>::operator ! () const {
21355     return MatchNotOf<T>( *this );
21356 }
21357
21358 } // namespace Impl
21359
21360 } // namespace Matchers
21361
21362 using namespace Matchers;
21363 using Matchers::Impl::MatcherBase;
21364
21365 } // namespace Catch
21366
21367 // end catch_matchers.h
21368 // start catch_matchers_exception.hpp
21369
21370 namespace Catch {
21371     namespace Matchers {
21372         namespace Exception {
21373
21374             class ExceptionMessageMatcher : public MatcherBase<std::exception> {
21375             public:
21376                 std::string m_message;
21377
21378                 ExceptionMessageMatcher( std::string const& message ) :
21379                     m_message( message )
21380                 {}
21381
21382                 bool match( std::exception const& ex ) const override;
21383
21384                 std::string describe() const override;

```

```

21385 };
21386
21387 } // namespace Exception
21388
21389 Exception::ExceptionMessageMatcher Message(std::string const& message);
21390
21391 } // namespace Matchers
21392 } // namespace Catch
21393
21394 // end catch_matchers_exception.hpp
21395 // start catch_matchers_floating.h
21396
21397 namespace Catch {
21398 namespace Matchers {
21399
21400     namespace Floating {
21401
21402         enum class FloatingPointKind : uint8_t;
21403
21404         struct WithinAbsMatcher : MatcherBase<double> {
21405             WithinAbsMatcher(double target, double margin);
21406             bool match(double const& matchee) const override;
21407             std::string describe() const override;
21408         private:
21409             double m_target;
21410             double m_margin;
21411         };
21412
21413         struct WithinUlpMatcher : MatcherBase<double> {
21414             WithinUlpMatcher(double target, uint64_t ulps, FloatingPointKind baseType);
21415             bool match(double const& matchee) const override;
21416             std::string describe() const override;
21417         private:
21418             double m_target;
21419             uint64_t m_ulps;
21420             FloatingPointKind m_type;
21421         };
21422
21423         // Given IEEE-754 format for floats and doubles, we can assume
21424         // that float -> double promotion is lossless. Given this, we can
21425         // assume that if we do the standard relative comparison of
21426         // |lhs - rhs| <= epsilon * max(fabs(lhs), fabs(rhs)), then we get
21427         // the same result if we do this for floats, as if we do this for
21428         // doubles that were promoted from floats.
21429         struct WithinRelMatcher : MatcherBase<double> {
21430             WithinRelMatcher(double target, double epsilon);
21431             bool match(double const& matchee) const override;
21432             std::string describe() const override;
21433         private:
21434             double m_target;
21435             double m_epsilon;
21436         };
21437
21438     } // namespace Floating
21439
21440     // The following functions create the actual matcher objects.
21441     // This allows the types to be inferred
21442     Floating::WithinUlpMatcher WithinULP(double target, uint64_t maxUlpDiff);
21443     Floating::WithinUlpMatcher WithinULP(float target, uint64_t maxUlpDiff);
21444     Floating::WithinAbsMatcher WithinAbs(double target, double margin);
21445     Floating::WithinRelMatcher WithinRel(double target, double eps);
21446     // defaults epsilon to 100*numeric_limits<double>::epsilon()
21447     Floating::WithinRelMatcher WithinRel(double target);
21448     Floating::WithinRelMatcher WithinRel(float target, float eps);
21449     // defaults epsilon to 100*numeric_limits<float>::epsilon()
21450     Floating::WithinRelMatcher WithinRel(float target);
21451
21452 } // namespace Matchers
21453 } // namespace Catch
21454
21455 // end catch_matchers_floating.h
21456 // start catch_matchers_generic.hpp
21457
21458 #include <functional>
21459 #include <string>
21460
21461 namespace Catch {
21462 namespace Matchers {
21463 namespace Generic {
21464
21465     namespace Detail {
21466         std::string finalizeDescription(const std::string& desc);
21467     }
21468
21469     template <typename T>
21470     class PredicateMatcher : public MatcherBase<T> {
21471     public:
21472         PredicateMatcher(T const& m_predicate);

```



```

21472     std::string m_description;
21473 public:
21474
21475     PredicateMatcher(std::function<bool(T const&)> const& elem, std::string const& descr)
21476         :m_predicate(std::move(elem)),
21477         m_description(Detail::finalizeDescription(descr))
21478     {}
21479
21480     bool match( T const& item ) const override {
21481         return m_predicate(item);
21482     }
21483
21484     std::string describe() const override {
21485         return m_description;
21486     }
21487 };
21488
21489 } // namespace Generic
21490
21491 // The following functions create the actual matcher objects.
21492 // The user has to explicitly specify type to the function, because
21493 // inferring std::function<bool(T const&)> is hard (but possible) and
21494 // requires a lot of TMP.
21495 template<typename T>
21496 Generic::PredicateMatcher<T> Predicate(std::function<bool(T const&)> const& predicate, std::string
const& description = "") {
21497     return Generic::PredicateMatcher<T>(predicate, description);
21498 }
21499
21500 } // namespace Matchers
21501 } // namespace Catch
21502
21503 // end catch_matchers_generic.hpp
21504 // start catch_matchers_string.h
21505
21506 #include <string>
21507
21508 namespace Catch {
21509 namespace Matchers {
21510     namespace StdString {
21511
21512         struct CasedString
21513         {
21514             CasedString( std::string const& str, CaseSensitive::Choice caseSensitivity );
21515             std::string adjustString( std::string const& str ) const;
21516             std::string caseSensitivitySuffix() const;
21517
21518             CaseSensitive::Choice m_caseSensitivity;
21519             std::string m_str;
21520         };
21521
21522         struct StringMatcherBase : MatcherBase<std::string> {
21523             StringMatcherBase( std::string const& operation, CasedString const& comparator );
21524             std::string describe() const override;
21525
21526             CasedString m_comparator;
21527             std::string m_operation;
21528         };
21529
21530         struct EqualsMatcher : StringMatcherBase {
21531             EqualsMatcher( CasedString const& comparator );
21532             bool match( std::string const& source ) const override;
21533         };
21534
21535         struct ContainsMatcher : StringMatcherBase {
21536             ContainsMatcher( CasedString const& comparator );
21537             bool match( std::string const& source ) const override;
21538         };
21539
21540         struct StartsWithMatcher : StringMatcherBase {
21541             StartsWithMatcher( CasedString const& comparator );
21542             bool match( std::string const& source ) const override;
21543         };
21544
21545         struct EndsWithMatcher : StringMatcherBase {
21546             EndsWithMatcher( CasedString const& comparator );
21547             bool match( std::string const& source ) const override;
21548         };
21549
21550         struct RegexMatcher : MatcherBase<std::string> {
21551             RegexMatcher( std::string regex, CaseSensitive::Choice caseSensitivity );
21552             bool match( std::string const& matchee ) const override;
21553             std::string describe() const override;
21554
21555         private:
21556             std::string m_regex;
21557             CaseSensitive::Choice m_caseSensitivity;
21558         };
21559     }
21560 }

```

```

21558     } // namespace StdString
21559
21560     // The following functions create the actual matcher objects.
21561     // This allows the types to be inferred
21562
21563     StdString::EqualsMatcher Equals( std::string const& str, CaseSensitive::Choice caseSensitivity =
CaseSensitive::Yes );
21564     StdString::ContainsMatcher Contains( std::string const& str, CaseSensitive::Choice caseSensitivity
= CaseSensitive::Yes );
21565     StdString::EndsWithMatcher EndsWith( std::string const& str, CaseSensitive::Choice caseSensitivity
= CaseSensitive::Yes );
21566     StdString::StartsWithMatcher StartsWith( std::string const& str, CaseSensitive::Choice
caseSensitivity = CaseSensitive::Yes );
21567     StdString::RegexMatcher Matches( std::string const& regex, CaseSensitive::Choice caseSensitivity =
CaseSensitive::Yes );
21568
21569 } // namespace Matchers
21570 } // namespace Catch
21571
21572 // end catch_matchers_string.h
21573 // start catch_matchers_vector.h
21574
21575 #include <algorithm>
21576
21577 namespace Catch {
21578     namespace Matchers {
21579
21580         namespace Vector {
21581             template<typename T, typename Alloc>
21582             struct ContainsElementMatcher : MatcherBase<std::vector<T, Alloc> {
21583
21584                 ContainsElementMatcher(T const& comparator) : m_comparator( comparator ) {}
21585
21586                 bool match(std::vector<T, Alloc> const& v) const override {
21587                     for (auto const& el : v) {
21588                         if (el == m_comparator) {
21589                             return true;
21590                         }
21591                     }
21592                     return false;
21593                 }
21594
21595                 std::string describe() const override {
21596                     return "Contains: " + ::Catch::Detail::stringify( m_comparator );
21597                 }
21598
21599                 T const& m_comparator;
21600             };
21601
21602             template<typename T, typename AllocComp, typename AllocMatch>
21603             struct ContainsMatcher : MatcherBase<std::vector<T, AllocMatch> {
21604
21605                 ContainsMatcher(std::vector<T, AllocComp> const& comparator) : m_comparator( comparator ) {}
21606
21607                 bool match(std::vector<T, AllocMatch> const& v) const override {
21608                     // !TBD: see note in EqualsMatcher
21609                     if (m_comparator.size() > v.size())
21610                         return false;
21611                     for (auto const& comparator : m_comparator) {
21612                         auto present = false;
21613                         for (const auto& el : v) {
21614                             if (el == comparator) {
21615                                 present = true;
21616                                 break;
21617                             }
21618                         }
21619                         if (!present) {
21620                             return false;
21621                         }
21622                     }
21623                     return true;
21624                 }
21625
21626                 std::string describe() const override {
21627                     return "Contains: " + ::Catch::Detail::stringify( m_comparator );
21628                 }
21629
21630                 std::vector<T, AllocComp> const& m_comparator;
21631             };
21632
21633             template<typename T, typename AllocComp, typename AllocMatch>
21634             struct EqualsMatcher : MatcherBase<std::vector<T, AllocMatch> {
21635
21636                 EqualsMatcher(std::vector<T, AllocComp> const& comparator) : m_comparator( comparator ) {}
21637
21638                 bool match(std::vector<T, AllocMatch> const& v) const override {
21639                     // !TBD: This currently works if all elements can be compared using !=

```

```

21639         // - a more general approach would be via a compare template that defaults
21640         // to using !=. but could be specialised for, e.g. std::vector<T, Alloc> etc
21641         // - then just call that directly
21642         if (m_comparator.size() != v.size())
21643             return false;
21644         for (std::size_t i = 0; i < v.size(); ++i)
21645             if (m_comparator[i] != v[i])
21646                 return false;
21647         return true;
21648     }
21649     std::string describe() const override {
21650         return "Equals: " + ::Catch::Detail::stringify( m_comparator );
21651     }
21652     std::vector<T, AllocComp> const& m_comparator;
21653 };
21654
21655 template<typename T, typename AllocComp, typename AllocMatch>
21656 struct ApproxMatcher : MatcherBase<std::vector<T, AllocMatch> {
21657     ApproxMatcher(std::vector<T, AllocComp> const& comparator) : m_comparator( comparator ) {}
21658
21659     bool match(std::vector<T, AllocMatch> const& v) const override {
21660         if (m_comparator.size() != v.size())
21661             return false;
21662         for (std::size_t i = 0; i < v.size(); ++i)
21663             if (m_comparator[i] != approx(v[i]))
21664                 return false;
21665         return true;
21666     }
21667     std::string describe() const override {
21668         return "is approx: " + ::Catch::Detail::stringify( m_comparator );
21669     }
21670     template <typename = typename std::enable_if<std::is_constructible<double,
21671 T>::value>::type>
21672     ApproxMatcher& epsilon( T const& newEpsilon ) {
21673         approx.epsilon(newEpsilon);
21674         return *this;
21675     }
21676     template <typename = typename std::enable_if<std::is_constructible<double,
21677 T>::value>::type>
21678     ApproxMatcher& margin( T const& newMargin ) {
21679         approx.margin(newMargin);
21680         return *this;
21681     }
21682     template <typename = typename std::enable_if<std::is_constructible<double,
21683 T>::value>::type>
21684     ApproxMatcher& scale( T const& newScale ) {
21685         approx.scale(newScale);
21686         return *this;
21687     }
21688     std::vector<T, AllocComp> const& m_comparator;
21689     mutable Catch::Detail::Approx approx = Catch::Detail::Approx::custom();
21690 };
21691
21692 template<typename T, typename AllocComp, typename AllocMatch>
21693 struct UnorderedEqualsMatcher : MatcherBase<std::vector<T, AllocMatch> {
21694     UnorderedEqualsMatcher(std::vector<T, AllocComp> const& target) : m_target(target) {}
21695     bool match(std::vector<T, AllocMatch> const& vec) const override {
21696         if (m_target.size() != vec.size()) {
21697             return false;
21698         }
21699         return std::is_permutation(m_target.begin(), m_target.end(), vec.begin());
21700     }
21701     std::string describe() const override {
21702         return "UnorderedEquals: " + ::Catch::Detail::stringify(m_target);
21703     }
21704 private:
21705     std::vector<T, AllocComp> const& m_target;
21706 };
21707
21708 } // namespace Vector
21709
21710 // The following functions create the actual matcher objects.
21711 // This allows the types to be inferred
21712
21713 template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
21714 Vector::ContainsMatcher<T, AllocComp, AllocMatch> Contains( std::vector<T, AllocComp> const&
comparator ) {
21715     return Vector::ContainsMatcher<T, AllocComp, AllocMatch>( comparator );
21716 }
21717
21718 template<typename T, typename Alloc = std::allocator<T>
Vector::ContainsElementMatcher<T, Alloc> VectorContains( T const& comparator ) {
21719     return Vector::ContainsElementMatcher<T, Alloc>( comparator );
21720 }
21721

```

```

21722
21723     template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
21724     Vector::EqualsMatcher<T, AllocComp, AllocMatch> Equals( std::vector<T, AllocComp> const&
comparator ) {
21725         return Vector::EqualsMatcher<T, AllocComp, AllocMatch>( comparator );
21726     }
21727
21728     template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
21729     Vector::ApproxMatcher<T, AllocComp, AllocMatch> Approx( std::vector<T, AllocComp> const&
comparator ) {
21730         return Vector::ApproxMatcher<T, AllocComp, AllocMatch>( comparator );
21731     }
21732
21733     template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
21734     Vector::UnorderedEqualsMatcher<T, AllocComp, AllocMatch> UnorderedEquals( std::vector<T, AllocComp>
const& target ) {
21735         return Vector::UnorderedEqualsMatcher<T, AllocComp, AllocMatch>( target );
21736     }
21737
21738 } // namespace Matchers
21739 } // namespace Catch
21740
21741 // end catch_matchers_vector.h
21742 namespace Catch {
21743
21744     template<typename ArgT, typename MatcherT>
21745     class MatchExpr : public ITransientExpression {
21746     public:
21747         ArgT const& m_arg;
21748         MatcherT m_matcher;
21749         StringRef m_matcherString;
21750
21751         MatchExpr( ArgT const& arg, MatcherT const& matcher, StringRef const& matcherString )
21752         : ITransientExpression( true, matcher.match( arg ) ),
21753           m_arg( arg ),
21754           m_matcher( matcher ),
21755           m_matcherString( matcherString )
21756         {}
21757
21758         void streamReconstructedExpression( std::ostream &os ) const override {
21759             auto matcherAsString = m_matcher.toString();
21760             os << "Catch::Detail::stringify( m_arg ) << ' '";
21761             if( matcherAsString == Detail::unprintableString )
21762                 os << m_matcherString;
21763             else
21764                 os << matcherAsString;
21765         }
21766
21767         using StringMatcher = Matchers::Impl::MatcherBase<std::string>;
21768
21769         void handleExceptionMatchExpr( AssertionHandler& handler, StringMatcher const& matcher, StringRef
const& matcherString );
21770
21771     template<typename ArgT, typename MatcherT>
21772     auto makeMatchExpr( ArgT const& arg, MatcherT const& matcher, StringRef const& matcherString ) ->
MatchExpr<ArgT, MatcherT> {
21773         return MatchExpr<ArgT, MatcherT>( arg, matcher, matcherString );
21774     }
21775
21776 } // namespace Catch
21777
21778 #define INTERNAL_CHECK_THAT( macroName, matcher, resultDisposition, arg ) \
21779     do { \
21780         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
CATCH_INTERNAL_STRINGIFY(arg) ", " CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
21781         INTERNAL_CATCH_TRY { \
21782             catchAssertionHandler.handleExpr( Catch::makeMatchExpr( arg, matcher, #matcher##_catch_sr \
) ); \
21783         } \
21784         INTERNAL_CATCH_CATCH( catchAssertionHandler ) \
21785         INTERNAL_CATCH_REACT( catchAssertionHandler ) \
21786     } while( false )
21787
21788 #define INTERNAL_CATCH_THROWS_MATCHES( macroName, exceptionType, resultDisposition, matcher, ... ) \
21789     do { \
21790         Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, \
CATCH_INTERNAL_STRINGIFY(__VA_ARGS__) ", " CATCH_INTERNAL_STRINGIFY(exceptionType) ", " \
CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
21791         if( catchAssertionHandler.allowThrows() ) \
21792             try { \
21793                 static_cast<void>( __VA_ARGS__ ); \
21794                 catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
21795             } \
21796             catch( exceptionType const& ex ) { \
21797                 catchAssertionHandler.handleExpr( Catch::makeMatchExpr( ex, matcher, \
#matcher##_catch_sr ) ); \
21798             } \
21799         catch( ... ) { \
21800

```

```

21801         catchAssertionHandler.handleUnexpectedInflightException(); \
21802     } \
21803     else \
21804         catchAssertionHandler.handleThrowingCallSkipped(); \
21805     INTERNAL_CATCH_REACT( catchAssertionHandler ) \
21806 } while( false )
21807
21808 // end catch_capture_matchers.h
21809 #endif
21810 // start catch_generators.hpp
21811
21812 // start catch_interfaces_generatortracker.h
21813
21814 #include <memory>
21815 namespace Catch {
21816     namespace Generators {
21817         class GeneratorUntypedBase {
21818         public:
21819             GeneratorUntypedBase() = default;
21820             virtual ~GeneratorUntypedBase();
21821             // Attempts to move the generator to the next element
21822             //
21823             // Returns true iff the move succeeded (and a valid element
21824             // can be retrieved).
21825             virtual bool next() = 0;
21826         };
21827         using GeneratorBasePtr = std::unique_ptr<GeneratorUntypedBase>;
21828     } // namespace Generators
21829
21830     struct IGeneratorTracker {
21831         virtual ~IGeneratorTracker();
21832         virtual auto hasGenerator() const -> bool = 0;
21833         virtual auto getGenerator() const -> Generators::GeneratorBasePtr const& = 0;
21834         virtual void setGenerator( Generators::GeneratorBasePtr&& generator ) = 0;
21835     };
21836 } // namespace Catch
21837
21838 // end catch_interfaces_generatortracker.h
21839 // start catch_enforce.h
21840
21841 #include <exception>
21842 namespace Catch {
21843     template <typename Ex>
21844     [[noreturn]]
21845     void throw_exception(Ex const& e) {
21846         throw e;
21847     }
21848 #else // ^^ Exceptions are enabled // Exceptions are disabled vv
21849     [[noreturn]]
21850     void throw_exception(std::exception const& e);
21851 #endif
21852
21853     [[noreturn]]
21854     void throw_logic_error(std::string const& msg);
21855     [[noreturn]]
21856     void throw_domain_error(std::string const& msg);
21857     [[noreturn]]
21858     void throw_runtime_error(std::string const& msg);
21859 } // namespace Catch;
21860
21861 #define CATCH_MAKE_MSG(...) \
21862     (Catch::ReusableStringStream() << __VA_ARGS__).str()
21863
21864 #define CATCH_INTERNAL_ERROR(...) \
21865     Catch::throw_logic_error(CATCH_MAKE_MSG( CATCH_INTERNAL_LINEINFO << " Internal Catch2 error: " << __VA_ARGS__))
21866
21867 #define CATCH_ERROR(...) \
21868     Catch::throw_domain_error(CATCH_MAKE_MSG( __VA_ARGS__ ))
21869
21870 #define CATCH_RUNTIME_ERROR(...) \
21871     Catch::throw_runtime_error(CATCH_MAKE_MSG( __VA_ARGS__ ))
21872
21873 #define CATCH_ENFORCE( condition, ... ) \
21874     do{ if( !(condition) ) CATCH_ERROR( __VA_ARGS__ ); } while(false)
21875
21876 // end catch_enforce.h
21877 #include <memory>
21878 #include <vector>

```

```

21887 #include <cassert>
21888
21889 #include <utility>
21890 #include <exception>
21891
21892 namespace Catch {
21893
21894 class GeneratorException : public std::exception {
21895     const char* const m_msg = "";
21896
21897 public:
21898     GeneratorException(const char* msg):
21899         m_msg(msg)
21900     {}
21901
21902     const char* what() const noexcept override final;
21903 };
21904
21905 namespace Generators {
21906
21907     // !TBD move this into its own location?
21908     namespace pf{
21909         template<typename T, typename... Args>
21910         std::unique_ptr<T> make_unique( Args&&... args ) {
21911             return std::unique_ptr<T>(new T(std::forward<Args>(args)...));
21912         }
21913     }
21914
21915     template<typename T>
21916     struct IGenerator : GeneratorUntypedBase {
21917         virtual ~IGenerator() = default;
21918
21919         // Returns the current element of the generator
21920         //
21921         // \Precondition The generator is either freshly constructed,
21922         // or the last call to `next()` returned true
21923         virtual T const& get() const = 0;
21924         using type = T;
21925     };
21926
21927     template<typename T>
21928     class SingleValueGenerator final : public IGenerator<T> {
21929         T m_value;
21930     public:
21931         SingleValueGenerator(T&& value) : m_value(std::move(value)) {}
21932
21933         T const& get() const override {
21934             return m_value;
21935         }
21936         bool next() override {
21937             return false;
21938         }
21939     };
21940
21941     template<typename T>
21942     class FixedValuesGenerator final : public IGenerator<T> {
21943         static_assert(!std::is_same<T, bool>::value,
21944             "FixedValuesGenerator does not support bools because of std::vector<bool>"
21945             "specialization, use SingleValue Generator instead.");
21946         std::vector<T> m_values;
21947         size_t m_idx = 0;
21948     public:
21949         FixedValuesGenerator( std::initializer_list<T> values ) : m_values( values ) {}
21950
21951         T const& get() const override {
21952             return m_values[m_idx];
21953         }
21954         bool next() override {
21955             ++m_idx;
21956             return m_idx < m_values.size();
21957         }
21958     };
21959
21960     template <typename T>
21961     class GeneratorWrapper final {
21962         std::unique_ptr<IGenerator<T> m_generator;
21963     public:
21964         GeneratorWrapper(std::unique_ptr<IGenerator<T> generator):
21965             m_generator(std::move(generator))
21966         {}
21967         T const& get() const {
21968             return m_generator->get();
21969         }
21970         bool next() {
21971             return m_generator->next();
21972         }
21973     };

```

```

21974
21975     template <typename T>
21976     GeneratorWrapper<T> value(T&& value) {
21977         return GeneratorWrapper<T>(pf::make_unique<SingleValueGenerator<T>(std::forward<T>(value))));
21978     }
21979     template <typename T>
21980     GeneratorWrapper<T> values(std::initializer_list<T> values) {
21981         return GeneratorWrapper<T>(pf::make_unique<FixedValuesGenerator<T>(values)));
21982     }
21983
21984     template<typename T>
21985     class Generators : public IGenerator<T> {
21986     public:
21987         std::vector<GeneratorWrapper<T>> m_generators;
21988         size_t m_current = 0;
21989
21990         void populate(GeneratorWrapper<T>&& generator) {
21991             m_generators.emplace_back(std::move(generator));
21992         }
21993         void populate(T&& val) {
21994             m_generators.emplace_back(value(std::forward<T>(val)));
21995         }
21996         template<typename U>
21997         void populate(U&& val) {
21998             populate(T(std::forward<U>(val)));
21999         }
22000         template<typename U, typename... Gs>
22001         void populate(U&& valueOrGenerator, Gs &&... moreGenerators) {
22002             populate(std::forward<U>(valueOrGenerator));
22003             populate(std::forward<Gs>(moreGenerators)...);
22004         }
22005     public:
22006         template <typename... Gs>
22007         Generators(Gs &&... moreGenerators) {
22008             m_generators.reserve(sizeof...(Gs));
22009             populate(std::forward<Gs>(moreGenerators)...);
22010         }
22011
22012         T const& get() const override {
22013             return m_generators[m_current].get();
22014         }
22015
22016         bool next() override {
22017             if (m_current >= m_generators.size()) {
22018                 return false;
22019             }
22020             const bool current_status = m_generators[m_current].next();
22021             if (!current_status) {
22022                 ++m_current;
22023             }
22024             return m_current < m_generators.size();
22025         }
22026     };
22027
22028     template<typename... Ts>
22029     GeneratorWrapper<std::tuple<Ts...>> table( std::initializer_list<std::tuple<typename
std::decay<Ts>::type...> tuples ) {
22030         return values<std::tuple<Ts...>>( tuples );
22031     }
22032
22033     // Tag type to signal that a generator sequence should convert arguments to a specific type
22034     template <typename T>
22035     struct as {};
22036
22037     template<typename T, typename... Gs>
22038     auto makeGenerators( GeneratorWrapper<T>&& generator, Gs &&... moreGenerators ) -> Generators<T> {
22039         return Generators<T>(std::move(generator), std::forward<Gs>(moreGenerators)...);
22040     }
22041     template<typename T>
22042     auto makeGenerators( GeneratorWrapper<T>&& generator ) -> Generators<T> {
22043         return Generators<T>(std::move(generator));
22044     }
22045     template<typename T, typename... Gs>
22046     auto makeGenerators( T&& val, Gs &&... moreGenerators ) -> Generators<T> {
22047         return makeGenerators( value( std::forward<T>( val ) ), std::forward<Gs>( moreGenerators )...
);
22048     }
22049     template<typename T, typename U, typename... Gs>
22050     auto makeGenerators( as<T>, U&& val, Gs &&... moreGenerators ) -> Generators<T> {
22051         return makeGenerators( value( T( std::forward<U>( val ) ) ), std::forward<Gs>( moreGenerators
)... );
22052     }
22053
22054     auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
IGeneratorTracker&;
22055
22056     template<typename L>

```

```

22057 // Note: The type after -> is weird, because VS2015 cannot parse
22058 // the expression used in the typedef inside, when it is in
22059 // return type. Yeah.
22060 auto generate( StringRef generatorName, SourceLineInfo const& lineInfo, L const&
generatorExpression ) -> decltype( std::declval<decltype(generatorExpression())>().get() ) {
22061     using UnderlyingType = typename decltype(generatorExpression())::type;
22062
22063     IGeneratorTracker& tracker = acquireGeneratorTracker( generatorName, lineInfo );
22064     if (!tracker.hasGenerator()) {
22065         tracker.setGenerator(pf::make_unique<Generators<UnderlyingType>>(generatorExpression()));
22066     }
22067
22068     auto const& generator = static_cast<IGenerator<UnderlyingType>> const&>(
*tracker.getGenerator() );
22069     return generator.get();
22070 }
22071
22072 } // namespace Generators
22073 } // namespace Catch
22074
22075 #define GENERATE( ... ) \
22076     Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
22077     CATCH_INTERNAL_LINEINFO, \
22078     [ ]{ using namespace Catch::Generators; return makeGenerators(
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
22079 #define GENERATE_COPY( ... ) \
22080     Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
22081     CATCH_INTERNAL_LINEINFO, \
22082     [ ]{ using namespace Catch::Generators; return makeGenerators(
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
22083 #define GENERATE_REF( ... ) \
22084     Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
22085     CATCH_INTERNAL_LINEINFO, \
22086     [ & ]{ using namespace Catch::Generators; return makeGenerators(
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
22087
22088 // end catch_generators.hpp
22089 // start catch_generators_generic.hpp
22090
22091 namespace Catch {
22092 namespace Generators {
22093
22094     template <typename T>
22095     class TakeGenerator : public IGenerator<T> {
22096     public:
22097         GeneratorWrapper<T> m_generator;
22098         size_t m_returned = 0;
22099         size_t m_target;
22100
22101         TakeGenerator(size_t target, GeneratorWrapper<T>&& generator):
22102             m_generator(std::move(generator)),
22103             m_target(target)
22104         {
22105             assert(target != 0 && "Empty generators are not allowed");
22106         }
22107         T const& get() const override {
22108             return m_generator.get();
22109         }
22110         bool next() override {
22111             ++m_returned;
22112             if (m_returned >= m_target) {
22113                 return false;
22114             }
22115             const auto success = m_generator.next();
22116             // If the underlying generator does not contain enough values
22117             // then we cut short as well
22118             if (!success) {
22119                 m_returned = m_target;
22120             }
22121             return success;
22122         }
22123     };
22124
22125     template <typename T>
22126     GeneratorWrapper<T> take(size_t target, GeneratorWrapper<T>&& generator) {
22127         return GeneratorWrapper<T>(pf::make_unique<TakeGenerator<T>>(target, std::move(generator)));
22128     }
22129
22130     template <typename T, typename Predicate>
22131     class FilterGenerator : public IGenerator<T> {
22132     public:
22133         GeneratorWrapper<T> m_generator;
22134         Predicate m_predicate;
22135
22136         template <typename P = Predicate>
22137         FilterGenerator(P&& pred, GeneratorWrapper<T>&& generator):
22138             m_generator(std::move(generator)),
22139             m_predicate(std::forward<P>(pred))
22140     };

```



```

22139     {
22140         if (!m_predicate(m_generator.get())) {
22141             // It might happen that there are no values that pass the
22142             // filter. In that case we throw an exception.
22143             auto has_initial_value = nextImpl();
22144             if (!has_initial_value) {
22145                 Catch::throw_exception(GeneratorException("No valid value found in filtered
generator"));
22146             }
22147         }
22148     }
22149
22150     T const& get() const override {
22151         return m_generator.get();
22152     }
22153
22154     bool next() override {
22155         return nextImpl();
22156     }
22157
22158 private:
22159     bool nextImpl() {
22160         bool success = m_generator.next();
22161         if (!success) {
22162             return false;
22163         }
22164         while (!m_predicate(m_generator.get()) && (success = m_generator.next()) == true);
22165         return success;
22166     }
22167 };
22168
22169 template <typename T, typename Predicate>
22170 GeneratorWrapper<T> filter(Predicate&& pred, GeneratorWrapper<T>&& generator) {
22171     return GeneratorWrapper<T>(std::unique_ptr<IGenerator<T>>(pf::make_unique<FilterGenerator<T,
Predicate>>(std::forward<Predicate>(pred), std::move(generator))));
22172 }
22173
22174 template <typename T>
22175 class RepeatGenerator : public IGenerator<T> {
22176     static_assert(!std::is_same<T, bool>::value,
22177         "RepeatGenerator currently does not support bools"
22178         "because of std::vector<bool> specialization");
22179     GeneratorWrapper<T> m_generator;
22180     mutable std::vector<T> m_returned;
22181     size_t m_target_repeats;
22182     size_t m_current_repeat = 0;
22183     size_t m_repeat_index = 0;
22184 public:
22185     RepeatGenerator(size_t repeats, GeneratorWrapper<T>&& generator):
22186         m_generator(std::move(generator)),
22187         m_target_repeats(repeats)
22188     {
22189         assert(m_target_repeats > 0 && "Repeat generator must repeat at least once");
22190     }
22191
22192     T const& get() const override {
22193         if (m_current_repeat == 0) {
22194             m_returned.push_back(m_generator.get());
22195             return m_returned.back();
22196         }
22197         return m_returned[m_repeat_index];
22198     }
22199
22200     bool next() override {
22201         // There are 2 basic cases:
22202         // 1) We are still reading the generator
22203         // 2) We are reading our own cache
22204
22205         // In the first case, we need to poke the underlying generator.
22206         // If it happily moves, we are left in that state, otherwise it is time to start reading
from our cache
22207         if (m_current_repeat == 0) {
22208             const auto success = m_generator.next();
22209             if (!success) {
22210                 ++m_current_repeat;
22211             }
22212             return m_current_repeat < m_target_repeats;
22213         }
22214
22215         // In the second case, we need to move indices forward and check that we haven't run up
against the end
22216         ++m_repeat_index;
22217         if (m_repeat_index == m_returned.size()) {
22218             m_repeat_index = 0;
22219             ++m_current_repeat;
22220         }
22221         return m_current_repeat < m_target_repeats;

```

```

22222     }
22223 };
22224
22225 template <typename T>
22226 GeneratorWrapper<T> repeat(size_t repeats, GeneratorWrapper<T>&& generator) {
22227     return GeneratorWrapper<T>(pf::make_unique<RepeatGenerator<T>(repeats, std::move(generator))));
22228 }
22229
22230 template <typename T, typename U, typename Func>
22231 class MapGenerator : public IGenerator<T> {
22232     // TBD: provide static assert for mapping function, for friendly error message
22233     GeneratorWrapper<U> m_generator;
22234     Func m_function;
22235     // To avoid returning dangling reference, we have to save the values
22236     T m_cache;
22237 public:
22238     template <typename F2 = Func>
22239     MapGenerator(F2&& function, GeneratorWrapper<U>&& generator) :
22240         m_generator(std::move(generator)),
22241         m_function(std::forward<F2>(function)),
22242         m_cache(m_function(m_generator.get()))
22243     {}
22244
22245     T const& get() const override {
22246         return m_cache;
22247     }
22248     bool next() override {
22249         const auto success = m_generator.next();
22250         if (success) {
22251             m_cache = m_function(m_generator.get());
22252         }
22253         return success;
22254     }
22255 };
22256
22257 template <typename Func, typename U, typename T = FunctionReturnType<Func, U>
22258 GeneratorWrapper<T> map(Func&& function, GeneratorWrapper<U>&& generator) {
22259     return GeneratorWrapper<T>(
22260         pf::make_unique<MapGenerator<T, U, Func>(std::forward<Func>(function),
22261         std::move(generator))
22262     );
22263 }
22264
22265 template <typename T, typename U, typename Func>
22266 GeneratorWrapper<T> map(Func&& function, GeneratorWrapper<U>&& generator) {
22267     return GeneratorWrapper<T>(
22268         pf::make_unique<MapGenerator<T, U, Func>(std::forward<Func>(function),
22269         std::move(generator))
22270     );
22271 }
22272
22273 template <typename T>
22274 class ChunkGenerator final : public IGenerator<std::vector<T>> {
22275     std::vector<T> m_chunk;
22276     size_t m_chunk_size;
22277     GeneratorWrapper<T> m_generator;
22278     bool m_used_up = false;
22279 public:
22280     ChunkGenerator(size_t size, GeneratorWrapper<T> generator) :
22281         m_chunk_size(size), m_generator(std::move(generator))
22282     {
22283         m_chunk.reserve(m_chunk_size);
22284         if (m_chunk_size != 0) {
22285             m_chunk.push_back(m_generator.get());
22286             for (size_t i = 1; i < m_chunk_size; ++i) {
22287                 if (!m_generator.next()) {
22288                     Catch::throw_exception(GeneratorException("Not enough values to initialize the
22289 first chunk"));
22290                 }
22291             }
22292             m_chunk.push_back(m_generator.get());
22293         }
22294     }
22295     std::vector<T> const& get() const override {
22296         return m_chunk;
22297     }
22298     bool next() override {
22299         m_chunk.clear();
22300         for (size_t idx = 0; idx < m_chunk_size; ++idx) {
22301             if (!m_generator.next()) {
22302                 return false;
22303             }
22304             m_chunk.push_back(m_generator.get());
22305         }
22306         return true;
22307     }
22308 };

```

```

22306
22307     template <typename T>
22308     GeneratorWrapper<std::vector<T>> chunk(size_t size, GeneratorWrapper<T>&& generator) {
22309         return GeneratorWrapper<std::vector<T>>{
22310             pf::make_unique<ChunkGenerator<T>>(size, std::move(generator))
22311         };
22312     }
22313
22314 } // namespace Generators
22315 } // namespace Catch
22316
22317 // end catch_generators_generic.hpp
22318 // start catch_generators_specific.hpp
22319
22320 // start catch_context.h
22321
22322 #include <memory>
22323
22324 namespace Catch {
22325
22326     struct IResultCapture;
22327     struct IRunner;
22328     struct IConfig;
22329     struct IMutableContext;
22330
22331     using IConfigPtr = std::shared_ptr<IConfig const>;
22332
22333     struct IContext
22334     {
22335         virtual ~IContext();
22336
22337         virtual IResultCapture* getResultCapture() = 0;
22338         virtual IRunner* getRunner() = 0;
22339         virtual IConfigPtr const& getConfig() const = 0;
22340     };
22341
22342     struct IMutableContext : IContext
22343     {
22344         virtual ~IMutableContext();
22345         virtual void setResultCapture( IResultCapture* resultCapture ) = 0;
22346         virtual void setRunner( IRunner* runner ) = 0;
22347         virtual void setConfig( IConfigPtr const& config ) = 0;
22348
22349     private:
22350         static IMutableContext *currentContext;
22351         friend IMutableContext& getCurrentMutableContext();
22352         friend void cleanUpContext();
22353         static void createContext();
22354     };
22355
22356     inline IMutableContext& getCurrentMutableContext()
22357     {
22358         if( !IMutableContext::currentContext )
22359             IMutableContext::createContext();
22360         // NOLINTNEXTLINE(clang-analyzer-core.uninitialized.UndefReturn)
22361         return *IMutableContext::currentContext;
22362     }
22363
22364     inline IContext& getCurrentContext()
22365     {
22366         return getCurrentMutableContext();
22367     }
22368
22369     void cleanUpContext();
22370
22371     class SimplePcg32;
22372     SimplePcg32& rng();
22373 }
22374
22375 // end catch_context.h
22376 // start catch_interfaces_config.h
22377
22378 // start catch_option.hpp
22379
22380 namespace Catch {
22381
22382     // An optional type
22383     template<typename T>
22384     class Option {
22385     public:
22386         Option() : nullableValue( nullptr ) {}
22387         Option( T const& _value )
22388             : nullableValue( new( storage ) T( _value ) )
22389         {}
22390         Option( Option const& _other )
22391             : nullableValue( _other ? new( storage ) T( *_other ) : nullptr )
22392         {}

```

```

22393
22394 ~Option() {
22395     reset();
22396 }
22397
22398 Option& operator= ( Option const& _other ) {
22399     if( &_other != this ) {
22400         reset();
22401         if( !_other )
22402             nullableValue = new( storage ) T( *_other );
22403     }
22404     return *this;
22405 }
22406 Option& operator = ( T const& _value ) {
22407     reset();
22408     nullableValue = new( storage ) T( _value );
22409     return *this;
22410 }
22411
22412 void reset() {
22413     if( nullableValue )
22414         nullableValue->~T();
22415     nullableValue = nullptr;
22416 }
22417
22418 T& operator*() { return *nullableValue; }
22419 T const& operator*() const { return *nullableValue; }
22420 T* operator->() { return nullableValue; }
22421 const T* operator->() const { return nullableValue; }
22422
22423 T valueOr( T const& defaultValue ) const {
22424     return nullableValue ? *nullableValue : defaultValue;
22425 }
22426
22427 bool some() const { return nullableValue != nullptr; }
22428 bool none() const { return nullableValue == nullptr; }
22429
22430 bool operator !() const { return nullableValue == nullptr; }
22431 explicit operator bool() const {
22432     return some();
22433 }
22434
22435 private:
22436     T *nullableValue;
22437     alignas(alignof(T)) char storage[sizeof(T)];
22438 };
22439
22440 } // end namespace Catch
22441
22442 // end catch_option.hpp
22443 #include <chrono>
22444 #include <iosfwd>
22445 #include <string>
22446 #include <vector>
22447 #include <memory>
22448
22449 namespace Catch {
22450
22451     enum class Verbosity {
22452         Quiet = 0,
22453         Normal,
22454         High
22455     };
22456
22457     struct WarnAbout { enum What {
22458         Nothing = 0x00,
22459         NoAssertions = 0x01,
22460         NoTests = 0x02
22461     }; };
22462
22463     struct ShowDurations { enum OrNot {
22464         DefaultForReporter,
22465         Always,
22466         Never
22467     }; };
22468     struct RunTests { enum InWhatOrder {
22469         InDeclarationOrder,
22470         InLexicographicalOrder,
22471         InRandomOrder
22472     }; };
22473     struct UseColour { enum YesOrNo {
22474         Auto,
22475         Yes,
22476         No
22477     }; };
22478     struct WaitForKeypress { enum When {
22479         Never,

```

```

22480         BeforeStart = 1,
22481         BeforeExit = 2,
22482         BeforeStartAndExit = BeforeStart | BeforeExit
22483     }; };
22484
22485     class TestSpec;
22486
22487     struct IConfig : NonCopyable {
22488
22489         virtual ~IConfig();
22490
22491         virtual bool allowThrows() const = 0;
22492         virtual std::ostream& stream() const = 0;
22493         virtual std::string name() const = 0;
22494         virtual bool includeSuccessfulResults() const = 0;
22495         virtual bool shouldDebugBreak() const = 0;
22496         virtual bool warnAboutMissingAssertions() const = 0;
22497         virtual bool warnAboutNoTests() const = 0;
22498         virtual int abortAfter() const = 0;
22499         virtual bool showInvisibles() const = 0;
22500         virtual ShowDurations::OrNot showDurations() const = 0;
22501         virtual double minDuration() const = 0;
22502         virtual TestSpec const& testSpec() const = 0;
22503         virtual bool hasTestFilters() const = 0;
22504         virtual std::vector<std::string> const& getTestsOrTags() const = 0;
22505         virtual RunTests::InWhatOrder runOrder() const = 0;
22506         virtual unsigned int rngSeed() const = 0;
22507         virtual UseColour::YesOrNo useColour() const = 0;
22508         virtual std::vector<std::string> const& getSectionsToRun() const = 0;
22509         virtual Verbosity verbosity() const = 0;
22510
22511         virtual bool benchmarkNoAnalysis() const = 0;
22512         virtual int benchmarkSamples() const = 0;
22513         virtual double benchmarkConfidenceInterval() const = 0;
22514         virtual unsigned int benchmarkResamples() const = 0;
22515         virtual std::chrono::milliseconds benchmarkWarmupTime() const = 0;
22516     };
22517
22518     using IConfigPtr = std::shared_ptr<IConfig const>;
22519 }
22520
22521 // end catch_interfaces_config.h
22522 // start catch_random_number_generator.h
22523
22524 #include <cstdint>
22525
22526 namespace Catch {
22527
22528     // This is a simple implementation of C++11 Uniform Random Number
22529     // Generator. It does not provide all operators, because Catch2
22530     // does not use it, but it should behave as expected inside stdlib's
22531     // distributions.
22532     // The implementation is based on the PCG family (http://pcg-random.org)
22533     class SimplePcg32 {
22534     public:
22535         using state_type = std::uint64_t;
22536         using result_type = std::uint32_t;
22537         static constexpr result_type (min)() {
22538             return 0;
22539         }
22540         static constexpr result_type (max)() {
22541             return static_cast<result_type>(-1);
22542         }
22543
22544         // Provide some default initial state for the default constructor
22545         SimplePcg32(): SimplePcg32(0xed743cc4U) {}
22546
22547         explicit SimplePcg32(result_type seed_);
22548
22549         void seed(result_type seed_);
22550         void discard(uint64_t skip);
22551
22552         result_type operator()();
22553
22554     private:
22555         friend bool operator==(SimplePcg32 const& lhs, SimplePcg32 const& rhs);
22556         friend bool operator!=(SimplePcg32 const& lhs, SimplePcg32 const& rhs);
22557
22558         // In theory we also need operator< and operator>
22559         // In practice we do not use them, so we will skip them for now
22560
22561         std::uint64_t m_state;
22562         // This part of the state determines which "stream" of the numbers
22563         // is chosen -- we take it as a constant for Catch2, so we only
22564         // need to deal with seeding the main state.
22565         // Picked by reading 8 bytes from `/dev/random` :-)
22566         static const std::uint64_t s_inc = (0x13ed0cc53f939476ULL « 1ULL) | 1ULL;

```

```

22567     };
22568
22569 } // end namespace Catch
22570
22571 // end catch_random_number_generator.h
22572 #include <random>
22573
22574 namespace Catch {
22575 namespace Generators {
22576
22577 template <typename Float>
22578 class RandomFloatingGenerator final : public IGenerator<Float> {
22579     Catch::SimplePcg32& m_rng;
22580     std::uniform_real_distribution<Float> m_dist;
22581     Float m_current_number;
22582 public:
22583
22584     RandomFloatingGenerator(Float a, Float b):
22585         m_rng(rng()),
22586         m_dist(a, b) {
22587             static_cast<void>(next());
22588         }
22589
22590     Float const& get() const override {
22591         return m_current_number;
22592     }
22593     bool next() override {
22594         m_current_number = m_dist(m_rng);
22595         return true;
22596     }
22597 };
22598
22599 template <typename Integer>
22600 class RandomIntegerGenerator final : public IGenerator<Integer> {
22601     Catch::SimplePcg32& m_rng;
22602     std::uniform_int_distribution<Integer> m_dist;
22603     Integer m_current_number;
22604 public:
22605
22606     RandomIntegerGenerator(Integer a, Integer b):
22607         m_rng(rng()),
22608         m_dist(a, b) {
22609             static_cast<void>(next());
22610         }
22611
22612     Integer const& get() const override {
22613         return m_current_number;
22614     }
22615     bool next() override {
22616         m_current_number = m_dist(m_rng);
22617         return true;
22618     }
22619 };
22620
22621 // TODO: Ideally this would be also constrained against the various char types,
22622 // but I don't expect users to run into that in practice.
22623 template <typename T>
22624 typename std::enable_if<std::is_integral<T>::value && !std::is_same<T, bool>::value,
22625 GeneratorWrapper<T>::type
22626 random(T a, T b) {
22627     return GeneratorWrapper<T>{
22628         pf::make_unique<RandomIntegerGenerator<T>(a, b)
22629     };
22630 }
22631
22632 template <typename T>
22633 typename std::enable_if<std::is_floating_point<T>::value,
22634 GeneratorWrapper<T>::type
22635 random(T a, T b) {
22636     return GeneratorWrapper<T>{
22637         pf::make_unique<RandomFloatingGenerator<T>(a, b)
22638     };
22639 }
22640
22641 template <typename T>
22642 class RangeGenerator final : public IGenerator<T> {
22643     T m_current;
22644     T m_end;
22645     T m_step;
22646     bool m_positive;
22647 public:
22648     RangeGenerator(T const& start, T const& end, T const& step):
22649         m_current(start),
22650         m_end(end),
22651         m_step(step),
22652         m_positive(m_step > T(0))
22653

```

```

22654     {
22655         assert(m_current != m_end && "Range start and end cannot be equal");
22656         assert(m_step != T(0) && "Step size cannot be zero");
22657         assert(((m_positive && m_current <= m_end) || (!m_positive && m_current >= m_end)) && "Step
moves away from end");
22658     }
22659
22660     RangeGenerator(T const& start, T const& end):
22661     {
22662         RangeGenerator(start, end, (start < end) ? T(1) : T(-1))
22663     }
22664
22665     T const& get() const override {
22666         return m_current;
22667     }
22668
22669     bool next() override {
22670         m_current += m_step;
22671         return (m_positive) ? (m_current < m_end) : (m_current > m_end);
22672     };
22673
22674 template <typename T>
22675 GeneratorWrapper<T> range(T const& start, T const& end, T const& step) {
22676     static_assert(std::is_arithmetic<T>::value && !std::is_same<T, bool>::value, "Type must be
numeric");
22677     return GeneratorWrapper<T>(pf::make_unique<RangeGenerator<T>>(start, end, step));
22678 }
22679
22680 template <typename T>
22681 GeneratorWrapper<T> range(T const& start, T const& end) {
22682     static_assert(std::is_integral<T>::value && !std::is_same<T, bool>::value, "Type must be an
integer");
22683     return GeneratorWrapper<T>(pf::make_unique<RangeGenerator<T>>(start, end));
22684 }
22685
22686 template <typename T>
22687 class IteratorGenerator final : public IGenerator<T> {
22688     static_assert(!std::is_same<T, bool>::value,
22689         "IteratorGenerator currently does not support bools"
22690         "because of std::vector<bool> specialization");
22691
22692     std::vector<T> m_elems;
22693     size_t m_current = 0;
22694 public:
22695     template <typename InputIterator, typename InputSentinel>
22696     IteratorGenerator(InputIterator first, InputSentinel last):m_elems(first, last) {
22697         if (m_elems.empty()) {
22698             Catch::throw_exception(GeneratorException("IteratorGenerator received no valid values"));
22699         }
22700     }
22701
22702     T const& get() const override {
22703         return m_elems[m_current];
22704     }
22705
22706     bool next() override {
22707         ++m_current;
22708         return m_current != m_elems.size();
22709     }
22710 };
22711
22712 template <typename InputIterator,
22713         typename InputSentinel,
22714         typename ResultType = typename std::iterator_traits<InputIterator>::value_type>
22715 GeneratorWrapper<ResultType> from_range(InputIterator from, InputSentinel to) {
22716     return GeneratorWrapper<ResultType>(pf::make_unique<IteratorGenerator<ResultType>>(from, to));
22717 }
22718
22719 template <typename Container,
22720         typename ResultType = typename Container::value_type>
22721 GeneratorWrapper<ResultType> from_range(Container const& cnt) {
22722     return GeneratorWrapper<ResultType>(pf::make_unique<IteratorGenerator<ResultType>>(cnt.begin(),
cnt.end()));
22723 }
22724
22725 } // namespace Generators
22726 } // namespace Catch
22727
22728 // end catch_generators_specific.hpp
22729
22730 // These files are included here so the single_include script doesn't put them
22731 // in the conditionally compiled sections
22732 // start catch_test_case_info.h
22733
22734 #include <string>
22735 #include <vector>
22736 #include <memory>

```

```

22737
22738 #ifdef __clang__
22739 #pragma clang diagnostic push
22740 #pragma clang diagnostic ignored "-Wpadded"
22741 #endif
22742
22743 namespace Catch {
22744
22745     struct ITestInvoker;
22746
22747     struct TestCaseInfo {
22748         enum SpecialProperties{
22749             None = 0,
22750             IsHidden = 1 << 1,
22751             ShouldFail = 1 << 2,
22752             MayFail = 1 << 3,
22753             Throws = 1 << 4,
22754             NonPortable = 1 << 5,
22755             Benchmark = 1 << 6
22756         };
22757
22758         TestCaseInfo( std::string const& _name,
22759                     std::string const& _className,
22760                     std::string const& _description,
22761                     std::vector<std::string> const& _tags,
22762                     SourceLineInfo const& _lineInfo );
22763
22764         friend void setTags( TestCaseInfo& testCaseInfo, std::vector<std::string> tags );
22765
22766         bool isHidden() const;
22767         bool throws() const;
22768         bool okToFail() const;
22769         bool expectedToFail() const;
22770
22771         std::string tagsAsString() const;
22772
22773         std::string name;
22774         std::string className;
22775         std::string description;
22776         std::vector<std::string> tags;
22777         std::vector<std::string> lcCaseTags;
22778         SourceLineInfo lineInfo;
22779         SpecialProperties properties;
22780     };
22781
22782     class TestCase : public TestCaseInfo {
22783     public:
22784
22785         TestCase( ITestInvoker* testCase, TestCaseInfo&& info );
22786
22787         TestCase withName( std::string const& _newName ) const;
22788
22789         void invoke() const;
22790
22791         TestCaseInfo const& getTestCaseInfo() const;
22792
22793         bool operator == ( TestCase const& other ) const;
22794         bool operator < ( TestCase const& other ) const;
22795
22796     private:
22797         std::shared_ptr<ITestInvoker> test;
22798     };
22799
22800     TestCase makeTestCase( ITestInvoker* testCase,
22801                          std::string const& className,
22802                          NameAndTags const& nameAndTags,
22803                          SourceLineInfo const& lineInfo );
22804 }
22805
22806 #ifdef __clang__
22807 #pragma clang diagnostic pop
22808 #endif
22809
22810 // end catch_test_case_info.h
22811 // start catch_interfaces_runner.h
22812
22813 namespace Catch {
22814
22815     struct IRunner {
22816         virtual ~IRunner();
22817         virtual bool aborting() const = 0;
22818     };
22819 }
22820
22821 // end catch_interfaces_runner.h
22822
22823 #ifdef __OBJC__

```



```

22824 // start catch_objc.hpp
22825
22826 #import <objc/runtime.h>
22827
22828 #include <string>
22829
22830 // NB. Any general catch headers included here must be included
22831 // in catch.hpp first to make sure they are included by the single
22832 // header for non objc-usage
22833
22835 // This protocol is really only here for (self) documenting purposes, since
22836 // all its methods are optional.
22837 @protocol OcFixture
22838
22839 @optional
22840
22841 -(void) setUp;
22842 -(void) tearDown;
22843
22844 @end
22845
22846 namespace Catch {
22847
22848     class OcMethod : public ITestInvoker {
22849     public:
22850         OcMethod( Class cls, SEL sel ) : m_cls( cls ), m_sel( sel ) {}
22851
22852         virtual void invoke() const {
22853             id obj = [[m_cls alloc] init];
22854
22855             performOptionalSelector( obj, @selector(setUp) );
22856             performOptionalSelector( obj, m_sel );
22857             performOptionalSelector( obj, @selector(tearDown) );
22858
22859             arcSafeRelease( obj );
22860         }
22861     private:
22862         virtual ~OcMethod() {}
22863
22864         Class m_cls;
22865         SEL m_sel;
22866     };
22867
22868     namespace Detail{
22869
22870         inline std::string getAnnotation( Class cls,
22871             std::string const& annotationName,
22872             std::string const& testCaseName ) {
22873             NSString* selStr = [[NSString alloc] initWithFormat:@"Catch_%s_%s",
22874                 annotationName.c_str(), testCaseName.c_str()];
22875             SEL sel = NSSelectorFromString( selStr );
22876             arcSafeRelease( selStr );
22877             id value = performOptionalSelector( cls, sel );
22878             if( value )
22879                 return [(NSString*)value UTF8String];
22880             return "";
22881         }
22882     }
22883
22884     inline std::size_t registerTestMethods() {
22885         std::size_t noTestMethods = 0;
22886         int noClasses = objc_getClassList( nullptr, 0 );
22887
22888         Class* classes = (CATCH_UNSAFE_UNRETAINED Class *)malloc( sizeof(Class) * noClasses);
22889         objc_getClassList( classes, noClasses );
22890
22891         for( int c = 0; c < noClasses; c++ ) {
22892             Class cls = classes[c];
22893             {
22894                 u_int count;
22895                 Method* methods = class_copyMethodList( cls, &count );
22896                 for( u_int m = 0; m < count ; m++ ) {
22897                     SEL selector = method_getName(methods[m]);
22898                     std::string methodName = sel_getName(selector);
22899                     if( startsWith( methodName, "Catch_TestCase_" ) ) {
22900                         std::string testCaseName = methodName.substr( 15 );
22901                         std::string name = Detail::getAnnotation( cls, "Name", testCaseName );
22902                         std::string desc = Detail::getAnnotation( cls, "Description", testCaseName );
22903                         const char* className = class_getName( cls );
22904
22905                         getMutableRegistryHub().registerTest( makeTestCase( new OcMethod( cls,
22906                             selector ), className, NameAndTags( name.c_str(), desc.c_str() ), SourceLineInfo("",0) ) );
22907                         noTestMethods++;
22908                     }
22909                 }
22910             }
22911             free(methods);
22912         }
22913     }

```

```

22910     }
22911 }
22912     return noTestMethods;
22913 }
22914
22915 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
22916 namespace Matchers {
22917     namespace Impl {
22918         namespace NSStringMatchers {
22919
22920             struct StringHolder : MatcherBase<NSString*>{
22921                 StringHolder( NSString* substr ) : m_substr( [substr copy] ){}
22922                 StringHolder( StringHolder const& other ) : m_substr( [other.m_substr copy] ){}
22923                 StringHolder() {
22924                     arcSafeRelease( m_substr );
22925                 }
22926
22927                 bool match( NSString* str ) const override {
22928                     return false;
22929                 }
22930
22931                 NSString* CATCH_ARC_STRONG m_substr;
22932             };
22933
22934             struct Equals : StringHolder {
22935                 Equals( NSString* substr ) : StringHolder( substr ){}
22936
22937                 bool match( NSString* str ) const override {
22938                     return (str != nil || m_substr == nil) &&
22939                         [str isEqualToString:m_substr];
22940                 }
22941
22942                 std::string describe() const override {
22943                     return "equals string: " + Catch::Detail::stringify( m_substr );
22944                 }
22945             };
22946
22947             struct Contains : StringHolder {
22948                 Contains( NSString* substr ) : StringHolder( substr ){}
22949
22950                 bool match( NSString* str ) const override {
22951                     return (str != nil || m_substr == nil) &&
22952                         [str rangeOfString:m_substr].location != NSNotFound;
22953                 }
22954
22955                 std::string describe() const override {
22956                     return "contains string: " + Catch::Detail::stringify( m_substr );
22957                 }
22958             };
22959
22960             struct StartsWith : StringHolder {
22961                 StartsWith( NSString* substr ) : StringHolder( substr ){}
22962
22963                 bool match( NSString* str ) const override {
22964                     return (str != nil || m_substr == nil) &&
22965                         [str rangeOfString:m_substr].location == 0;
22966                 }
22967
22968                 std::string describe() const override {
22969                     return "starts with: " + Catch::Detail::stringify( m_substr );
22970                 }
22971             };
22972
22973             struct EndsWith : StringHolder {
22974                 EndsWith( NSString* substr ) : StringHolder( substr ){}
22975
22976                 bool match( NSString* str ) const override {
22977                     return (str != nil || m_substr == nil) &&
22978                         [str rangeOfString:m_substr].location == [str length] - [m_substr length];
22979                 }
22980
22981                 std::string describe() const override {
22982                     return "ends with: " + Catch::Detail::stringify( m_substr );
22983                 }
22984             };
22985
22986         } // namespace NSStringMatchers
22987     } // namespace Impl
22988
22989     inline Impl::NSStringMatchers::Equals
22990     Equals( NSString* substr ){ return Impl::NSStringMatchers::Equals( substr ); }
22991
22992     inline Impl::NSStringMatchers::Contains
22993     Contains( NSString* substr ){ return Impl::NSStringMatchers::Contains( substr ); }
22994
22995     inline Impl::NSStringMatchers::StartsWith
22996     StartsWith( NSString* substr ){ return Impl::NSStringMatchers::StartsWith( substr ); }

```

```

22997
22998     inline Impl::NSStringMatchers::EndsWith
22999         EndsWith( NSString* substr ){ return Impl::NSStringMatchers::EndsWith( substr ); }
23000
23001     } // namespace Matchers
23002
23003     using namespace Matchers;
23004
23005 #endif // CATCH_CONFIG_DISABLE_MATCHERS
23006
23007 } // namespace Catch
23008
23009 #define OC_MAKE_UNIQUE_NAME( root, uniqueSuffix ) root##uniqueSuffix
23010 #define OC_TEST_CASE2( name, desc, uniqueSuffix ) \
23011     +(NSString*) OC_MAKE_UNIQUE_NAME( Catch_Name_test_, uniqueSuffix ) \
23012     { \
23013         return @ name; \
23014     } \
23015     +(NSString*) OC_MAKE_UNIQUE_NAME( Catch_Description_test_, uniqueSuffix ) \
23016     { \
23017         return @ desc; \
23018     } \
23019     -(void) OC_MAKE_UNIQUE_NAME( Catch_TestCase_test_, uniqueSuffix )
23020
23021 #define OC_TEST_CASE( name, desc ) OC_TEST_CASE2( name, desc, __LINE__ )
23022
23023 // end catch_objc.hpp
23024 #endif
23025
23026 // Benchmarking needs the externally-facing parts of reporters to work
23027 #if defined(CATCH_CONFIG_EXTERNAL_INTERFACES) || defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
23028 // start catch_external_interfaces.h
23029
23030 // start catch_reporter_bases.hpp
23031
23032 // start catch_interfaces_reporter.h
23033
23034 // start catch_config.hpp
23035
23036 // start catch_test_spec_parser.h
23037
23038 #ifdef __clang__
23039 #pragma clang diagnostic push
23040 #pragma clang diagnostic ignored "-Wpadded"
23041 #endif
23042
23043 // start catch_test_spec.h
23044
23045 #ifdef __clang__
23046 #pragma clang diagnostic push
23047 #pragma clang diagnostic ignored "-Wpadded"
23048 #endif
23049
23050 // start catch_wildcard_pattern.h
23051
23052 namespace Catch {
23053 {
23054     class WildcardPattern {
23055     public:
23056         enum WildcardPosition {
23057             NoWildcard = 0,
23058             WildcardAtStart = 1,
23059             WildcardAtEnd = 2,
23060             WildcardAtBothEnds = WildcardAtStart | WildcardAtEnd
23061         };
23062
23063         WildcardPattern( std::string const& pattern, CaseSensitive::Choice caseSensitivity );
23064         virtual ~WildcardPattern() = default;
23065         virtual bool matches( std::string const& str ) const;
23066
23067     private:
23068         std::string normaliseString( std::string const& str ) const;
23069         CaseSensitive::Choice m_caseSensitivity;
23070         WildcardPosition m_wildcard = NoWildcard;
23071         std::string m_pattern;
23072     };
23073 }
23074
23075 // end catch_wildcard_pattern.h
23076
23077 #include <string>
23078 #include <vector>
23079 #include <memory>
23080
23081 namespace Catch {
23082     struct IConfig;

```

```

23085
23086     class TestSpec {
23087     class Pattern {
23088     public:
23089         explicit Pattern( std::string const& name );
23090         virtual ~Pattern();
23091         virtual bool matches( TestCaseInfo const& testCase ) const = 0;
23092         std::string const& name() const;
23093     private:
23094         std::string const m_name;
23095     };
23096     using PatternPtr = std::shared_ptr<Pattern>;
23097
23098     class NamePattern : public Pattern {
23099     public:
23100         explicit NamePattern( std::string const& name, std::string const& filterString );
23101         bool matches( TestCaseInfo const& testCase ) const override;
23102     private:
23103         WildcardPattern m_wildcardPattern;
23104     };
23105
23106     class TagPattern : public Pattern {
23107     public:
23108         explicit TagPattern( std::string const& tag, std::string const& filterString );
23109         bool matches( TestCaseInfo const& testCase ) const override;
23110     private:
23111         std::string m_tag;
23112     };
23113
23114     class ExcludedPattern : public Pattern {
23115     public:
23116         explicit ExcludedPattern( PatternPtr const& underlyingPattern );
23117         bool matches( TestCaseInfo const& testCase ) const override;
23118     private:
23119         PatternPtr m_underlyingPattern;
23120     };
23121
23122     struct Filter {
23123         std::vector<PatternPtr> m_patterns;
23124
23125         bool matches( TestCaseInfo const& testCase ) const;
23126         std::string name() const;
23127     };
23128
23129     public:
23130         struct FilterMatch {
23131             std::string name;
23132             std::vector<TestCase const*> tests;
23133         };
23134         using Matches = std::vector<FilterMatch>;
23135         using vectorStrings = std::vector<std::string>;
23136
23137         bool hasFilters() const;
23138         bool matches( TestCaseInfo const& testCase ) const;
23139         Matches matchesByFilter( std::vector<TestCase> const& testCases, IConfig const& config )
23140     const;
23141         const vectorStrings & getInvalidArgs() const;
23142     private:
23143         std::vector<Filter> m_filters;
23144         std::vector<std::string> m_invalidArgs;
23145         friend class TestSpecParser;
23146     };
23147 }
23148
23149 #ifdef __clang__
23150 #pragma clang diagnostic pop
23151 #endif
23152
23153 // end catch_test_spec.h
23154 // start catch_interfaces_tag_alias_registry.h
23155
23156 #include <string>
23157
23158 namespace Catch {
23159
23160     struct TagAlias;
23161
23162     struct ITagAliasRegistry {
23163         virtual ~ITagAliasRegistry();
23164         // Nullptr if not present
23165         virtual TagAlias const* find( std::string const& alias ) const = 0;
23166         virtual std::string expandAliases( std::string const& unexpandedTestSpec ) const = 0;
23167
23168         static ITagAliasRegistry const& get();
23169     };
23170

```

```

23171 } // end namespace Catch
23172
23173 // end catch_interfaces_tag_alias_registry.h
23174 namespace Catch {
23175
23176     class TestSpecParser {
23177     public:
23178         enum Mode{ None, Name, QuotedName, Tag, EscapedName };
23179         Mode m_mode = None;
23180         Mode lastMode = None;
23181         bool m_exclusion = false;
23182         std::size_t m_pos = 0;
23183         std::size_t m_realPatternPos = 0;
23184         std::string m_arg;
23185         std::string m_substring;
23186         std::string m_patternName;
23187         std::vector<std::size_t> m_escapeChars;
23188         TestSpec::Filter m_currentFilter;
23189         TestSpec m_testSpec;
23190         ITagAliasRegistry const* m_tagAliases = nullptr;
23191
23192     public:
23193         TestSpecParser( ITagAliasRegistry const& tagAliases );
23194
23195         TestSpecParser& parse( std::string const& arg );
23196         TestSpec testSpec();
23197
23198     private:
23199         bool visitChar( char c );
23200         void startNewMode( Mode mode );
23201         bool processNoneChar( char c );
23202         void processNameChar( char c );
23203         bool processOtherChar( char c );
23204         void endMode();
23205         void escape();
23206         bool isControlChar( char c ) const;
23207         void saveLastMode();
23208         void revertBackToLastMode();
23209         void addFilter();
23210         bool separate();
23211
23212         // Handles common preprocessing of the pattern for name/tag patterns
23213         std::string preprocessPattern();
23214         // Adds the current pattern as a test name
23215         void addNamePattern();
23216         // Adds the current pattern as a tag
23217         void addTagPattern();
23218
23219         inline void addCharToPattern(char c) {
23220             m_substring += c;
23221             m_patternName += c;
23222             m_realPatternPos++;
23223         }
23224     };
23225
23226     TestSpec parseTestSpec( std::string const& arg );
23227 } // namespace Catch
23228
23229 #ifdef __clang__
23230 #pragma clang diagnostic pop
23231 #endif
23232
23233 // end catch_test_spec_parser.h
23234 // Libstdc++ doesn't like incomplete classes for unique_ptr
23235
23236 #include <memory>
23237 #include <vector>
23238 #include <string>
23239
23240 #ifndef CATCH_CONFIG_CONSOLE_WIDTH
23241 #define CATCH_CONFIG_CONSOLE_WIDTH 80
23242 #endif
23243
23244 namespace Catch {
23245
23246     struct IStream;
23247
23248     struct ConfigData {
23249         bool listTests = false;
23250         bool listTags = false;
23251         bool listReporters = false;
23252         bool listTestNamesOnly = false;
23253
23254         bool showSuccessfulTests = false;
23255         bool shouldDebugBreak = false;
23256         bool noThrow = false;
23257         bool showHelp = false;

```

```

23258     bool showInvisibles = false;
23259     bool filenamesAsTags = false;
23260     bool libIdentify = false;
23261
23262     int abortAfter = -1;
23263     unsigned int rngSeed = 0;
23264
23265     bool benchmarkNoAnalysis = false;
23266     unsigned int benchmarkSamples = 100;
23267     double benchmarkConfidenceInterval = 0.95;
23268     unsigned int benchmarkResamples = 100000;
23269     std::chrono::milliseconds::rep benchmarkWarmupTime = 100;
23270
23271     Verbosity verbosity = Verbosity::Normal;
23272     WarnAbout::What warnings = WarnAbout::Nothing;
23273     ShowDurations::OrNot showDurations = ShowDurations::DefaultForReporter;
23274     double minDuration = -1;
23275     RunTests::InWhatOrder runOrder = RunTests::InDeclarationOrder;
23276     UseColour::YesOrNo useColour = UseColour::Auto;
23277     WaitForKeypress::When waitForKeypress = WaitForKeypress::Never;
23278
23279     std::string outputFilename;
23280     std::string name;
23281     std::string processName;
23282 #ifndef CATCH_CONFIG_DEFAULT_REPORTER
23283 #define CATCH_CONFIG_DEFAULT_REPORTER "console"
23284 #endif
23285     std::string reporterName = CATCH_CONFIG_DEFAULT_REPORTER;
23286 #undef CATCH_CONFIG_DEFAULT_REPORTER
23287
23288     std::vector<std::string> testsOrTags;
23289     std::vector<std::string> sectionsToRun;
23290 };
23291
23292 class Config : public IConfig {
23293 public:
23294
23295     Config() = default;
23296     Config( ConfigData const& data );
23297     virtual ~Config() = default;
23298
23299     std::string const& getFilename() const;
23300
23301     bool listTests() const;
23302     bool listTestNamesOnly() const;
23303     bool listTags() const;
23304     bool listReporters() const;
23305
23306     std::string getProcessName() const;
23307     std::string const& getReporterName() const;
23308
23309     std::vector<std::string> const& getTestsOrTags() const override;
23310     std::vector<std::string> const& getSectionsToRun() const override;
23311
23312     TestSpec const& testSpec() const override;
23313     bool hasTestFilters() const override;
23314
23315     bool showHelp() const;
23316
23317     // IConfig interface
23318     bool allowThrows() const override;
23319     std::ostream& stream() const override;
23320     std::string name() const override;
23321     bool includeSuccessfulResults() const override;
23322     bool warnAboutMissingAssertions() const override;
23323     bool warnAboutNoTests() const override;
23324     ShowDurations::OrNot showDurations() const override;
23325     double minDuration() const override;
23326     RunTests::InWhatOrder runOrder() const override;
23327     unsigned int rngSeed() const override;
23328     UseColour::YesOrNo useColour() const override;
23329     bool shouldDebugBreak() const override;
23330     int abortAfter() const override;
23331     bool showInvisibles() const override;
23332     Verbosity verbosity() const override;
23333     bool benchmarkNoAnalysis() const override;
23334     int benchmarkSamples() const override;
23335     double benchmarkConfidenceInterval() const override;
23336     unsigned int benchmarkResamples() const override;
23337     std::chrono::milliseconds benchmarkWarmupTime() const override;
23338
23339 private:
23340
23341     IStream const* openStream();
23342     ConfigData m_data;
23343
23344     std::unique_ptr<IStream const> m_stream;

```

```

23345         TestSpec m_testSpec;
23346         bool m_hasTestFilters = false;
23347     };
23348
23349 } // end namespace Catch
23350
23351 // end catch_config.hpp
23352 // start catch_assertionresult.h
23353
23354 #include <string>
23355
23356 namespace Catch {
23357
23358     struct AssertionResultData
23359     {
23360         AssertionResultData() = delete;
23361
23362         AssertionResultData( ResultWas::OfType _resultType, LazyExpression const& _lazyExpression );
23363
23364         std::string message;
23365         mutable std::string reconstructedExpression;
23366         LazyExpression lazyExpression;
23367         ResultWas::OfType resultType;
23368
23369         std::string reconstructExpression() const;
23370     };
23371
23372     class AssertionResult {
23373     public:
23374         AssertionResult() = delete;
23375         AssertionResult( AssertionInfo const& info, AssertionResultData const& data );
23376
23377         bool isOk() const;
23378         bool succeeded() const;
23379         ResultWas::OfType getResultType() const;
23380         bool hasExpression() const;
23381         bool hasMessage() const;
23382         std::string getExpression() const;
23383         std::string getExpressionInMacro() const;
23384         bool hasExpandedExpression() const;
23385         std::string getExpandedExpression() const;
23386         std::string getMessage() const;
23387         SourceLineInfo getSourceInfo() const;
23388         StringRef getTestMacroName() const;
23389
23390     protected:
23391         AssertionInfo m_info;
23392         AssertionResultData m_resultData;
23393     };
23394
23395 } // end namespace Catch
23396
23397 // end catch_assertionresult.h
23398 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
23399 // start catch_estimate.hpp
23400
23401 // Statistics estimates
23402
23403 namespace Catch {
23404     namespace Benchmark {
23405         template <typename Duration>
23406         struct Estimate {
23407             Duration point;
23408             Duration lower_bound;
23409             Duration upper_bound;
23410             double confidence_interval;
23411
23412             template <typename Duration2>
23413             operator Estimate<Duration2>() const {
23414                 return { point, lower_bound, upper_bound, confidence_interval };
23415             }
23416         };
23417     };
23418 } // namespace Benchmark
23419 } // namespace Catch
23420
23421 // end catch_estimate.hpp
23422 // start catch_outlier_classification.hpp
23423
23424 // Outlier information
23425 namespace Catch {
23426     namespace Benchmark {
23427         struct OutlierClassification {
23428             int samples_seen = 0;
23429             int low_severe = 0; // more than 3 times IQR below Q1
23430             int low_mild = 0;   // 1.5 to 3 times IQR below Q1

```

```

23432         int high_mild = 0;        // 1.5 to 3 times IQR above Q3
23433         int high_severe = 0;      // more than 3 times IQR above Q3
23434
23435         int total() const {
23436             return low_severe + low_mild + high_mild + high_severe;
23437         }
23438     };
23439 } // namespace Benchmark
23440 } // namespace Catch
23441
23442 // end catch_outlier_classification.hpp
23443
23444 #include <iterator>
23445 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
23446
23447 #include <string>
23448 #include <iosfwd>
23449 #include <map>
23450 #include <set>
23451 #include <memory>
23452 #include <algorithm>
23453
23454 namespace Catch {
23455
23456     struct ReporterConfig {
23457         explicit ReporterConfig( IConfigPtr const& _fullConfig );
23458
23459         ReporterConfig( IConfigPtr const& _fullConfig, std::ostream& _stream );
23460
23461         std::ostream& stream() const;
23462         IConfigPtr fullConfig() const;
23463
23464     private:
23465         std::ostream* m_stream;
23466         IConfigPtr m_fullConfig;
23467     };
23468
23469     struct ReporterPreferences {
23470         bool shouldRedirectStdOut = false;
23471         bool shouldReportAllAssertions = false;
23472     };
23473
23474     template<typename T>
23475     struct LazyStat : Option<T> {
23476         LazyStat& operator=( T const& _value ) {
23477             Option<T>::operator=( _value );
23478             used = false;
23479             return *this;
23480         }
23481         void reset() {
23482             Option<T>::reset();
23483             used = false;
23484         }
23485         bool used = false;
23486     };
23487
23488     struct TestRunInfo {
23489         TestRunInfo( std::string const& _name );
23490         std::string name;
23491     };
23492     struct GroupInfo {
23493         GroupInfo( std::string const& _name,
23494                   std::size_t _groupIndex,
23495                   std::size_t _groupsCount );
23496
23497         std::string name;
23498         std::size_t groupIndex;
23499         std::size_t groupsCounts;
23500     };
23501
23502     struct AssertionStats {
23503         AssertionStats( AssertionResult const& _assertionResult,
23504                         std::vector<MessageInfo> const& _infoMessages,
23505                         Totals const& _totals );
23506
23507         AssertionStats( AssertionStats const& ) = default;
23508         AssertionStats& operator=( AssertionStats const& ) = default;
23509         AssertionStats& operator=( AssertionStats const& ) = delete;
23510         AssertionStats& operator=( AssertionStats const& ) = delete;
23511         virtual ~AssertionStats();
23512
23513         AssertionResult assertionResult;
23514         std::vector<MessageInfo> infoMessages;
23515         Totals totals;
23516     };
23517
23518     struct SectionStats {

```



```

23519     SectionStats( SectionInfo const& _sectionInfo,
23520                   Counts const& _assertions,
23521                   double _durationInSeconds,
23522                   bool _missingAssertions );
23523     SectionStats( SectionStats const& ) = default;
23524     SectionStats( SectionStats && ) = default;
23525     SectionStats& operator = ( SectionStats const& ) = default;
23526     SectionStats& operator = ( SectionStats && ) = default;
23527     virtual ~SectionStats();
23528
23529     SectionInfo sectionInfo;
23530     Counts assertions;
23531     double durationInSeconds;
23532     bool missingAssertions;
23533 };
23534
23535 struct TestCaseStats {
23536     TestCaseStats( TestCaseInfo const& _testInfo,
23537                   Totals const& _totals,
23538                   std::string const& _stdOut,
23539                   std::string const& _stdErr,
23540                   bool _aborting );
23541
23542     TestCaseStats( TestCaseStats const& ) = default;
23543     TestCaseStats( TestCaseStats && ) = default;
23544     TestCaseStats& operator = ( TestCaseStats const& ) = default;
23545     TestCaseStats& operator = ( TestCaseStats && ) = default;
23546     virtual ~TestCaseStats();
23547
23548     TestCaseInfo testInfo;
23549     Totals totals;
23550     std::string stdOut;
23551     std::string stdErr;
23552     bool aborting;
23553 };
23554
23555 struct TestGroupStats {
23556     TestGroupStats( GroupInfo const& _groupInfo,
23557                    Totals const& _totals,
23558                    bool _aborting );
23559     TestGroupStats( GroupInfo const& _groupInfo );
23560
23561     TestGroupStats( TestGroupStats const& ) = default;
23562     TestGroupStats( TestGroupStats && ) = default;
23563     TestGroupStats& operator = ( TestGroupStats const& ) = default;
23564     TestGroupStats& operator = ( TestGroupStats && ) = default;
23565     virtual ~TestGroupStats();
23566
23567     GroupInfo groupInfo;
23568     Totals totals;
23569     bool aborting;
23570 };
23571
23572 struct TestRunStats {
23573     TestRunStats( TestRunInfo const& _runInfo,
23574                  Totals const& _totals,
23575                  bool _aborting );
23576
23577     TestRunStats( TestRunStats const& ) = default;
23578     TestRunStats( TestRunStats && ) = default;
23579     TestRunStats& operator = ( TestRunStats const& ) = default;
23580     TestRunStats& operator = ( TestRunStats && ) = default;
23581     virtual ~TestRunStats();
23582
23583     TestRunInfo runInfo;
23584     Totals totals;
23585     bool aborting;
23586 };
23587
23588 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
23589 struct BenchmarkInfo {
23590     std::string name;
23591     double estimatedDuration;
23592     int iterations;
23593     int samples;
23594     unsigned int resamples;
23595     double clockResolution;
23596     double clockCost;
23597 };
23598
23599 template <class Duration>
23600 struct BenchmarkStats {
23601     BenchmarkInfo info;
23602
23603     std::vector<Duration> samples;
23604     Benchmark::Estimate<Duration> mean;
23605     Benchmark::Estimate<Duration> standardDeviation;

```

```

23606     Benchmark::OutlierClassification outliers;
23607     double outlierVariance;
23608
23609     template <typename Duration2>
23610     operator BenchmarkStats<Duration2>() const {
23611         std::vector<Duration2> samples2;
23612         samples2.reserve(samples.size());
23613         std::transform(samples.begin(), samples.end(), std::back_inserter(samples2), [](Duration
23614 d) { return Duration2(d); });
23614         return {
23615             info,
23616             std::move(samples2),
23617             mean,
23618             standardDeviation,
23619             outliers,
23620             outlierVariance,
23621         };
23622     }
23623 };
23624 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
23625
23626 struct IStreamingReporter {
23627     virtual ~IStreamingReporter() = default;
23628
23629     // Implementing class must also provide the following static methods:
23630     // static std::string getDescription();
23631     // static std::set<Verbosity> getSupportedVerbsities()
23632
23633     virtual ReporterPreferences getPreferences() const = 0;
23634
23635     virtual void noMatchingTestCases( std::string const& spec ) = 0;
23636
23637     virtual void reportInvalidArguments(std::string const&) {}
23638
23639     virtual void testRunStarting( TestRunInfo const& testRunInfo ) = 0;
23640     virtual void testGroupStarting( GroupInfo const& groupInfo ) = 0;
23641
23642     virtual void testCaseStarting( TestCaseInfo const& testInfo ) = 0;
23643     virtual void sectionStarting( SectionInfo const& sectionInfo ) = 0;
23644
23645 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
23646     virtual void benchmarkPreparing( std::string const& ) {}
23647     virtual void benchmarkStarting( BenchmarkInfo const& ) {}
23648     virtual void benchmarkEnded( BenchmarkStats<> const& ) {}
23649     virtual void benchmarkFailed( std::string const& ) {}
23650 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
23651
23652     virtual void assertionStarting( AssertionInfo const& assertionInfo ) = 0;
23653
23654     // The return value indicates if the messages buffer should be cleared:
23655     virtual bool assertionEnded( AssertionStats const& assertionStats ) = 0;
23656
23657     virtual void sectionEnded( SectionStats const& sectionStats ) = 0;
23658     virtual void testCaseEnded( TestCaseStats const& testCaseStats ) = 0;
23659     virtual void testGroupEnded( TestGroupStats const& testGroupStats ) = 0;
23660     virtual void testRunEnded( TestRunStats const& testRunStats ) = 0;
23661
23662     virtual void skipTest( TestCaseInfo const& testInfo ) = 0;
23663
23664     // Default empty implementation provided
23665     virtual void fatalErrorEncountered( StringRef name );
23666
23667     virtual bool isMulti() const;
23668 };
23669 using IStreamingReporterPtr = std::unique_ptr<IStreamingReporter>;
23670
23671 struct IReporterFactory {
23672     virtual ~IReporterFactory();
23673     virtual IStreamingReporterPtr create( ReporterConfig const& config ) const = 0;
23674     virtual std::string getDescription() const = 0;
23675 };
23676 using IReporterFactoryPtr = std::shared_ptr<IReporterFactory>;
23677
23678 struct IReporterRegistry {
23679     using FactoryMap = std::map<std::string, IReporterFactoryPtr>;
23680     using Listeners = std::vector<IReporterFactoryPtr>;
23681
23682     virtual ~IReporterRegistry();
23683     virtual IStreamingReporterPtr create( std::string const& name, IConfigPtr const& config )
23684 const = 0;
23684     virtual FactoryMap const& getFactories() const = 0;
23685     virtual Listeners const& getListeners() const = 0;
23686 };
23687
23688 } // end namespace Catch
23689
23690 // end catch_interfaces_reporter.h

```

```

23691 #include <algorithm>
23692 #include <cstring>
23693 #include <cfloat>
23694 #include <cstdio>
23695 #include <cassert>
23696 #include <memory>
23697 #include <ostream>
23698
23699 namespace Catch {
23700     void prepareExpandedExpression( AssertionResult& result );
23701
23702     // Returns double formatted as %.3f (format expected on output)
23703     std::string getFormattedDuration( double duration );
23704
23706     bool shouldShowDuration( IConfig const& config, double duration );
23707
23708     std::string serializeFilters( std::vector<std::string> const& container );
23709
23710     template<typename DerivedT>
23711     struct StreamingReporterBase : IStreamingReporter {
23712
23713         StreamingReporterBase( ReporterConfig const& _config )
23714         :   m_config( _config.fullConfig() ),
23715             stream( _config.stream() )
23716         {
23717             m_reporterPrefs.shouldRedirectStdOut = false;
23718             if( !DerivedT::getSupportedVerbsities().count( m_config->verbosity() ) )
23719                 CATCH_ERROR( "Verbosity level not supported by this reporter" );
23720         }
23721
23722         ReporterPreferences getPreferences() const override {
23723             return m_reporterPrefs;
23724         }
23725
23726         static std::set<Verbosity> getSupportedVerbsities() {
23727             return { Verbosity::Normal };
23728         }
23729
23730         ~StreamingReporterBase() override = default;
23731
23732         void noMatchingTestCases( std::string const& ) override {}
23733
23734         void reportInvalidArguments( std::string const& ) override {}
23735
23736         void testRunStarting( TestRunInfo const& _testRunInfo ) override {
23737             currentTestRunInfo = _testRunInfo;
23738         }
23739
23740         void testGroupStarting( GroupInfo const& _groupInfo ) override {
23741             currentGroupInfo = _groupInfo;
23742         }
23743
23744         void testCaseStarting( TestCaseInfo const& _testInfo ) override {
23745             currentTestCaseInfo = _testInfo;
23746         }
23747         void sectionStarting( SectionInfo const& _sectionInfo ) override {
23748             m_sectionStack.push_back( _sectionInfo );
23749         }
23750
23751         void sectionEnded( SectionStats const& /* _sectionStats */ ) override {
23752             m_sectionStack.pop_back();
23753         }
23754         void testCaseEnded( TestCaseStats const& /* _testCaseStats */ ) override {
23755             currentTestCaseInfo.reset();
23756         }
23757         void testGroupEnded( TestGroupStats const& /* _testGroupStats */ ) override {
23758             currentGroupInfo.reset();
23759         }
23760         void testRunEnded( TestRunStats const& /* _testRunStats */ ) override {
23761             currentTestCaseInfo.reset();
23762             currentGroupInfo.reset();
23763             currentTestRunInfo.reset();
23764         }
23765
23766         void skipTest( TestCaseInfo const& ) override {
23767             // Don't do anything with this by default.
23768             // It can optionally be overridden in the derived class.
23769         }
23770
23771         IConfigPtr m_config;
23772         std::ostream& stream;
23773
23774         LazyStat<TestRunInfo> currentTestRunInfo;
23775         LazyStat<GroupInfo> currentGroupInfo;
23776         LazyStat<TestCaseInfo> currentTestCaseInfo;
23777
23778         std::vector<SectionInfo> m_sectionStack;

```

```

23779     ReporterPreferences m_reporterPrefs;
23780 };
23781
23782 template<typename DerivedT>
23783 struct CumulativeReporterBase : IStreamingReporter {
23784     template<typename T, typename ChildNodeT>
23785     struct Node {
23786         explicit Node( T const& _value ) : value( _value ) {}
23787         virtual ~Node() {}
23788
23789         using ChildNodes = std::vector<std::shared_ptr<ChildNodeT>;
23790         T value;
23791         ChildNodes children;
23792     };
23793     struct SectionNode {
23794         explicit SectionNode(SectionStats const& _stats) : stats(_stats) {}
23795         virtual ~SectionNode() = default;
23796
23797         bool operator == (SectionNode const& other) const {
23798             return stats.sectionInfo.lineInfo == other.stats.sectionInfo.lineInfo;
23799         }
23800         bool operator == (std::shared_ptr<SectionNode> const& other) const {
23801             return operator==( *other );
23802         }
23803
23804         SectionStats stats;
23805         using ChildSections = std::vector<std::shared_ptr<SectionNode>;
23806         using Assertions = std::vector<AssertionStats>;
23807         ChildSections childSections;
23808         Assertions assertions;
23809         std::string stdOut;
23810         std::string stdErr;
23811     };
23812
23813     struct BySectionInfo {
23814         BySectionInfo( SectionInfo const& other ) : m_other( other ) {}
23815         BySectionInfo( BySectionInfo const& other ) : m_other( other.m_other ) {}
23816         bool operator () (std::shared_ptr<SectionNode> const& node) const {
23817             return ((node->stats.sectionInfo.name == m_other.name) &&
23818                 (node->stats.sectionInfo.lineInfo == m_other.lineInfo));
23819         }
23820         void operator=(BySectionInfo const&) = delete;
23821
23822     private:
23823         SectionInfo const& m_other;
23824     };
23825
23826     using TestCaseNode = Node<TestCaseStats, SectionNode>;
23827     using TestGroupNode = Node<TestGroupStats, TestCaseNode>;
23828     using TestRunNode = Node<TestRunStats, TestGroupNode>;
23829
23830     CumulativeReporterBase( ReporterConfig const& _config )
23831     : m_config( _config.fullConfig() ),
23832       stream( _config.stream() )
23833     {
23834         m_reporterPrefs.shouldRedirectStdOut = false;
23835         if( !DerivedT::getSupportedVerbsities().count( m_config->verbosity() ) )
23836             CATCH_ERROR( "Verbosity level not supported by this reporter" );
23837     }
23838     ~CumulativeReporterBase() override = default;
23839
23840     ReporterPreferences getPreferences() const override {
23841         return m_reporterPrefs;
23842     }
23843
23844     static std::set<Verbosity> getSupportedVerbsities() {
23845         return { Verbosity::Normal };
23846     }
23847
23848     void testRunStarting( TestRunInfo const& ) override {}
23849     void testGroupStarting( GroupInfo const& ) override {}
23850
23851     void testCaseStarting( TestCaseInfo const& ) override {}
23852
23853     void sectionStarting( SectionInfo const& sectionInfo ) override {
23854         SectionStats incompleteStats( sectionInfo, Counts(), 0, false );
23855         std::shared_ptr<SectionNode> node;
23856         if( m_sectionStack.empty() ) {
23857             if( !m_rootSection )
23858                 m_rootSection = std::make_shared<SectionNode>( incompleteStats );
23859             node = m_rootSection;
23860         }
23861         else {
23862             SectionNode& parentNode = *m_sectionStack.back();
23863             auto it =
23864                 std::find_if( parentNode.childSections.begin(),
23865                             parentNode.childSections.end(),

```

```

23866             BySectionInfo( sectionInfo ) );
23867         if( it == parentNode.childSections.end() ) {
23868             node = std::make_shared<SectionNode>( incompleteStats );
23869             parentNode.childSections.push_back( node );
23870         }
23871         else
23872             node = *it;
23873     }
23874     m_sectionStack.push_back( node );
23875     m_deepestSection = std::move(node);
23876 }
23877
23878 void assertionStarting(AssertionInfo const& override) {}
23879
23880 bool assertionEnded(AssertionStats const& assertionStats) override {
23881     assert(!m_sectionStack.empty());
23882     // AssertionResult holds a pointer to a temporary DecomposedExpression,
23883     // which getExpandedExpression() calls to build the expression string.
23884     // Our section stack copy of the assertionResult will likely outlive the
23885     // temporary, so it must be expanded or discarded now to avoid calling
23886     // a destroyed object later.
23887     prepareExpandedExpression(const_cast<AssertionResult&>( assertionStats.assertionResult ));
23888 };
23889     SectionNode& sectionNode = *m_sectionStack.back();
23890     sectionNode.assertions.push_back( assertionStats );
23891     return true;
23892 }
23893 void sectionEnded(SectionStats const& sectionStats) override {
23894     assert(!m_sectionStack.empty());
23895     SectionNode& node = *m_sectionStack.back();
23896     node.stats = sectionStats;
23897     m_sectionStack.pop_back();
23898 }
23899 void testCaseEnded(TestCaseStats const& testCaseStats) override {
23900     auto node = std::make_shared<TestCaseNode>( testCaseStats );
23901     assert(m_sectionStack.size() == 0);
23902     node->children.push_back( m_rootSection );
23903     m_testCases.push_back( node );
23904     m_rootSection.reset();
23905
23906     assert( m_deepestSection );
23907     m_deepestSection->stdOut = testCaseStats.stdOut;
23908     m_deepestSection->stdErr = testCaseStats.stdErr;
23909 }
23910 void testGroupEnded(TestGroupStats const& testGroupStats) override {
23911     auto node = std::make_shared<TestGroupNode>( testGroupStats );
23912     node->children.swap( m_testCases );
23913     m_testGroups.push_back( node );
23914 }
23915 void testRunEnded(TestRunStats const& testRunStats) override {
23916     auto node = std::make_shared<TestRunNode>( testRunStats );
23917     node->children.swap( m_testGroups );
23918     m_testRuns.push_back( node );
23919     testRunEndedCumulative();
23920 }
23921 virtual void testRunEndedCumulative() = 0;
23922
23923 void skipTest(TestCaseInfo const& override) {}
23924
23925 IConfigPtr m_config;
23926 std::ostream& stream;
23927 std::vector<AssertionStats> m_assertions;
23928 std::vector<std::vector<std::shared_ptr<SectionNode>>> m_sections;
23929 std::vector<std::shared_ptr<TestCaseNode>> m_testCases;
23930 std::vector<std::shared_ptr<TestGroupNode>> m_testGroups;
23931
23932 std::vector<std::shared_ptr<TestRunNode>> m_testRuns;
23933
23934 std::shared_ptr<SectionNode> m_rootSection;
23935 std::shared_ptr<SectionNode> m_deepestSection;
23936 std::vector<std::shared_ptr<SectionNode>> m_sectionStack;
23937 ReporterPreferences m_reporterPrefs;
23938 };
23939
23940 template<char C>
23941 char const* getLineOfChars() {
23942     static char line[CATCH_CONFIG_CONSOLE_WIDTH] = {0};
23943     if( !*line ) {
23944         std::memset( line, C, CATCH_CONFIG_CONSOLE_WIDTH-1 );
23945         line[CATCH_CONFIG_CONSOLE_WIDTH-1] = 0;
23946     }
23947     return line;
23948 }
23949
23950 struct TestEventListenerBase : StreamingReporterBase<TestEventListenerBase> {
23951     TestEventListenerBase( ReporterConfig const& _config );

```

```

23952         static std::set<Verbosity> getSupportedVerbsities();
23953
23954         void assertionStarting(AssertionInfo const&) override;
23955         bool assertionEnded(AssertionStats const&) override;
23956     };
23957
23958 } // end namespace Catch
23959
23960 // end catch_reporter_bases.hpp
23961 // start catch_console_colour.h
23962
23963 namespace Catch {
23964
23965     struct Colour {
23966         enum Code {
23967             None = 0,
23968
23969             White,
23970             Red,
23971             Green,
23972             Blue,
23973             Cyan,
23974             Yellow,
23975             Grey,
23976
23977             Bright = 0x10,
23978
23979             BrightRed = Bright | Red,
23980             BrightGreen = Bright | Green,
23981             LightGrey = Bright | Grey,
23982             BrightWhite = Bright | White,
23983             BrightYellow = Bright | Yellow,
23984
23985             // By intention
23986             FileName = LightGrey,
23987             Warning = BrightYellow,
23988             ResultError = BrightRed,
23989             ResultSuccess = BrightGreen,
23990             ResultExpectedFailure = Warning,
23991
23992             Error = BrightRed,
23993             Success = Green,
23994
23995             OriginalExpression = Cyan,
23996             ReconstructedExpression = BrightYellow,
23997
23998             SecondaryText = LightGrey,
23999             Headers = White
24000         };
24001
24002         // Use constructed object for RAII guard
24003         Colour( Code _colourCode );
24004         Colour( Colour&& other ) noexcept;
24005         Colour& operator=( Colour&& other ) noexcept;
24006         ~Colour();
24007
24008         // Use static method for one-shot changes
24009         static void use( Code _colourCode );
24010
24011     private:
24012         bool m_moved = false;
24013     };
24014
24015     std::ostream& operator << ( std::ostream& os, Colour const& );
24016
24017 } // end namespace Catch
24018
24019 // end catch_console_colour.h
24020 // start catch_reporter_registrars.hpp
24021
24022 namespace Catch {
24023
24024     template<typename T>
24025     class ReporterRegistrar {
24026
24027     public:
24028         class ReporterFactory : public IReporterFactory {
24029
24030         public:
24031             IStreamingReporterPtr create( ReporterConfig const& config ) const override {
24032                 return std::unique_ptr<T>( new T( config ) );
24033             }
24034
24035             std::string getDescription() const override {
24036                 return T::getDescription();
24037             }
24038         };
24039     };

```

```

24039     public:
24040
24041         explicit ReporterRegistrar( std::string const& name ) {
24042             getMutableRegistryHub().registerReporter( name, std::make_shared<ReporterFactory>() );
24043         }
24044     };
24045
24046     template<typename T>
24047     class ListenerRegistrar {
24048     public:
24049         class ListenerFactory : public IReporterFactory {
24050         public:
24051             IStreamingReporterPtr create( ReporterConfig const& config ) const override {
24052                 return std::unique_ptr<T>( new T( config ) );
24053             }
24054             std::string getDescription() const override {
24055                 return std::string();
24056             }
24057         };
24058     public:
24059         ListenerRegistrar() {
24060             getMutableRegistryHub().registerListener( std::make_shared<ListenerFactory>() );
24061         }
24062     };
24063
24064 #if !defined(CATCH_CONFIG_DISABLE)
24065 #define CATCH_REGISTER_REPORTER( name, reporterType ) \
24066     namespace{ Catch::ReporterRegistrar<reporterType> catch_internal_RegistrarFor##reporterType( name \
24067     ); } \
24068     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
24069 #define CATCH_REGISTER_LISTENER( listenerType ) \
24070     namespace{ Catch::ListenerRegistrar<listenerType> catch_internal_RegistrarFor##listenerType; } \
24071     CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
24072 #else // CATCH_CONFIG_DISABLE
24073 #define CATCH_REGISTER_REPORTER( name, reporterType )
24074 #define CATCH_REGISTER_LISTENER( listenerType )
24075 #endif // CATCH_CONFIG_DISABLE
24076 // end catch_reporter_registrars.hpp
24077 // Allow users to base their work off existing reporters
24078 // start catch_reporter_compact.h
24079 namespace Catch {
24080     struct CompactReporter : StreamingReporterBase<CompactReporter> {
24081     public:
24082         using StreamingReporterBase::StreamingReporterBase;
24083         ~CompactReporter() override;
24084         static std::string getDescription();
24085         void noMatchingTestCases( std::string const& spec ) override;
24086         void assertionStarting( AssertionInfo const& ) override;
24087         bool assertionEnded( AssertionStats const& _assertionStats ) override;
24088         void sectionEnded( SectionStats const& _sectionStats ) override;
24089         void testRunEnded( TestRunStats const& _testRunStats ) override;
24090     };
24091 } // end namespace Catch
24092 // end catch_reporter_compact.h
24093 // start catch_reporter_console.h
24094 #if defined(_MSC_VER)
24095 #pragma warning(push)
24096 // Not all labels are EXPLICITLY handled in switch
24097 // Note that 4062 (not all labels are handled
24098 // and default is missing) is enabled
24099 #endif
24100 #endif

```

```

24125 namespace Catch {
24126     // Fwd decls
24127     struct SummaryColumn;
24128     class TablePrinter;
24129
24130     struct ConsoleReporter : StreamingReporterBase<ConsoleReporter> {
24131         std::unique_ptr<TablePrinter> m_tablePrinter;
24132
24133         ConsoleReporter(ReporterConfig const& config);
24134         ~ConsoleReporter() override;
24135         static std::string getDescription();
24136
24137         void noMatchingTestCases(std::string const& spec) override;
24138
24139         void reportInvalidArguments(std::string const& arg) override;
24140
24141         void assertionStarting(AssertionInfo const&) override;
24142
24143         bool assertionEnded(AssertionStats const& _assertionStats) override;
24144
24145         void sectionStarting(SectionInfo const& _sectionInfo) override;
24146         void sectionEnded(SectionStats const& _sectionStats) override;
24147
24148 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
24149         void benchmarkPreparing(std::string const& name) override;
24150         void benchmarkStarting(BenchmarkInfo const& info) override;
24151         void benchmarkEnded(BenchmarkStats<> const& stats) override;
24152         void benchmarkFailed(std::string const& error) override;
24153 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
24154
24155         void testCaseEnded(TestCaseStats const& _testCaseStats) override;
24156         void testGroupEnded(TestGroupStats const& _testGroupStats) override;
24157         void testRunEnded(TestRunStats const& _testRunStats) override;
24158         void testRunStarting(TestRunInfo const& _testRunInfo) override;
24159     private:
24160         void lazyPrint();
24161
24162         void lazyPrintWithoutClosingBenchmarkTable();
24163         void lazyPrintRunInfo();
24164         void lazyPrintGroupInfo();
24165         void printTestCaseAndSectionHeader();
24166
24167         void printClosedHeader(std::string const& _name);
24168         void printOpenHeader(std::string const& _name);
24169
24170         // if string has a : in first line will set indent to follow it on
24171         // subsequent lines
24172         void printHeaderString(std::string const& _string, std::size_t indent = 0);
24173
24174         void printTotals(Totals const& totals);
24175         void printSummaryRow(std::string const& label, std::vector<SummaryColumn> const& cols,
24176             std::size_t row);
24177
24178         void printTotalsDivider(Totals const& totals);
24179         void printSummaryDivider();
24180         void printTestFilters();
24181
24182     private:
24183         bool m_headerPrinted = false;
24184     };
24185 } // end namespace Catch
24186
24187 #if defined(_MSC_VER)
24188 #pragma warning(pop)
24189 #endif
24190
24191 // end catch_reporter_console.h
24192 // start catch_reporter_junit.h
24193
24194 // start catch_xmlwriter.h
24195
24196 #include <vector>
24197
24198 namespace Catch {
24199     enum class XmlFormatting {
24200         None = 0x00,
24201         Indent = 0x01,
24202         Newline = 0x02,
24203     };
24204
24205     XmlFormatting operator | (XmlFormatting lhs, XmlFormatting rhs);
24206     XmlFormatting operator & (XmlFormatting lhs, XmlFormatting rhs);
24207
24208     class XmlEncode {
24209     public:

```



```

24211     enum ForWhat { ForTextNodes, ForAttributes };
24212
24213     XmlEncode( std::string const& str, ForWhat forWhat = ForTextNodes );
24214
24215     void encodeTo( std::ostream& os ) const;
24216
24217     friend std::ostream& operator < ( std::ostream& os, XmlEncode const& xmlEncode );
24218
24219 private:
24220     std::string m_str;
24221     ForWhat m_forWhat;
24222 };
24223
24224 class XmlWriter {
24225 public:
24226
24227     class ScopedElement {
24228     public:
24229         ScopedElement( XmlWriter* writer, XmlFormatting fmt );
24230
24231         ScopedElement( ScopedElement&& other ) noexcept;
24232         ScopedElement& operator=( ScopedElement&& other ) noexcept;
24233
24234         ~ScopedElement();
24235
24236         ScopedElement& writeText( std::string const& text, XmlFormatting fmt =
24237             XmlFormatting::Newline | XmlFormatting::Indent );
24238
24239         template<typename T>
24240         ScopedElement& writeAttribute( std::string const& name, T const& attribute ) {
24241             m_writer->writeAttribute( name, attribute );
24242             return *this;
24243         }
24244
24245     private:
24246         mutable XmlWriter* m_writer = nullptr;
24247         XmlFormatting m_fmt;
24248     };
24249
24250     XmlWriter( std::ostream& os = Catch::cout() );
24251     ~XmlWriter();
24252
24253     XmlWriter( XmlWriter const& ) = delete;
24254     XmlWriter& operator=( XmlWriter const& ) = delete;
24255
24256     XmlWriter& startElement( std::string const& name, XmlFormatting fmt = XmlFormatting::Newline |
24257         XmlFormatting::Indent);
24258
24259     ScopedElement scopedElement( std::string const& name, XmlFormatting fmt =
24260         XmlFormatting::Newline | XmlFormatting::Indent);
24261
24262     XmlWriter& endElement(XmlFormatting fmt = XmlFormatting::Newline | XmlFormatting::Indent);
24263
24264     XmlWriter& writeAttribute( std::string const& name, std::string const& attribute );
24265
24266     XmlWriter& writeAttribute( std::string const& name, bool attribute );
24267
24268     template<typename T>
24269     XmlWriter& writeAttribute( std::string const& name, T const& attribute ) {
24270         ReusableStringStream rss;
24271         rss << attribute;
24272         return writeAttribute( name, rss.str() );
24273     }
24274
24275     XmlWriter& writeText( std::string const& text, XmlFormatting fmt = XmlFormatting::Newline |
24276         XmlFormatting::Indent);
24277
24278     XmlWriter& writeComment( std::string const& text, XmlFormatting fmt = XmlFormatting::Newline |
24279         XmlFormatting::Indent);
24280
24281     void writeStylesheetRef( std::string const& url );
24282
24283     XmlWriter& writeBlankLine();
24284
24285     void ensureTagClosed();
24286
24287 private:
24288     void applyFormatting(XmlFormatting fmt);
24289
24290     void writeDeclaration();
24291
24292     void newlineIfNecessary();
24293
24294     bool m_tagIsOpen = false;
24295     bool m_needsNewline = false;
24296     std::vector<std::string> m_tags;

```

```

24293         std::string m_indent;
24294         std::ostream& m_os;
24295     };
24296
24297 }
24298
24299 // end catch_xmlwriter.h
24300 namespace Catch {
24301
24302     class JunitReporter : public CumulativeReporterBase<JunitReporter> {
24303     public:
24304         JunitReporter(ReporterConfig const& _config);
24305
24306         ~JunitReporter() override;
24307
24308         static std::string getDescription();
24309
24310         void noMatchingTestCases(std::string const& /*spec*/) override;
24311
24312         void testRunStarting(TestRunInfo const& runInfo) override;
24313
24314         void testGroupStarting(GroupInfo const& groupInfo) override;
24315
24316         void testCaseStarting(TestCaseInfo const& testCaseInfo) override;
24317         bool assertionEnded(AssertionStats const& assertionStats) override;
24318
24319         void testCaseEnded(TestCaseStats const& testCaseStats) override;
24320
24321         void testGroupEnded(TestGroupStats const& testGroupStats) override;
24322
24323         void testRunEndedCumulative() override;
24324
24325         void writeGroup(TestGroupNode const& groupNode, double suiteTime);
24326
24327         void writeTestCase(TestCaseNode const& testCaseNode);
24328
24329         void writeSection( std::string const& className,
24330                           std::string const& rootName,
24331                           SectionNode const& sectionNode,
24332                           bool testOkToFail );
24333
24334         void writeAssertions(SectionNode const& sectionNode);
24335         void writeAssertion(AssertionStats const& stats);
24336
24337         XmlWriter xml;
24338         Timer suiteTimer;
24339         std::string stdOutForSuite;
24340         std::string stdErrForSuite;
24341         unsigned int unexpectedExceptions = 0;
24342         bool m_okToFail = false;
24343     };
24344
24345 } // end namespace Catch
24346
24347 // end catch_reporter_junit.h
24348 // start catch_reporter_xml.h
24349
24350 namespace Catch {
24351     class XmlReporter : public StreamingReporterBase<XmlReporter> {
24352     public:
24353         XmlReporter(ReporterConfig const& _config);
24354
24355         ~XmlReporter() override;
24356
24357         static std::string getDescription();
24358
24359         virtual std::string getStylesheetRef() const;
24360
24361         void writeSourceInfo(SourceLineInfo const& sourceInfo);
24362
24363     public: // StreamingReporterBase
24364
24365         void noMatchingTestCases(std::string const& s) override;
24366
24367         void testRunStarting(TestRunInfo const& testInfo) override;
24368
24369         void testGroupStarting(GroupInfo const& groupInfo) override;
24370
24371         void testCaseStarting(TestCaseInfo const& testInfo) override;
24372
24373         void sectionStarting(SectionInfo const& sectionInfo) override;
24374
24375         void assertionStarting(AssertionInfo const&) override;
24376
24377         bool assertionEnded(AssertionStats const& assertionStats) override;
24378
24379         void sectionEnded(SectionStats const& sectionStats) override;

```

```

24380
24381     void testCaseEnded(TestCaseStats const& testCaseStats) override;
24382
24383     void testGroupEnded(TestGroupStats const& testGroupStats) override;
24384
24385     void testRunEnded(TestRunStats const& testRunStats) override;
24386
24387 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
24388     void benchmarkPreparing(std::string const& name) override;
24389     void benchmarkStarting(BenchmarkInfo const&) override;
24390     void benchmarkEnded(BenchmarkStats<> const&) override;
24391     void benchmarkFailed(std::string const&) override;
24392 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
24393
24394     private:
24395         Timer m_testCaseTimer;
24396         XmlWriter m_xml;
24397         int m_sectionDepth = 0;
24398     };
24399
24400 } // end namespace Catch
24401
24402 // end catch_reporter_xml.h
24403
24404 // end catch_external_interfaces.h
24405 #endif
24406
24407 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
24408 // start catch_benchmarking_all.hpp
24409
24410 // A proxy header that includes all of the benchmarking headers to allow
24411 // concise include of the benchmarking features. You should prefer the
24412 // individual includes in standard use.
24413
24414 // start catch_benchmark.hpp
24415
24416 // Benchmark
24417
24418 // start catch_chronometer.hpp
24419
24420 // User-facing chronometer
24421
24422
24423 // start catch_clock.hpp
24424
24425 // Clocks
24426
24427 #include <chrono>
24428 #include <ratio>
24429
24430 namespace Catch {
24431     namespace Benchmark {
24432         template <typename Clock>
24433         using ClockDuration = typename Clock::duration;
24434         template <typename Clock>
24435         using FloatDuration = std::chrono::duration<double, typename Clock::period>;
24436
24437         template <typename Clock>
24438         using TimePoint = typename Clock::time_point;
24439
24440         using default_clock = std::chrono::steady_clock;
24441
24442         template <typename Clock>
24443         struct now {
24444             TimePoint<Clock> operator()() const {
24445                 return Clock::now();
24446             }
24447         };
24448     };
24449
24450     using fp_seconds = std::chrono::duration<double, std::ratio<1>;
24451 } // namespace Benchmark
24452 } // namespace Catch
24453
24454 // end catch_clock.hpp
24455 // start catch_optimizer.hpp
24456
24457 // Hinting the optimizer
24458
24459
24460 #if defined(_MSC_VER)
24461 #    include <atomic> // atomic_thread_fence
24462 #endif
24463
24464 namespace Catch {
24465     namespace Benchmark {
24466 #if defined(__GNUC__) || defined(__clang__)

```

```

24467     template <typename T>
24468     inline void keep_memory(T* p) {
24469         asm volatile("" : : "g"(p) : "memory");
24470     }
24471     inline void keep_memory() {
24472         asm volatile("" : : : "memory");
24473     }
24474
24475     namespace Detail {
24476         inline void optimizer_barrier() { keep_memory(); }
24477     } // namespace Detail
24478 #elif defined(_MSC_VER)
24479
24480 #pragma optimize("", off)
24481     template <typename T>
24482     inline void keep_memory(T* p) {
24483         // thanks @milleniumbug
24484         *reinterpret_cast<char volatile*>(p) = *reinterpret_cast<char const volatile*>(p);
24485     }
24486     // TODO equivalent keep_memory()
24487 #pragma optimize("", on)
24488
24489     namespace Detail {
24490         inline void optimizer_barrier() {
24491             std::atomic_thread_fence(std::memory_order_seq_cst);
24492         }
24493     } // namespace Detail
24494
24495 #endif
24496
24497     template <typename T>
24498     inline void deoptimize_value(T&& x) {
24499         keep_memory(&x);
24500     }
24501
24502     template <typename Fn, typename... Args>
24503     inline auto invoke_deoptimized(Fn&& fn, Args&&... args) -> typename
std::enable_if<!std::is_same<void, decltype(fn(args...))>::value>::type {
24504     deoptimize_value(std::forward<Fn>(fn) (std::forward<Args...>(args...)));
24505 }
24506
24507     template <typename Fn, typename... Args>
24508     inline auto invoke_deoptimized(Fn&& fn, Args&&... args) -> typename
std::enable_if<std::is_same<void, decltype(fn(args...))>::value>::type {
24509     std::forward<Fn>(fn) (std::forward<Args...>(args...));
24510 }
24511 } // namespace Benchmark
24512 } // namespace Catch
24513
24514 // end catch_optimizer.hpp
24515 // start catch_complete_invoke.hpp
24516
24517 // Invoke with a special case for void
24518
24519
24520 #include <type_traits>
24521 #include <utility>
24522
24523 namespace Catch {
24524     namespace Benchmark {
24525         namespace Detail {
24526             template <typename T>
24527             struct CompleteType { using type = T; };
24528             template <>
24529             struct CompleteType<void> { struct type {}; };
24530
24531             template <typename T>
24532             using CompleteType_t = typename CompleteType<T>::type;
24533
24534             template <typename Result>
24535             struct CompleteInvoker {
24536                 template <typename Fun, typename... Args>
24537                 static Result invoke(Fun&& fun, Args&&... args) {
24538                     return std::forward<Fun>(fun) (std::forward<Args>(args)...);
24539                 }
24540             };
24541             template <>
24542             struct CompleteInvoker<void> {
24543                 template <typename Fun, typename... Args>
24544                 static CompleteType_t<void> invoke(Fun&& fun, Args&&... args) {
24545                     std::forward<Fun>(fun) (std::forward<Args>(args)...);
24546                     return {};
24547                 }
24548             };
24549
24550             // invoke and not return void :(
24551             template <typename Fun, typename... Args>

```

```

24552         CompleteType_t<FunctionReturnType<Fun, Args...> complete_invoke(Fun&& fun, Args&&... args)
24553     {
24554         return CompleteInvoker<FunctionReturnType<Fun,
24555     Args...>::invoke(std::forward<Fun>(fun), std::forward<Args>(args)...);
24556     }
24557     const std::string benchmarkErrorMsg = "a benchmark failed to run successfully";
24558     } // namespace Detail
24559     template <typename Fun>
24560     Detail::CompleteType_t<FunctionReturnType<Fun> user_code(Fun&& fun) {
24561         CATCH_TRY{
24562             return Detail::complete_invoke(std::forward<Fun>(fun));
24563         } CATCH_CATCH_ALL{
24564             getResultCapture().benchmarkFailed(translateActiveException());
24565             CATCH_RUNTIME_ERROR(Detail::benchmarkErrorMsg);
24566         }
24567     }
24568     } // namespace Benchmark
24569 } // namespace Catch
24570
24571 // end catch_complete_invoke.hpp
24572 namespace Catch {
24573     namespace Benchmark {
24574         namespace Detail {
24575             struct ChronometerConcept {
24576                 virtual void start() = 0;
24577                 virtual void finish() = 0;
24578                 virtual ~ChronometerConcept() = default;
24579             };
24580             template <typename Clock>
24581             struct ChronometerModel final : public ChronometerConcept {
24582                 void start() override { started = Clock::now(); }
24583                 void finish() override { finished = Clock::now(); }
24584
24585                 ClockDuration<Clock> elapsed() const { return finished - started; }
24586
24587                 TimePoint<Clock> started;
24588                 TimePoint<Clock> finished;
24589             };
24590         } // namespace Detail
24591
24592         struct Chronometer {
24593         public:
24594             template <typename Fun>
24595             void measure(Fun&& fun) { measure(std::forward<Fun>(fun), is_callable<Fun(int)>()); }
24596
24597             int runs() const { return k; }
24598
24599             Chronometer(Detail::ChronometerConcept& meter, int k)
24600                 : impl(&meter)
24601                 , k(k) {}
24602
24603         private:
24604             template <typename Fun>
24605             void measure(Fun&& fun, std::false_type) {
24606                 measure([&fun](int) { return fun(); }, std::true_type());
24607             }
24608
24609             template <typename Fun>
24610             void measure(Fun&& fun, std::true_type) {
24611                 Detail::optimizer_barrier();
24612                 impl->start();
24613                 for (int i = 0; i < k; ++i) invoke_deoptimized(fun, i);
24614                 impl->finish();
24615                 Detail::optimizer_barrier();
24616             }
24617
24618             Detail::ChronometerConcept* impl;
24619             int k;
24620         };
24621     } // namespace Benchmark
24622 } // namespace Catch
24623
24624 // end catch_chronometer.hpp
24625 // start catch_environment.hpp
24626
24627 // Environment information
24628
24629 namespace Catch {
24630     namespace Benchmark {
24631         namespace Detail {
24632             template <typename Duration>
24633             struct EnvironmentEstimate {
24634                 Duration mean;
24635                 OutlierClassification outliers;
24636             };

```

```

24637         template <typename Duration2>
24638         operator EnvironmentEstimate<Duration2>() const {
24639             return { mean, outliers };
24640         }
24641     };
24642     template <typename Clock>
24643     struct Environment {
24644         using clock_type = Clock;
24645         EnvironmentEstimate<FloatDuration<Clock>> clock_resolution;
24646         EnvironmentEstimate<FloatDuration<Clock>> clock_cost;
24647     };
24648     } // namespace Benchmark
24649 } // namespace Catch
24650
24651 // end catch_environment.hpp
24652 // start catch_execution_plan.hpp
24653
24654 // Execution plan
24655
24656
24657 // start catch_benchmark_function.hpp
24658
24659 // Dumb std::function implementation for consistent call overhead
24660
24661
24662 #include <cassert>
24663 #include <type_traits>
24664 #include <utility>
24665 #include <memory>
24666
24667 namespace Catch {
24668     namespace Benchmark {
24669         namespace Detail {
24670             template <typename T>
24671             using Decay = typename std::decay<T>::type;
24672             template <typename T, typename U>
24673             struct is_related
24674                 : std::is_same<Decay<T>, Decay<U>> {};
24675
24676             struct BenchmarkFunction {
24677             private:
24678                 struct callable {
24679                     virtual void call(Chronometer meter) const = 0;
24680                     virtual callable* clone() const = 0;
24681                     virtual ~callable() = default;
24682                 };
24683                 template <typename Fun>
24684                 struct model : public callable {
24685                     model(Fun&& fun) : fun(std::move(fun)) {}
24686                     model(Fun const& fun) : fun(fun) {}
24687
24688                     model<Fun>* clone() const override { return new model<Fun>(*this); }
24689
24690                     void call(Chronometer meter) const override {
24691                         call(meter, is_callable<Fun(Chronometer)>());
24692                     }
24693                     void call(Chronometer meter, std::true_type) const {
24694                         fun(meter);
24695                     }
24696                     void call(Chronometer meter, std::false_type) const {
24697                         meter.measure(fun);
24698                     }
24699
24700                     Fun fun;
24701                 };
24702
24703             public:
24704                 struct do_nothing { void operator()() const {} };
24705
24706                 template <typename T>
24707                 BenchmarkFunction(model<T>* c) : f(c) {}
24708
24709             public:
24710                 BenchmarkFunction()
24711                     : f(new model<do_nothing>{ {} }) {}
24712
24713                 template <typename Fun,
24714                     typename std::enable_if<!is_related<Fun, BenchmarkFunction>::value, int>::type =
24715                     0>
24716                     BenchmarkFunction(Fun&& fun)
24717                         : f(new model<typename std::decay<Fun>::type>(std::forward<Fun>(fun))) {}
24718
24719                 BenchmarkFunction(BenchmarkFunction&& that)
24720                     : f(std::move(that.f)) {}
24721
24722                 BenchmarkFunction(BenchmarkFunction const& that)
24723                     : f(that.f->clone()) {}
24724
24725
24726
24727
24728
24729

```

```

24730         BenchmarkFunction& operator=(BenchmarkFunction&& that) {
24731             f = std::move(that.f);
24732             return *this;
24733         }
24734
24735         BenchmarkFunction& operator=(BenchmarkFunction const& that) {
24736             f.reset(that.f->clone());
24737             return *this;
24738         }
24739
24740         void operator()(Chronometer meter) const { f->call(meter); }
24741
24742     private:
24743         std::unique_ptr<callable> f;
24744     };
24745 } // namespace Detail
24746 } // namespace Benchmark
24747 } // namespace Catch
24748
24749 // end catch_benchmark_function.hpp
24750 // start catch_repeat.hpp
24751
24752 // repeat algorithm
24753
24754 #include <type_traits>
24755 #include <utility>
24756
24757 namespace Catch {
24758     namespace Benchmark {
24759         namespace Detail {
24760             template <typename Fun>
24761             struct repeater {
24762                 void operator()(int k) const {
24763                     for (int i = 0; i < k; ++i) {
24764                         fun();
24765                     }
24766                 }
24767             }
24768             Fun fun;
24769         };
24770         template <typename Fun>
24771         repeater<typename std::decay<Fun>::type> repeat(Fun&& fun) {
24772             return { std::forward<Fun>(fun) };
24773         }
24774     } // namespace Detail
24775 } // namespace Benchmark
24776 } // namespace Catch
24777
24778 // end catch_repeat.hpp
24779 // start catch_run_for_at_least.hpp
24780
24781 // Run a function for a minimum amount of time
24782
24783 // start catch_measure.hpp
24784
24785 // Measure
24786
24787 // start catch_timing.hpp
24788
24789 // Timing
24790
24791 #include <tuple>
24792 #include <type_traits>
24793
24794 namespace Catch {
24795     namespace Benchmark {
24796         template <typename Duration, typename Result>
24797         struct Timing {
24798             Duration elapsed;
24799             Result result;
24800             int iterations;
24801         };
24802         template <typename Clock, typename Func, typename... Args>
24803         using TimingOf = Timing<ClockDuration<Clock>, Detail::CompleteType_t<FunctionReturnType<Func,
24804             Args...>>>;
24805     } // namespace Benchmark
24806 } // namespace Catch
24807
24808 // end catch_timing.hpp
24809 #include <utility>
24810
24811 namespace Catch {
24812     namespace Benchmark {
24813         namespace Detail {

```

```

24816         template <typename Clock, typename Fun, typename... Args>
24817         TimingOf<Clock, Fun, Args...> measure(Fun&& fun, Args&&... args) {
24818             auto start = Clock::now();
24819             auto&& r = Detail::complete_invoke(fun, std::forward<Args>(args)...);
24820             auto end = Clock::now();
24821             auto delta = end - start;
24822             return { delta, std::forward<decltype(r)>(r), 1 };
24823         }
24824     } // namespace Detail
24825 } // namespace Benchmark
24826 } // namespace Catch
24827
24828 // end catch_measure.hpp
24829 #include <utility>
24830 #include <type_traits>
24831
24832 namespace Catch {
24833     namespace Benchmark {
24834         namespace Detail {
24835             template <typename Clock, typename Fun>
24836             TimingOf<Clock, Fun, int> measure_one(Fun&& fun, int iters, std::false_type) {
24837                 return Detail::measure<Clock>(fun, iters);
24838             }
24839             template <typename Clock, typename Fun>
24840             TimingOf<Clock, Fun, Chronometer> measure_one(Fun&& fun, int iters, std::true_type) {
24841                 Detail::ChronometerModel<Clock> meter;
24842                 auto&& result = Detail::complete_invoke(fun, Chronometer(meter, iters));
24843
24844                 return { meter.elapsed(), std::move(result), iters };
24845             }
24846
24847             template <typename Clock, typename Fun>
24848             using run_for_at_least_argument_t = typename
24849             std::conditional<is_callable<Fun(Chronometer)>::value, Chronometer, int>::type;
24850
24851             struct optimized_away_error : std::exception {
24852                 const char* what() const noexcept override {
24853                     return "could not measure benchmark, maybe it was optimized away";
24854                 }
24855             };
24856
24857             template <typename Clock, typename Fun>
24858             TimingOf<Clock, Fun, run_for_at_least_argument_t<Clock, Fun>
24859             run_for_at_least(ClockDuration<Clock> how_long, int seed, Fun&& fun) {
24860                 auto iters = seed;
24861                 while (iters < (1 << 30)) {
24862                     auto&& Timing = measure_one<Clock>(fun, iters, is_callable<Fun(Chronometer)>());
24863                     if (Timing.elapsed >= how_long) {
24864                         return { Timing.elapsed, std::move(Timing.result), iters };
24865                     }
24866                     iters *= 2;
24867                 }
24868                 Catch::throw_exception(optimized_away_error{});
24869             } // namespace Detail
24870         } // namespace Benchmark
24871     } // namespace Catch
24872
24873 // end catch_run_for_at_least.hpp
24874 #include <algorithm>
24875 #include <iterator>
24876
24877 namespace Catch {
24878     namespace Benchmark {
24879         template <typename Duration>
24880         struct ExecutionPlan {
24881             int iterations_per_sample;
24882             Duration estimated_duration;
24883             Detail::BenchmarkFunction benchmark;
24884             Duration warmup_time;
24885             int warmup_iterations;
24886
24887             template <typename Duration2>
24888             operator ExecutionPlan<Duration2>() const {
24889                 return { iterations_per_sample, estimated_duration, benchmark, warmup_time,
24890                     warmup_iterations };
24891             }
24892
24893             template <typename Clock>
24894             std::vector<FloatDuration<Clock>> run(const IConfig &cfg, Environment<FloatDuration<Clock>>
24895             env) const {
24896                 // warmup a bit
24897                 Detail::run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(warmup_time),
24898                 warmup_iterations, Detail::repeat(now<Clock>{}));
24899             }
24900         }

```



```

24897         std::vector<FloatDuration<Clock>> times;
24898         times.reserve(cfg.benchmarkSamples());
24899         std::generate_n(std::back_inserter(times), cfg.benchmarkSamples(), [this, env] {
24900             Detail::ChronometerModel<Clock> model;
24901             this->benchmark(Chronometer(model, iterations_per_sample));
24902             auto sample_time = model.elapsed() - env.clock_cost.mean;
24903             if (sample_time < FloatDuration<Clock>::zero()) sample_time =
FloatDuration<Clock>::zero();
24904             return sample_time / iterations_per_sample;
24905         });
24906         return times;
24907     }
24908 };
24909 } // namespace Benchmark
24910 } // namespace Catch
24911
24912 // end catch_execution_plan.hpp
24913 // start catch_estimate_clock.hpp
24914
24915 // Environment measurement
24916
24917
24918 // start catch_stats.hpp
24919
24920 // Statistical analysis tools
24921
24922
24923 #include <algorithm>
24924 #include <functional>
24925 #include <vector>
24926 #include <iterator>
24927 #include <numeric>
24928 #include <tuple>
24929 #include <cmath>
24930 #include <utility>
24931 #include <cstdint>
24932 #include <random>
24933
24934 namespace Catch {
24935     namespace Benchmark {
24936         namespace Detail {
24937             using sample = std::vector<double>;
24938
24939             double weighted_average_quantile(int k, int q, std::vector<double>::iterator first,
std::vector<double>::iterator last);
24940
24941             template <typename Iterator>
24942             OutlierClassification classify_outliers(Iterator first, Iterator last) {
24943                 std::vector<double> copy(first, last);
24944
24945                 auto q1 = weighted_average_quantile(1, 4, copy.begin(), copy.end());
24946                 auto q3 = weighted_average_quantile(3, 4, copy.begin(), copy.end());
24947                 auto iqr = q3 - q1;
24948                 auto los = q1 - (iqr * 3.);
24949                 auto lom = q1 - (iqr * 1.5);
24950                 auto him = q3 + (iqr * 1.5);
24951                 auto his = q3 + (iqr * 3.);
24952
24953                 OutlierClassification o;
24954                 for (; first != last; ++first) {
24955                     auto&& t = *first;
24956                     if (t < los) ++o.low_severe;
24957                     else if (t < lom) ++o.low_mild;
24958                     else if (t > his) ++o.high_severe;
24959                     else if (t > him) ++o.high_mild;
24960                     ++o.samples_seen;
24961                 }
24962                 return o;
24963             }
24964
24965             template <typename Iterator>
24966             double mean(Iterator first, Iterator last) {
24967                 auto count = last - first;
24968                 double sum = std::accumulate(first, last, 0.);
24969                 return sum / count;
24970             }
24971
24972             template <typename URng, typename Iterator, typename Estimator>
24973             sample resample(URng& rng, int resamples, Iterator first, Iterator last, Estimator&
estimator) {
24974                 auto n = last - first;
24975                 std::uniform_int_distribution<decltype(n)> dist(0, n - 1);
24976
24977                 sample out;
24978                 out.reserve(resamples);
24979                 std::generate_n(std::back_inserter(out), resamples, [n, first, &estimator, &dist,
&rng] {

```

```

24980         std::vector<double> resampled;
24981         resampled.reserve(n);
24982         std::generate_n(std::back_inserter(resampled), n, [first, &dist, &rng] { return
first[dist(rng)]; });
24983         return estimator(resampled.begin(), resampled.end());
24984     });
24985     std::sort(out.begin(), out.end());
24986     return out;
24987 }
24988
24989 template <typename Estimator, typename Iterator>
24990 sample jackknife(Estimator&& estimator, Iterator first, Iterator last) {
24991     auto n = last - first;
24992     auto second = std::next(first);
24993     sample results;
24994     results.reserve(n);
24995
24996     for (auto it = first; it != last; ++it) {
24997         std::iter_swap(it, first);
24998         results.push_back(estimator(second, last));
24999     }
25000
25001     return results;
25002 }
25003
25004 inline double normal_cdf(double x) {
25005     return std::erfc(-x / std::sqrt(2.0)) / 2.0;
25006 }
25007
25008 double erfc_inv(double x);
25009
25010 double normal_quantile(double p);
25011
25012 template <typename Iterator, typename Estimator>
25013 Estimate<double> bootstrap(double confidence_level, Iterator first, Iterator last, sample
const& resample, Estimator&& estimator) {
25014     auto n_samples = last - first;
25015
25016     double point = estimator(first, last);
25017     // Degenerate case with a single sample
25018     if (n_samples == 1) return { point, point, point, confidence_level };
25019
25020     sample jack = jackknife(estimator, first, last);
25021     double jack_mean = mean(jack.begin(), jack.end());
25022     double sum_squares, sum_cubes;
25023     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> {
25024         auto d = jack_mean - x;
25025         auto d2 = d * d;
25026         auto d3 = d2 * d;
25027         return { sqcb.first + d2, sqcb.second + d3 };
25028     });
25029
25030     double accel = sum_cubes / (6 * std::pow(sum_squares, 1.5));
25031     int n = static_cast<int>(resample.size());
25032     double prob_n = std::count_if(resample.begin(), resample.end(), [point](double x) {
return x < point; }) / (double)n;
25033     // degenerate case with uniform samples
25034     if (prob_n == 0) return { point, point, point, confidence_level };
25035
25036     double bias = normal_quantile(prob_n);
25037     double z1 = normal_quantile((1. - confidence_level) / 2.);
25038
25039     auto cumn = [n](double x) -> int {
25040         return std::lround(normal_cdf(x) * n); };
25041     auto a = [bias, accel](double b) { return bias + b / (1. - accel * b); };
25042     double b1 = bias + z1;
25043     double b2 = bias - z1;
25044     double a1 = a(b1);
25045     double a2 = a(b2);
25046     auto lo = (std::max)(cumn(a1), 0);
25047     auto hi = (std::min)(cumn(a2), n - 1);
25048
25049     return { point, resample[lo], resample[hi], confidence_level };
25050 }
25051
25052 double outlier_variance(Estimate<double> mean, Estimate<double> stddev, int n);
25053
25054 struct bootstrap_analysis {
25055     Estimate<double> mean;
25056     Estimate<double> standard_deviation;
25057     double outlier_variance;
25058 };
25059
25060 bootstrap_analysis analyse_samples(double confidence_level, int n_resamples,
std::vector<double>::iterator first, std::vector<double>::iterator last);

```

```

25061         } // namespace Detail
25062     } // namespace Benchmark
25063 } // namespace Catch
25064
25065 // end catch_stats.hpp
25066 #include <algorithm>
25067 #include <iterator>
25068 #include <tuple>
25069 #include <vector>
25070 #include <cmath>
25071
25072 namespace Catch {
25073     namespace Benchmark {
25074         namespace Detail {
25075             template <typename Clock>
25076             std::vector<double> resolution(int k) {
25077                 std::vector<TimePoint<Clock>> times;
25078                 times.reserve(k + 1);
25079                 std::generate_n(std::back_inserter(times), k + 1, now<Clock>{});
25080
25081                 std::vector<double> deltas;
25082                 deltas.reserve(k);
25083                 std::transform(std::next(times.begin()), times.end(), times.begin(),
25084                     std::back_inserter(deltas),
25085                     [](TimePoint<Clock> a, TimePoint<Clock> b) { return static_cast<double>((a -
25086 b).count()); });
25087
25088                 return deltas;
25089             }
25090
25091             const auto warmup_iterations = 10000;
25092             const auto warmup_time = std::chrono::milliseconds(100);
25093             const auto minimum_ticks = 1000;
25094             const auto warmup_seed = 10000;
25095             const auto clock_resolution_estimation_time = std::chrono::milliseconds(500);
25096             const auto clock_cost_estimation_time_limit = std::chrono::seconds(1);
25097             const auto clock_cost_estimation_tick_limit = 100000;
25098             const auto clock_cost_estimation_time = std::chrono::milliseconds(10);
25099             const auto clock_cost_estimation_iterations = 10000;
25100
25101             template <typename Clock>
25102             int warmup() {
25103                 return
25104                     run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(warmup_time), warmup_seed,
25105                     &resolution<Clock>)
25106                         .iterations;
25107             }
25108
25109             template <typename Clock>
25110             EnvironmentEstimate<FloatDuration<Clock>> estimate_clock_resolution(int iterations) {
25111                 auto r =
25112                     run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(clock_resolution_estimation_time),
25113                     iterations, &resolution<Clock>)
25114                         .result;
25115                 return
25116                     FloatDuration<Clock>(mean(r.begin(), r.end())),
25117                     classify_outliers(r.begin(), r.end()),
25118                     };
25119             }
25120
25121             template <typename Clock>
25122             EnvironmentEstimate<FloatDuration<Clock>> estimate_clock_cost(FloatDuration<Clock>
25123 resolution) {
25124                 auto time_limit = (std::min)(
25125                     resolution * clock_cost_estimation_tick_limit,
25126                     FloatDuration<Clock>(clock_cost_estimation_time_limit));
25127                 auto time_clock = [](int k) {
25128                     return Detail::measure<Clock>([k] {
25129                         for (int i = 0; i < k; ++i) {
25130                             volatile auto ignored = Clock::now();
25131                             (void)ignored;
25132                         }
25133                     }).elapsed;
25134                 };
25135                 time_clock(1);
25136                 int iters = clock_cost_estimation_iterations;
25137                 auto&& r =
25138                     run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(clock_cost_estimation_time),
25139                     iters, time_clock);
25140                 std::vector<double> times;
25141                 int nsamples = static_cast<int>(std::ceil(time_limit / r.elapsed));
25142                 times.reserve(nsamples);
25143                 std::generate_n(std::back_inserter(times), nsamples, [time_clock, &r] {
25144                     return static_cast<double>((time_clock(r.iterations) / r.iterations).count());
25145                 });
25146                 return
25147                     FloatDuration<Clock>(mean(times.begin(), times.end())),
25148                     classify_outliers(times.begin(), times.end()),
25149                     };
25150             }

```

```

25140     }
25141
25142     template <typename Clock>
25143     Environment<FloatDuration<Clock>> measure_environment() {
25144         static Environment<FloatDuration<Clock>>* env = nullptr;
25145         if (env) {
25146             return *env;
25147         }
25148
25149         auto iters = Detail::warmup<Clock>();
25150         auto resolution = Detail::estimate_clock_resolution<Clock>(iters);
25151         auto cost = Detail::estimate_clock_cost<Clock>(resolution.mean);
25152
25153         env = new Environment<FloatDuration<Clock>>{ resolution, cost };
25154         return *env;
25155     }
25156 } // namespace Detail
25157 } // namespace Benchmark
25158 } // namespace Catch
25159
25160 // end catch_estimate_clock.hpp
25161 // start catch_analyse.hpp
25162
25163 // Run and analyse one benchmark
25164
25165
25166 // start catch_sample_analysis.hpp
25167
25168 // Benchmark results
25169
25170
25171 #include <algorithm>
25172 #include <vector>
25173 #include <string>
25174 #include <iterator>
25175
25176 namespace Catch {
25177     namespace Benchmark {
25178         template <typename Duration>
25179         struct SampleAnalysis {
25180             std::vector<Duration> samples;
25181             Estimate<Duration> mean;
25182             Estimate<Duration> standard_deviation;
25183             OutlierClassification outliers;
25184             double outlier_variance;
25185
25186             template <typename Duration2>
25187             operator SampleAnalysis<Duration2>() const {
25188                 std::vector<Duration2> samples2;
25189                 samples2.reserve(samples.size());
25190                 std::transform(samples.begin(), samples.end(), std::back_inserter(samples2),
25191                     [](Duration d) { return Duration2(d); });
25192                 return {
25193                     std::move(samples2),
25194                     mean,
25195                     standard_deviation,
25196                     outliers,
25197                     outlier_variance,
25198                 };
25199             }
25200         } // namespace Benchmark
25201     } // namespace Catch
25202
25203 // end catch_sample_analysis.hpp
25204 #include <algorithm>
25205 #include <iterator>
25206 #include <vector>
25207
25208 namespace Catch {
25209     namespace Benchmark {
25210         namespace Detail {
25211             template <typename Duration, typename Iterator>
25212             SampleAnalysis<Duration> analyse(const IConfig &cfg, Environment<Duration>, Iterator
25213 first, Iterator last) {
25214                 if (!cfg.benchmarkNoAnalysis()) {
25215                     std::vector<double> samples;
25216                     samples.reserve(last - first);
25217                     std::transform(first, last, std::back_inserter(samples), [](Duration d) { return
25218 d.count(); });
25219                     auto analysis =
25220 Catch::Benchmark::Detail::analyse_samples(cfg.benchmarkConfidenceInterval(), cfg.benchmarkResamples(),
25221 samples.begin(), samples.end());
25222                     auto outliers = Catch::Benchmark::Detail::classify_outliers(samples.begin(),
25223 samples.end());
25224

```

```

25221         auto wrap_estimate = [](Estimate<double> e) {
25222             return Estimate<Duration> {
25223                 Duration(e.point),
25224                 Duration(e.lower_bound),
25225                 Duration(e.upper_bound),
25226                 e.confidence_interval,
25227             };
25228         };
25229         std::vector<Duration> samples2;
25230         samples2.reserve(samples.size());
25231         std::transform(samples.begin(), samples.end(), std::back_inserter(samples2),
25232             [](double d) { return Duration(d); });
25233         return {
25234             std::move(samples2),
25235             wrap_estimate(analysis.mean),
25236             wrap_estimate(analysis.standard_deviation),
25237             outliers,
25238             analysis.outlier_variance,
25239         };
25240     } else {
25241         std::vector<Duration> samples;
25242         samples.reserve(last - first);
25243
25244         Duration mean = Duration(0);
25245         int i = 0;
25246         for (auto it = first; it < last; ++it, ++i) {
25247             samples.push_back(Duration(*it));
25248             mean += Duration(*it);
25249         }
25250         mean /= i;
25251
25252         return {
25253             std::move(samples),
25254             Estimate<Duration>{mean, mean, mean, 0.0},
25255             Estimate<Duration>{Duration(0), Duration(0), Duration(0), 0.0},
25256             OutlierClassification{},
25257             0.0
25258         };
25259     }
25260 } // namespace Detail
25261 } // namespace Benchmark
25262 } // namespace Catch
25263
25264 // end catch_analyse.hpp
25265 #include <algorithm>
25266 #include <functional>
25267 #include <string>
25268 #include <vector>
25269 #include <cmath>
25270
25271 namespace Catch {
25272     namespace Benchmark {
25273         struct Benchmark {
25274             Benchmark(std::string &&name)
25275                 : name(std::move(name)) {}
25276
25277             template <class FUN>
25278             Benchmark(std::string &&name, FUN &&func)
25279                 : fun(std::move(func)), name(std::move(name)) {}
25280
25281             template <typename Clock>
25282             ExecutionPlan<FloatDuration<Clock>> prepare(const IConfig &cfg,
25283                 Environment<FloatDuration<Clock>> env) const {
25284                 auto min_time = env.clock_resolution.mean * Detail::minimum_ticks;
25285                 auto run_time = std::max(min_time,
25286                     std::chrono::duration_cast<decltype(min_time)>(cfg.benchmarkWarmupTime()));
25287                 auto&& test =
25288                     Detail::run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(run_time), 1, fun);
25289                 int new_iters = static_cast<int>(std::ceil(min_time * test.iterations /
25290                     test.elapsed));
25291                 return { new_iters, test.elapsed / test.iterations * new_iters *
25292                     cfg.benchmarkSamples(), fun,
25293                     std::chrono::duration_cast<FloatDuration<Clock>>(cfg.benchmarkWarmupTime()), Detail::warmup_iterations
25294                 };
25295             }
25296
25297             template <typename Clock = default_clock>
25298             void run() {
25299                 IConfigPtr cfg = getCurrentContext().getConfig();
25300
25301                 auto env = Detail::measure_environment<Clock>();
25302
25303                 getResultCapture().benchmarkPreparing(name);
25304                 CATCH_TRY{
25305                     auto plan = user_code([&] {
25306                         return prepare<Clock>(*cfg, env);
25307                     });
25308                 }
25309             }

```

```

25300         });
25301
25302         BenchmarkInfo info {
25303             name,
25304             plan.estimated_duration.count(),
25305             plan.iterations_per_sample,
25306             cfg->benchmarkSamples(),
25307             cfg->benchmarkResamples(),
25308             env.clock_resolution.mean.count(),
25309             env.clock_cost.mean.count()
25310         };
25311
25312         getResultCapture().benchmarkStarting(info);
25313
25314         auto samples = user_code([&] {
25315             return plan.template run<Clock>(*cfg, env);
25316         });
25317
25318         auto analysis = Detail::analyse(*cfg, env, samples.begin(), samples.end());
25319         BenchmarkStats<FloatDuration<Clock>> stats{ info, analysis.samples, analysis.mean,
analysis.standard_deviation, analysis.outliers, analysis.outlier_variance };
25320         getResultCapture().benchmarkEnded(stats);
25321
25322         } CATCH_CATCH_ALL{
25323             if (translateActiveException() != Detail::benchmarkErrorMsg) // benchmark errors
have been reported, otherwise rethrow.
25324                 std::rethrow_exception(std::current_exception());
25325         }
25326     }
25327
25328     // sets lambda to be used in fun *and* executes benchmark!
25329     template <typename Fun,
25330             typename std::enable_if<!Detail::is_related<Fun, Benchmark>::value, int>::type = 0>
25331             Benchmark & operator=(Fun func) {
25332         fun = Detail::BenchmarkFunction(func);
25333         run();
25334         return *this;
25335     }
25336
25337     explicit operator bool() {
25338         return true;
25339     }
25340
25341     private:
25342         Detail::BenchmarkFunction fun;
25343         std::string name;
25344     };
25345 }
25346 } // namespace Catch
25347
25348 #define INTERNAL_CATCH_GET_1_ARG(arg1, arg2, ...) arg1
25349 #define INTERNAL_CATCH_GET_2_ARG(arg1, arg2, ...) arg2
25350
25351 #define INTERNAL_CATCH_BENCHMARK(BenchmarkName, name, benchmarkIndex)\
25352     if( Catch::Benchmark::Benchmark BenchmarkName{name} ) \
25353         BenchmarkName = [&](int benchmarkIndex)
25354
25355 #define INTERNAL_CATCH_BENCHMARK_ADVANCED(BenchmarkName, name)\
25356     if( Catch::Benchmark::Benchmark BenchmarkName{name} ) \
25357         BenchmarkName = [&]
25358
25359 // end catch_benchmark.hpp
25360 // start catch_constructor.hpp
25361
25362 // Constructor and destructor helpers
25363
25364
25365 #include <type_traits>
25366
25367 namespace Catch {
25368     namespace Benchmark {
25369         namespace Detail {
25370             template <typename T, bool Destruct>
25371             struct ObjectStorage
25372             {
25373                 ObjectStorage() : data() {}
25374
25375                 ObjectStorage(const ObjectStorage& other)
25376                 {
25377                     new(&data) T(other.stored_object());
25378                 }
25379
25380                 ObjectStorage(ObjectStorage&& other)
25381                 {
25382                     new(&data) T(std::move(other.stored_object()));
25383                 }
25384             }

```

```

25385         ~ObjectStorage() { destruct_on_exit<T>(); }
25386
25387         template <typename... Args>
25388         void construct(Args&&... args)
25389         {
25390             new (&data) T(std::forward<Args>(args)...);
25391         }
25392
25393         template <bool AllowManualDestruction = !Destruct>
25394         typename std::enable_if<AllowManualDestruction>::type destruct()
25395         {
25396             stored_object().~T();
25397         }
25398
25399     private:
25400         // If this is a constructor benchmark, destruct the underlying object
25401         template <typename U>
25402         void destruct_on_exit(typename std::enable_if<Destruct, U>::type* = 0) {
25403             destruct<true>(); }
25404         // Otherwise, don't
25405         template <typename U>
25406         void destruct_on_exit(typename std::enable_if<!Destruct, U>::type* = 0) { }
25407
25408         T& stored_object() {
25409             return *static_cast<T*>(static_cast<void*>(&data));
25410         }
25411
25412         T const& stored_object() const {
25413             return *static_cast<T*>(static_cast<void*>(&data));
25414         }
25415
25416         struct { alignas(T) unsigned char data[sizeof(T)]; } data;
25417     };
25418
25419     template <typename T>
25420     using storage_for = Detail::ObjectStorage<T, true>;
25421
25422     template <typename T>
25423     using destructable_object = Detail::ObjectStorage<T, false>;
25424 }
25425 }
25426
25427 // end catch_constructor.hpp
25428 // end catch_benchmarking_all.hpp
25429 #endif
25430
25431 #endif // ! CATCH_CONFIG_IMPL_ONLY
25432
25433 #ifdef CATCH_IMPL
25434 // start catch_impl.hpp
25435
25436 #ifdef __clang__
25437 #pragma clang diagnostic push
25438 #pragma clang diagnostic ignored "-Wweak-vtables"
25439 #endif
25440
25441 // Keep these here for external reporters
25442 // start catch_test_case_tracker.h
25443
25444 #include <string>
25445 #include <vector>
25446 #include <memory>
25447
25448 namespace Catch {
25449     namespace TestCaseTracking {
25450
25451         struct NameAndLocation {
25452             std::string name;
25453             SourceLineInfo location;
25454
25455             NameAndLocation( std::string const& _name, SourceLineInfo const& _location );
25456             friend bool operator==(NameAndLocation const& lhs, NameAndLocation const& rhs) {
25457                 return lhs.name == rhs.name
25458                    && lhs.location == rhs.location;
25459             }
25460         };
25461
25462         class ITracker;
25463
25464         using ITrackerPtr = std::shared_ptr<ITracker>;
25465
25466         class ITracker {
25467             NameAndLocation m_nameAndLocation;
25468
25469         public:
25470             ITracker(NameAndLocation const& nameAndLoc) :

```

```

25471         m_nameAndLocation(nameAndLoc)
25472     {}
25473
25474     // static queries
25475     NameAndLocation const& nameAndLocation() const {
25476         return m_nameAndLocation;
25477     }
25478
25479     virtual ~ITracker();
25480
25481     // dynamic queries
25482     virtual bool isComplete() const = 0; // Successfully completed or failed
25483     virtual bool isSuccessfullyCompleted() const = 0;
25484     virtual bool isOpen() const = 0; // Started but not complete
25485     virtual bool hasChildren() const = 0;
25486     virtual bool hasStarted() const = 0;
25487
25488     virtual ITracker& parent() = 0;
25489
25490     // actions
25491     virtual void close() = 0; // Successfully complete
25492     virtual void fail() = 0;
25493     virtual void markAsNeedingAnotherRun() = 0;
25494
25495     virtual void addChild( ITrackerPtr const& child ) = 0;
25496     virtual ITrackerPtr findChild( NameAndLocation const& nameAndLocation ) = 0;
25497     virtual void openChild() = 0;
25498
25499     // Debug/ checking
25500     virtual bool isSectionTracker() const = 0;
25501     virtual bool isGeneratorTracker() const = 0;
25502 };
25503
25504 class TrackerContext {
25505
25506     enum RunState {
25507         NotStarted,
25508         Executing,
25509         CompletedCycle
25510     };
25511
25512     ITrackerPtr m_rootTracker;
25513     ITracker* m_currentTracker = nullptr;
25514     RunState m_runState = NotStarted;
25515
25516 public:
25517
25518     ITracker& startRun();
25519     void endRun();
25520
25521     void startCycle();
25522     void completeCycle();
25523
25524     bool completedCycle() const;
25525     ITracker& currentTracker();
25526     void setCurrentTracker( ITracker* tracker );
25527 };
25528
25529 class TrackerBase : public ITracker {
25530 protected:
25531     enum CycleState {
25532         NotStarted,
25533         Executing,
25534         ExecutingChildren,
25535         NeedsAnotherRun,
25536         CompletedSuccessfully,
25537         Failed
25538     };
25539
25540     using Children = std::vector<ITrackerPtr>;
25541     TrackerContext& m_ctx;
25542     ITracker* m_parent;
25543     Children m_children;
25544     CycleState m_runState = NotStarted;
25545
25546 public:
25547     TrackerBase( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker* parent );
25548
25549     bool isComplete() const override;
25550     bool isSuccessfullyCompleted() const override;
25551     bool isOpen() const override;
25552     bool hasChildren() const override;
25553     bool hasStarted() const override {
25554         return m_runState != NotStarted;
25555     }
25556
25557     void addChild( ITrackerPtr const& child ) override;

```



```

25558
25559     ITrackerPtr findChild( NameAndLocation const& nameAndLocation ) override;
25560     ITracker& parent() override;
25561
25562     void openChild() override;
25563
25564     bool isSectionTracker() const override;
25565     bool isGeneratorTracker() const override;
25566
25567     void open();
25568
25569     void close() override;
25570     void fail() override;
25571     void markAsNeedingAnotherRun() override;
25572
25573 private:
25574     void moveToParent();
25575     void moveToThis();
25576 };
25577
25578 class SectionTracker : public TrackerBase {
25579     std::vector<std::string> m_filters;
25580     std::string m_trimmed_name;
25581 public:
25582     SectionTracker( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker* parent
25583 );
25584     bool isSectionTracker() const override;
25585
25586     bool isComplete() const override;
25587
25588     static SectionTracker& acquire( TrackerContext& ctx, NameAndLocation const& nameAndLocation );
25589
25590     void tryOpen();
25591
25592     void addInitialFilters( std::vector<std::string> const& filters );
25593     void addNextFilters( std::vector<std::string> const& filters );
25594     std::vector<std::string> const& getFilters() const;
25595     std::string const& trimmedName() const;
25596 };
25597
25600 } // namespace TestCaseTracking
25601
25602 using TestCaseTracking::ITracker;
25603 using TestCaseTracking::TrackerContext;
25604 using TestCaseTracking::SectionTracker;
25605
25606 } // namespace Catch
25607
25608 // end catch_test_case_tracker.h
25609
25610 // start catch_leak_detector.h
25611
25612 namespace Catch {
25613
25614     struct LeakDetector {
25615         LeakDetector();
25616         ~LeakDetector();
25617     };
25618
25619 }
25620 // end catch_leak_detector.h
25621 // Cpp files will be included in the single-header file here
25622 // start catch_stats.cpp
25623
25624 // Statistical analysis tools
25625
25626 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
25627
25628 #include <cassert>
25629 #include <random>
25630
25631 #if defined(CATCH_CONFIG_USE_ASYNC)
25632 #include <future>
25633 #endif
25634
25635 namespace {
25636     double erf_inv(double x) {
25637         // Code accompanying the article "Approximating the erfinv function" in GPU Computing Gems,
25638         // Volume 2
25639         double w, p;
25640
25641         w = -log((1.0 - x) * (1.0 + x));
25642
25643         if (w < 6.250000) {
25644             w = w - 3.125000;
25645             p = -3.6444120640178196996e-21;

```

```

25645         p = -1.685059138182016589e-19 + p * w;
25646         p = 1.2858480715256400167e-18 + p * w;
25647         p = 1.115787767802518096e-17 + p * w;
25648         p = -1.333171662854620906e-16 + p * w;
25649         p = 2.0972767875968561637e-17 + p * w;
25650         p = 6.6376381343583238325e-15 + p * w;
25651         p = -4.0545662729752068639e-14 + p * w;
25652         p = -8.1519341976054721522e-14 + p * w;
25653         p = 2.6335093153082322977e-12 + p * w;
25654         p = -1.2975133253453532498e-11 + p * w;
25655         p = -5.4154120542946279317e-11 + p * w;
25656         p = 1.051212273321532285e-09 + p * w;
25657         p = -4.1126339803469836976e-09 + p * w;
25658         p = -2.9070369957882005086e-08 + p * w;
25659         p = 4.2347877827932403518e-07 + p * w;
25660         p = -1.3654692000834678645e-06 + p * w;
25661         p = -1.3882523362786468719e-05 + p * w;
25662         p = 0.0001867342080340571352 + p * w;
25663         p = -0.00074070253416626697512 + p * w;
25664         p = -0.0060336708714301490533 + p * w;
25665         p = 0.24015818242558961693 + p * w;
25666         p = 1.6536545626831027356 + p * w;
25667     } else if (w < 16.000000) {
25668         w = sqrt(w) - 3.250000;
25669         p = 2.2137376921775787049e-09;
25670         p = 9.0756561938885390979e-08 + p * w;
25671         p = -2.7517406297064545428e-07 + p * w;
25672         p = 1.8239629214389227755e-08 + p * w;
25673         p = 1.5027403968909827627e-06 + p * w;
25674         p = -4.013867526981545969e-06 + p * w;
25675         p = 2.9234449089955446044e-06 + p * w;
25676         p = 1.2475304481671778723e-05 + p * w;
25677         p = -4.7318229009055733981e-05 + p * w;
25678         p = 6.8284851459573175448e-05 + p * w;
25679         p = 2.4031110387097893999e-05 + p * w;
25680         p = -0.0003550375203628474796 + p * w;
25681         p = 0.00095328937973738049703 + p * w;
25682         p = -0.0016882755560235047313 + p * w;
25683         p = 0.0024914420961078508066 + p * w;
25684         p = -0.0037512085075692412107 + p * w;
25685         p = 0.005370914553590063617 + p * w;
25686         p = 1.0052589676941592334 + p * w;
25687         p = 3.0838856104922207635 + p * w;
25688     } else {
25689         w = sqrt(w) - 5.000000;
25690         p = -2.7109920616438573243e-11;
25691         p = -2.5556418169965252055e-10 + p * w;
25692         p = 1.5076572693500548083e-09 + p * w;
25693         p = -3.7894654401267369937e-09 + p * w;
25694         p = 7.6157012080783393804e-09 + p * w;
25695         p = -1.4960026627149240478e-08 + p * w;
25696         p = 2.9147953450901080826e-08 + p * w;
25697         p = -6.7711997758452339498e-08 + p * w;
25698         p = 2.2900482228026654717e-07 + p * w;
25699         p = -9.9298272942317002539e-07 + p * w;
25700         p = 4.5260625972231537039e-06 + p * w;
25701         p = -1.9681778105531670567e-05 + p * w;
25702         p = 7.5995277030017761139e-05 + p * w;
25703         p = -0.00021503011930044477347 + p * w;
25704         p = -0.00013871931833623122026 + p * w;
25705         p = 1.0103004648645343977 + p * w;
25706         p = 4.8499064014085844221 + p * w;
25707     }
25708     return p * x;
25709 }
25710
25711 double standard_deviation(std::vector<double>::iterator first, std::vector<double>::iterator last)
25712 {
25713     auto m = Catch::Benchmark::Detail::mean(first, last);
25714     double variance = std::accumulate(first, last, 0., [m](double a, double b) {
25715         double diff = b - m;
25716         return a + diff * diff;
25717     }) / (last - first);
25718     return std::sqrt(variance);
25719 }
25720 }
25721
25722 namespace Catch {
25723     namespace Benchmark {
25724         namespace Detail {
25725
25726             double weighted_average_quantile(int k, int q, std::vector<double>::iterator first,
25727                 std::vector<double>::iterator last) {
25728                 auto count = last - first;
25729                 double idx = (count - 1) * k / static_cast<double>(q);
25729                 int j = static_cast<int>(idx);

```

```

25730         double g = idx - j;
25731         std::nth_element(first, first + j, last);
25732         auto xj = first[j];
25733         if (g == 0) return xj;
25734
25735         auto xj1 = *std::min_element(first + (j + 1), last);
25736         return xj + g * (xj1 - xj);
25737     }
25738
25739     double erfc_inv(double x) {
25740         return erf_inv(1.0 - x);
25741     }
25742
25743     double normal_quantile(double p) {
25744         static const double ROOT_TWO = std::sqrt(2.0);
25745
25746         double result = 0.0;
25747         assert(p >= 0 && p <= 1);
25748         if (p < 0 || p > 1) {
25749             return result;
25750         }
25751
25752         result = -erfc_inv(2.0 * p);
25753         // result *= normal distribution standard deviation (1.0) * sqrt(2)
25754         result *= /*sd * */ / ROOT_TWO;
25755         // result += normal disttribution mean (0)
25756         return result;
25757     }
25758
25759     double outlier_variance(Estimate<double> mean, Estimate<double> stddev, int n) {
25760         double sb = stddev.point();
25761         double mn = mean.point() / n;
25762         double mg_min = mn / 2.;
25763         double sg = (std::min)(mg_min / 4., sb / std::sqrt(n));
25764         double sg2 = sg * sg;
25765         double sb2 = sb * sb;
25766
25767         auto c_max = [n, mn, sb2, sg2](double x) -> double {
25768             double k = mn - x;
25769             double d = k * k;
25770             double nd = n * d;
25771             double k0 = -n * nd;
25772             double k1 = sb2 - n * sg2 + nd;
25773             double det = k1 * k1 - 4 * sg2 * k0;
25774             return (int)(-2. * k0 / (k1 + std::sqrt(det)));
25775         };
25776
25777         auto var_out = [n, sb2, sg2](double c) {
25778             double nc = n - c;
25779             return (nc / n) * (sb2 - nc * sg2);
25780         };
25781
25782         return (std::min)(var_out(1), var_out((std::min)(c_max(0.), c_max(mg_min)))) / sb2;
25783     }
25784
25785     bootstrap_analysis analyse_samples(double confidence_level, int n_resamples,
25786 std::vector<double>::iterator first, std::vector<double>::iterator last) {
25787         CATCH_INTERNAL_START_WARNINGS_SUPPRESSION
25788         CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS
25789         static std::random_device entropy;
25790         CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
25791
25792         auto n = static_cast<int>(last - first); // seriously, one can't use integral types
25793         without hell in C++
25794
25795         auto mean = &Detail::mean<std::vector<double>::iterator>;
25796         auto stddev = &standard_deviation;
25797
25798         #if defined(CATCH_CONFIG_USE_ASYNC)
25799         auto Estimate = [=](double(*f)(std::vector<double>::iterator,
25800 std::vector<double>::iterator)) {
25801             auto seed = entropy();
25802             return std::async(std::launch::async, [=] {
25803                 std::mt19937 rng(seed);
25804                 auto resampled = resample(rng, n_resamples, first, last, f);
25805                 return bootstrap(confidence_level, first, last, resampled, f);
25806             });
25807         };
25808
25809         auto mean_future = Estimate(mean);
25810         auto stddev_future = Estimate(stddev);
25811
25812         auto mean_estimate = mean_future.get();
25813         auto stddev_estimate = stddev_future.get();
25814     #else
25815         auto Estimate = [=](double(*f)(std::vector<double>::iterator,
25816 std::vector<double>::iterator)) {

```

```

25813         auto seed = entropy();
25814         std::mt19937 rng(seed);
25815         auto resampled = resample(rng, n_resamples, first, last, f);
25816         return bootstrap(confidence_level, first, last, resampled, f);
25817     };
25818
25819     auto mean_estimate = Estimate(mean);
25820     auto stddev_estimate = Estimate(stddev);
25821 #endif // CATCH_USE_ASYNC
25822
25823     double outlier_variance = Detail::outlier_variance(mean_estimate, stddev_estimate, n);
25824
25825     return { mean_estimate, stddev_estimate, outlier_variance };
25826 }
25827 } // namespace Detail
25828 } // namespace Benchmark
25829 } // namespace Catch
25830
25831 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
25832 // end catch_stats.cpp
25833 // start catch_approx.cpp
25834
25835 #include <cmath>
25836 #include <limits>
25837
25838 namespace {
25839
25840 // Performs equivalent check of std::fabs(lhs - rhs) <= margin
25841 // But without the subtraction to allow for INFINITY in comparison
25842 bool marginComparison(double lhs, double rhs, double margin) {
25843     return (lhs + margin >= rhs) && (rhs + margin >= lhs);
25844 }
25845
25846 }
25847
25848 namespace Catch {
25849 namespace Detail {
25850
25851     Approx::Approx( double value )
25852     :   m_epsilon( std::numeric_limits<float>::epsilon()*100 ),
25853         m_margin( 0.0 ),
25854         m_scale( 0.0 ),
25855         m_value( value )
25856     {}
25857
25858     Approx Approx::custom() {
25859         return Approx( 0 );
25860     }
25861
25862     Approx Approx::operator-() const {
25863         auto temp(*this);
25864         temp.m_value = -temp.m_value;
25865         return temp;
25866     }
25867
25868     std::string Approx::toString() const {
25869         ReusableStringStream rss;
25870         rss << "Approx( " << ::Catch::Detail::stringify( m_value ) << " )";
25871         return rss.str();
25872     }
25873
25874     bool Approx::equalityComparisonImpl(const double other) const {
25875         // First try with fixed margin, then compute margin based on epsilon, scale and Approx's value
25876         // Thanks to Richard Harris for his help refining the scaled margin value
25877         return marginComparison(m_value, other, m_margin) ||
25878            marginComparison(m_value, other, m_epsilon * (m_scale + std::fabs(std::isinf(m_value)?
25879 0 : m_value)));
25880     }
25881
25882     void Approx::setMargin(double newMargin) {
25883         CATCH_ENFORCE(newMargin >= 0,
25884             "Invalid Approx::margin: " << newMargin << "."
25885             << " Approx::Margin has to be non-negative.");
25886         m_margin = newMargin;
25887     }
25888
25889     void Approx::setEpsilon(double newEpsilon) {
25890         CATCH_ENFORCE(newEpsilon >= 0 && newEpsilon <= 1.0,
25891             "Invalid Approx::epsilon: " << newEpsilon << "."
25892             << " Approx::epsilon has to be in [0, 1]");
25893         m_epsilon = newEpsilon;
25894     }
25895 } // end namespace Detail
25896
25897 namespace literals {
25898     Detail::Approx operator "" _a(long double val) {

```

```

25899         return Detail::Approx(val);
25900     }
25901     Detail::Approx operator "" _a(unsigned long long val) {
25902         return Detail::Approx(val);
25903     }
25904 } // end namespace literals
25905
25906 std::string StringMaker<Catch::Detail::Approx>::convert(Catch::Detail::Approx const& value) {
25907     return value.toString();
25908 }
25909
25910 } // end namespace Catch
25911 // end catch_approx.cpp
25912 // start catch_assertionhandler.cpp
25913
25914 // start catch_debugger.h
25915
25916 namespace Catch {
25917     bool isDebuggerActive();
25918 }
25919
25920 #ifdef CATCH_PLATFORM_MAC
25921
25922     #if defined(__i386__) || defined(__x86_64__)
25923         #define CATCH_TRAP() __asm__("int $3\n" : : ) /* NOLINT */
25924     #elif defined(__aarch64__)
25925         #define CATCH_TRAP() __asm__(".inst 0xd43e0000")
25926     #endif
25927
25928 #elif defined(CATCH_PLATFORM_IPHONE)
25929
25930     // use inline assembler
25931     #if defined(__i386__) || defined(__x86_64__)
25932         #define CATCH_TRAP() __asm__("int $3")
25933     #elif defined(__aarch64__)
25934         #define CATCH_TRAP() __asm__(".inst 0xd4200000")
25935     #elif defined(__arm__) && !defined(__thumb__)
25936         #define CATCH_TRAP() __asm__(".inst 0xe7f001f0")
25937     #elif defined(__arm__) && defined(__thumb__)
25938         #define CATCH_TRAP() __asm__(".inst 0xde01")
25939     #endif
25940
25941 #elif defined(CATCH_PLATFORM_LINUX)
25942     // If we can use inline assembler, do it because this allows us to break
25943     // directly at the location of the failing check instead of breaking inside
25944     // raise() called from it, i.e. one stack frame below.
25945     #if defined(__GNUC__) && (defined(__i386__) || defined(__x86_64__))
25946         #define CATCH_TRAP() asm volatile ("int $3") /* NOLINT */
25947     #else // Fall back to the generic way.
25948         #include <signal.h>
25949
25950         #define CATCH_TRAP() raise(SIGTRAP)
25951     #endif
25952 #elif defined(_MSC_VER)
25953     #define CATCH_TRAP() __debugbreak()
25954 #elif defined(__MINGW32__)
25955     extern "C" __declspec(dllimport) void __stdcall DebugBreak();
25956     #define CATCH_TRAP() DebugBreak()
25957 #endif
25958
25959 #ifndef CATCH_BREAK_INTO_DEBUGGER
25960     #ifdef CATCH_TRAP
25961         #define CATCH_BREAK_INTO_DEBUGGER() []{ if( Catch::isDebuggerActive() ) { CATCH_TRAP(); } }()
25962     #else
25963         #define CATCH_BREAK_INTO_DEBUGGER() []{}()
25964     #endif
25965 #endif
25966
25967 // end catch_debugger.h
25968 // start catch_run_context.h
25969
25970 // start catch_fatal_condition.h
25971
25972 #include <cassert>
25973
25974 namespace Catch {
25975
25976     // Wrapper for platform-specific fatal error (signals/SEH) handlers
25977     //
25978     // Tries to be cooperative with other handlers, and not step over
25979     // other handlers. This means that unknown structured exceptions
25980     // are passed on, previous signal handlers are called, and so on.
25981     //
25982     // Can only be instantiated once, and assumes that once a signal
25983     // is caught, the binary will end up terminating. Thus, there
25984     class FatalConditionHandler {
25985     public:
25986         bool m_started = false;

```

```

25986
25987 // Install/disengage implementation for specific platform.
25988 // Should be if-defed to work on current platform, can assume
25989 // engage-disengage 1:1 pairing.
25990 void engage_platform();
25991 void disengage_platform();
25992 public:
25993 // Should also have platform-specific implementations as needed
25994 FatalConditionHandler();
25995 ~FatalConditionHandler();
25996
25997 void engage() {
25998     assert(!m_started && "Handler cannot be installed twice.");
25999     m_started = true;
26000     engage_platform();
26001 }
26002
26003 void disengage() {
26004     assert(m_started && "Handler cannot be uninstalled without being installed first");
26005     m_started = false;
26006     disengage_platform();
26007 }
26008 };
26009
26011 class FatalConditionHandlerGuard {
26012     FatalConditionHandler* m_handler;
26013 public:
26014     FatalConditionHandlerGuard(FatalConditionHandler* handler):
26015         m_handler(handler) {
26016         m_handler->engage();
26017     }
26018     ~FatalConditionHandlerGuard() {
26019         m_handler->disengage();
26020     }
26021 };
26022
26023 } // end namespace Catch
26024
26025 // end catch_fatal_condition.h
26026 #include <string>
26027
26028 namespace Catch {
26029
26030     struct IMutableContext;
26031
26032
26033
26034     class RunContext : public IResultCapture, public IRunner {
26035     public:
26036         RunContext( RunContext const& ) = delete;
26037         RunContext& operator =( RunContext const& ) = delete;
26038
26039         explicit RunContext( IConfigPtr const& _config, IStreamingReporterPtr&& reporter );
26040
26041         ~RunContext() override;
26042
26043         void testGroupStarting( std::string const& testSpec, std::size_t groupIndex, std::size_t
groupsCount );
26044         void testGroupEnded( std::string const& testSpec, Totals const& totals, std::size_t
groupIndex, std::size_t groupsCount );
26045
26046         Totals runTest( TestCase const& testCase);
26047
26048         IConfigPtr config() const;
26049         IStreamingReporter& reporter() const;
26050
26051     public: // IResultCapture
26052         // Assertion handlers
26053         void handleExpr
26054             ( AssertionInfo const& info,
26055               ITransientExpression const& expr,
26056               AssertionReaction& reaction ) override;
26057         void handleMessage
26058             ( AssertionInfo const& info,
26059               ResultWas::OfType resultType,
26060               StringRef const& message,
26061               AssertionReaction& reaction ) override;
26062         void handleUnexpectedExceptionNotThrown
26063             ( AssertionInfo const& info,
26064               AssertionReaction& reaction ) override;
26065         void handleUnexpectedInflightException
26066             ( AssertionInfo const& info,
26067               std::string const& message,
26068               AssertionReaction& reaction ) override;
26069         void handleIncomplete
26070             ( AssertionInfo const& info ) override;
26071         void handleExpr
26072             ( AssertionInfo const& info ) override;

```

```

26073         void handleNonExpr
26074             ( AssertionInfo const& info,
26075               ResultWas::OfType resultType,
26076               AssertionReaction& reaction ) override;
26077
26078         bool sectionStarted( SectionInfo const& sectionInfo, Counts& assertions ) override;
26079
26080         void sectionEnded( SectionEndInfo const& endInfo ) override;
26081         void sectionEndedEarly( SectionEndInfo const& endInfo ) override;
26082
26083         auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
26084             IGeneratorTracker& override;
26085
26086         #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
26087         void benchmarkPreparing( std::string const& name ) override;
26088         void benchmarkStarting( BenchmarkInfo const& info ) override;
26089         void benchmarkEnded( BenchmarkStats<> const& stats ) override;
26090         void benchmarkFailed( std::string const& error ) override;
26091         #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
26092
26093         void pushScopedMessage( MessageInfo const& message ) override;
26094         void popScopedMessage( MessageInfo const& message ) override;
26095
26096         void emplaceUnscopedMessage( MessageBuilder const& builder ) override;
26097
26098         std::string getCurrentTestName() const override;
26099
26100         const AssertionResult* getLastResult() const override;
26101
26102         void exceptionEarlyReported() override;
26103
26104         void handleFatalErrorCondition( StringRef message ) override;
26105
26106         bool lastAssertionPassed() override;
26107
26108         void assertionPassed() override;
26109
26110     public:
26111         // !TBD We need to do this another way!
26112         bool aborting() const final;
26113
26114     private:
26115         void runCurrentTest( std::string& redirectedCout, std::string& redirectedCerr );
26116         void invokeActiveTestCase();
26117
26118         void resetAssertionInfo();
26119         bool testForMissingAssertions( Counts& assertions );
26120
26121         void assertionEnded( AssertionResult const& result );
26122         void reportExpr
26123             ( AssertionInfo const& info,
26124               ResultWas::OfType resultType,
26125               ITransientExpression const* expr,
26126               bool negated );
26127
26128         void populateReaction( AssertionReaction& reaction );
26129
26130     private:
26131
26132         void handleUnfinishedSections();
26133
26134         TestRunInfo m_runInfo;
26135         IMutableContext& m_context;
26136         TestCase const* m_activeTestCase = nullptr;
26137         ITracker* m_testCaseTracker = nullptr;
26138         Option<AssertionResult> m_lastResult;
26139
26140         IConfigPtr m_config;
26141         Totals m_totals;
26142         IStreamingReporterPtr m_reporter;
26143         std::vector<MessageInfo> m_messages;
26144         std::vector<ScopedMessage> m_messageScopes; /* Keeps owners of so-called unscoped messages. */
26145         AssertionInfo m_lastAssertionInfo;
26146         std::vector<SectionEndInfo> m_unfinishedSections;
26147         std::vector<ITracker*> m_activeSections;
26148         TrackerContext m_trackerContext;
26149         FatalConditionHandler m_fatalConditionHandler;
26150         bool m_lastAssertionPassed = false;
26151         bool m_shouldReportUnexpected = true;
26152         bool m_includeSuccessfulResults;
26153     };
26154
26155     void seedRng(IConfig const& config);
26156     unsigned int rngSeed();
26157 } // end namespace Catch
26158

```

```

26159 // end catch_run_context.h
26160 namespace Catch {
26161
26162     namespace {
26163         auto operator «( std::ostream& os, ITransientExpression const& expr ) -> std::ostream& {
26164             expr.streamReconstructedExpression( os );
26165             return os;
26166         }
26167     }
26168
26169     LazyExpression::LazyExpression( bool isNegated )
26170     : m_isNegated( isNegated )
26171     {}
26172
26173     LazyExpression::LazyExpression( LazyExpression const& other ) : m_isNegated( other.m_isNegated )
26174 {}
26175
26176     LazyExpression::operator bool() const {
26177         return m_transientExpression != nullptr;
26178     }
26179
26180     auto operator « ( std::ostream& os, LazyExpression const& lazyExpr ) -> std::ostream& {
26181         if( lazyExpr.m_isNegated )
26182             os << "!";
26183
26184         if( lazyExpr ) {
26185             if( lazyExpr.m_isNegated && lazyExpr.m_transientExpression->isBinaryExpression() )
26186                 os << "(" << *lazyExpr.m_transientExpression << ")";
26187             else
26188                 os << *lazyExpr.m_transientExpression;
26189         }
26190         else {
26191             os << "{** error - unchecked empty expression requested **}";
26192         }
26193         return os;
26194     }
26195
26196     AssertionHandler::AssertionHandler
26197     (   StringRef const& macroName,
26198         SourceLineInfo const& lineInfo,
26199         StringRef capturedExpression,
26200         ResultDisposition::Flags resultDisposition )
26201     : m_assertionInfo{ macroName, lineInfo, capturedExpression, resultDisposition },
26202       m_resultCapture( getResultCapture() )
26203     {}
26204
26205     void AssertionHandler::handleExpr( ITransientExpression const& expr ) {
26206         m_resultCapture.handleExpr( m_assertionInfo, expr, m_reaction );
26207     }
26208
26209     void AssertionHandler::handleMessage(ResultWas::OfType resultType, StringRef const& message) {
26210         m_resultCapture.handleMessage( m_assertionInfo, resultType, message, m_reaction );
26211     }
26212
26213     auto AssertionHandler::allowThrows() const -> bool {
26214         return getCurrentContext().getConfig()->allowThrows();
26215     }
26216
26217     void AssertionHandler::complete() {
26218         setCompleted();
26219         if( m_reaction.shouldDebugBreak ) {
26220             // If you find your debugger stopping you here then go one level up on the
26221             // call-stack for the code that caused it (typically a failed assertion)
26222
26223             // (To go back to the test and change execution, jump over the throw, next)
26224             CATCH_BREAK_INTO_DEBUGGER();
26225         }
26226         if (m_reaction.shouldThrow) {
26227             #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
26228             throw Catch::TestFailureException();
26229             #else
26230             CATCH_ERROR( "Test failure requires aborting test!" );
26231             #endif
26232         }
26233     }
26234
26235     void AssertionHandler::setCompleted() {
26236         m_completed = true;
26237     }
26238
26239     void AssertionHandler::handleUnexpectedInflightException() {
26240         m_resultCapture.handleUnexpectedInflightException( m_assertionInfo,
26241             Catch::translateActiveException(), m_reaction );
26242     }
26243
26244     void AssertionHandler::handleExceptionThrownAsExpected() {
26245         m_resultCapture.handleNonExpr( m_assertionInfo, ResultWas::Ok, m_reaction );
26246     }

```



```

26244     void AssertionHandler::handleExceptionNotThrownAsExpected() {
26245         m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
26246     }
26247
26248     void AssertionHandler::handleUnexpectedExceptionNotThrown() {
26249         m_resultCapture.handleUnexpectedExceptionNotThrown( m_assertionInfo, m_reaction );
26250     }
26251
26252     void AssertionHandler::handleThrowingCallSkipped() {
26253         m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
26254     }
26255
26256     // This is the overload that takes a string and infers the Equals matcher from it
26257     // The more general overload, that takes any string matcher, is in catch_capture_matchers.cpp
26258     void handleExceptionMatchExpr( AssertionHandler& handler, std::string const& str, StringRef const&
matcherString ) {
26259         handleExceptionMatchExpr( handler, Matchers::Equals( str ), matcherString );
26260     }
26261
26262 } // namespace Catch
26263 // end catch_assertionhandler.cpp
26264 // start catch_assertionresult.cpp
26265
26266 namespace Catch {
26267     AssertionResultData::AssertionResultData(ResultWas::OfType _resultType, LazyExpression const &
_lazyExpression):
26268         lazyExpression(_lazyExpression),
26269         resultType(_resultType) {}
26270
26271     std::string AssertionResultData::reconstructExpression() const {
26272
26273         if( reconstructedExpression.empty() ) {
26274             if( lazyExpression ) {
26275                 ReusableStringStream rss;
26276                 rss << lazyExpression;
26277                 reconstructedExpression = rss.str();
26278             }
26279         }
26280         return reconstructedExpression;
26281     }
26282
26283     AssertionResult::AssertionResult( AssertionInfo const& info, AssertionResultData const& data )
26284     :   m_info( info ),
26285         m_resultData( data )
26286     {}
26287
26288     // Result was a success
26289     bool AssertionResult::succeeded() const {
26290         return Catch::isOk( m_resultData.resultType );
26291     }
26292
26293     // Result was a success, or failure is suppressed
26294     bool AssertionResult::isOk() const {
26295         return Catch::isOk( m_resultData.resultType ) || shouldSuppressFailure(
m_info.resultDisposition );
26296     }
26297
26298     ResultWas::OfType AssertionResult::getResultType() const {
26299         return m_resultData.resultType;
26300     }
26301
26302     bool AssertionResult::hasExpression() const {
26303         return !m_info.capturedExpression.empty();
26304     }
26305
26306     bool AssertionResult::hasMessage() const {
26307         return !m_resultData.message.empty();
26308     }
26309
26310     std::string AssertionResult::getExpression() const {
26311         // Possibly overallocating by 3 characters should be basically free
26312         std::string expr; expr.reserve(m_info.capturedExpression.size() + 3);
26313         if (isFalseTest(m_info.resultDisposition)) {
26314             expr += "!(";
26315         }
26316         expr += m_info.capturedExpression;
26317         if (isFalseTest(m_info.resultDisposition)) {
26318             expr += ')';
26319         }
26320         return expr;
26321     }
26322
26323     std::string AssertionResult::getExpressionInMacro() const {
26324         std::string expr;
26325         if( m_info.macroName.empty() )
26326             expr = static_cast<std::string>(m_info.capturedExpression);
26327         else {

```

```

26328         expr.reserve( m_info.macroName.size() + m_info.capturedExpression.size() + 4 );
26329         expr += m_info.macroName;
26330         expr += " ( ";
26331         expr += m_info.capturedExpression;
26332         expr += " ) ";
26333     }
26334     return expr;
26335 }
26336
26337 bool AssertionResult::hasExpandedExpression() const {
26338     return hasExpression() && getExpandedExpression() != getExpression();
26339 }
26340
26341 std::string AssertionResult::getExpandedExpression() const {
26342     std::string expr = m_resultData.reconstructExpression();
26343     return expr.empty()
26344         ? getExpression()
26345         : expr;
26346 }
26347
26348 std::string AssertionResult::getMessage() const {
26349     return m_resultData.message;
26350 }
26351 SourceLineInfo AssertionResult::getSourceInfo() const {
26352     return m_info.lineInfo;
26353 }
26354
26355 StringRef AssertionResult::getTestMacroName() const {
26356     return m_info.macroName;
26357 }
26358
26359 } // end namespace Catch
26360 // end catch_assertionresult.cpp
26361 // start catch_capture_matchers.cpp
26362
26363 namespace Catch {
26364
26365     using StringMatcher = Matchers::Impl::MatcherBase<std::string>;
26366
26367     // This is the general overload that takes a any string matcher
26368     // There is another overload, in catch_assertionhandler.h/.cpp, that only takes a string and
26369     // infers
26370     // the Equals matcher (so the header does not mention matchers)
26371     void handleExceptionMatchExpr( AssertionHandler& handler, StringMatcher const& matcher, StringRef
26372     const& matcherString ) {
26373         std::string exceptionMessage = Catch::translateActiveException();
26374         MatchExpr<std::string, StringMatcher const&> expr( exceptionMessage, matcher, matcherString );
26375         handler.handleExpr( expr );
26376     }
26377 } // namespace Catch
26378 // end catch_capture_matchers.cpp
26379 // start catch_commandline.cpp
26380 // start catch_commandline.h
26381
26382 // start catch_clara.h
26383
26384 // Use Catch's value for console width (store Clara's off to the side, if present)
26385 #ifndef CLARA_CONFIG_CONSOLE_WIDTH
26386 #define CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
26387 #undef CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
26388 #endif
26389 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH CATCH_CONFIG_CONSOLE_WIDTH-1
26390
26391 #ifdef __clang__
26392 #pragma clang diagnostic push
26393 #pragma clang diagnostic ignored "-Wweak-vtables"
26394 #pragma clang diagnostic ignored "-Wexit-time-destructors"
26395 #pragma clang diagnostic ignored "-Wshadow"
26396 #endif
26397
26398 // start clara.hpp
26399 // Copyright 2017 Two Blue Cubes Ltd. All rights reserved.
26400 //
26401 // Distributed under the Boost Software License, Version 1.0. (See accompanying
26402 // file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
26403 //
26404 // See https://github.com/philsquared/Clara for more details
26405
26406 // Clara v1.1.5
26407
26408 #ifndef CATCH_CLARA_CONFIG_CONSOLE_WIDTH
26409 #define CATCH_CLARA_CONFIG_CONSOLE_WIDTH 80
26410 #endif
26411 #endif
26412

```

```

26413 #ifndef CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
26414 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH CATCH_CLARA_CONFIG_CONSOLE_WIDTH
26415 #endif
26416
26417 #ifndef CLARA_CONFIG_OPTIONAL_TYPE
26418 #ifdef __has_include
26419 #if __has_include(<optional>) && __cplusplus >= 201703L
26420 #include <optional>
26421 #define CLARA_CONFIG_OPTIONAL_TYPE std::optional
26422 #endif
26423 #endif
26424 #endif
26425
26426 // ----- #included from clara_textflow.hpp -----
26427
26428 // TextFlowCpp
26429 //
26430 // A single-header library for wrapping and laying out basic text, by Phil Nash
26431 //
26432 // Distributed under the Boost Software License, Version 1.0. (See accompanying
26433 // file LICENSE.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
26434 //
26435 // This project is hosted at https://github.com/philsquared/textflowcpp
26436
26437
26438 #include <cassert>
26439 #include <ostream>
26440 #include <sstream>
26441 #include <vector>
26442
26443 #ifndef CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
26444 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH 80
26445 #endif
26446
26447 namespace Catch {
26448     namespace clara {
26449         namespace TextFlow {
26450
26451             inline auto isWhitespace(char c) -> bool {
26452                 static std::string chars = " \t\n\r";
26453                 return chars.find(c) != std::string::npos;
26454             }
26455             inline auto isBreakableBefore(char c) -> bool {
26456                 static std::string chars = "[(|<|";
26457                 return chars.find(c) != std::string::npos;
26458             }
26459             inline auto isBreakableAfter(char c) -> bool {
26460                 static std::string chars = "])>.,;+*=~/\\\"";
26461                 return chars.find(c) != std::string::npos;
26462             }
26463
26464             class Columns;
26465
26466             class Column {
26467             public:
26468                 std::vector<std::string> m_strings;
26469                 size_t m_width = CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH;
26470                 size_t m_indent = 0;
26471                 size_t m_initialIndent = std::string::npos;
26472
26473                 class iterator {
26474                 public:
26475                     friend Column;
26476
26477                     Column const& m_column;
26478                     size_t m_stringIndex = 0;
26479                     size_t m_pos = 0;
26480
26481                     size_t m_len = 0;
26482                     size_t m_end = 0;
26483                     bool m_suffix = false;
26484
26485                     iterator(Column const& column, size_t stringIndex)
26486                         : m_column(column),
26487                         m_stringIndex(stringIndex) {}
26488
26489                     auto line() const -> std::string const& { return m_column.m_strings[m_stringIndex]; }
26490
26491                     auto isBoundary(size_t at) const -> bool {
26492                         assert(at > 0);
26493                         assert(at <= line().size());
26494
26495                         return at == line().size() ||
26496                                (isWhitespace(line()[at]) && !isWhitespace(line()[at - 1])) ||
26497                                isBreakableBefore(line()[at]) ||
26498                                isBreakableAfter(line()[at - 1]));
26499                     }
26500                 };
26501             };
26502
26503             Columns
26504
26505             Column
26506
26507             auto line() const -> std::string const& { return m_column.m_strings[m_stringIndex]; }
26508
26509             auto isBoundary(size_t at) const -> bool {
26510                 assert(at > 0);
26511                 assert(at <= line().size());
26512
26513                 return at == line().size() ||
26514                        (isWhitespace(line()[at]) && !isWhitespace(line()[at - 1])) ||
26515                        isBreakableBefore(line()[at]) ||
26516                        isBreakableAfter(line()[at - 1]));
26517             }
26518         };
26519     };
26520 }

```

```

26500     void calcLength() {
26501         assert(m_stringIndex < m_column.m_strings.size());
26502
26503         m_suffix = false;
26504         auto width = m_column.m_width - indent();
26505         m_end = m_pos;
26506         if (line()[m_pos] == '\n') {
26507             ++m_end;
26508         }
26509         while (m_end < line().size() && line()[m_end] != '\n')
26510             ++m_end;
26511
26512         if (m_end < m_pos + width) {
26513             m_len = m_end - m_pos;
26514         } else {
26515             size_t len = width;
26516             while (len > 0 && !isBoundary(m_pos + len))
26517                 --len;
26518             while (len > 0 && isWhitespace(line()[m_pos + len - 1]))
26519                 --len;
26520
26521             if (len > 0) {
26522                 m_len = len;
26523             } else {
26524                 m_suffix = true;
26525                 m_len = width - 1;
26526             }
26527         }
26528     }
26529
26530     auto indent() const -> size_t {
26531         auto initial = m_pos == 0 && m_stringIndex == 0 ? m_column.m_initialIndent :
std::string::npos;
26532         return initial == std::string::npos ? m_column.m_indent : initial;
26533     }
26534
26535     auto addIndentAndSuffix(std::string const &plain) const -> std::string {
26536         return std::string(indent(), ' ') + (m_suffix ? plain + "-" : plain);
26537     }
26538
26539 public:
26540     using difference_type = std::ptrdiff_t;
26541     using value_type = std::string;
26542     using pointer = value_type * ;
26543     using reference = value_type & ;
26544     using iterator_category = std::forward_iterator_tag;
26545
26546     explicit iterator(Column const& column) : m_column(column) {
26547         assert(m_column.m_width > m_column.m_indent);
26548         assert(m_column.m_initialIndent == std::string::npos || m_column.m_width >
m_column.m_initialIndent);
26549         calcLength();
26550         if (m_len == 0)
26551             m_stringIndex++; // Empty string
26552     }
26553
26554     auto operator *() const -> std::string {
26555         assert(m_stringIndex < m_column.m_strings.size());
26556         assert(m_pos <= m_end);
26557         return addIndentAndSuffix(line().substr(m_pos, m_len));
26558     }
26559
26560     auto operator ++() -> iterator& {
26561         m_pos += m_len;
26562         if (m_pos < line().size() && line()[m_pos] == '\n')
26563             m_pos += 1;
26564         else
26565             while (m_pos < line().size() && isWhitespace(line()[m_pos]))
26566                 ++m_pos;
26567
26568         if (m_pos == line().size()) {
26569             m_pos = 0;
26570             ++m_stringIndex;
26571         }
26572         if (m_stringIndex < m_column.m_strings.size())
26573             calcLength();
26574         return *this;
26575     }
26576     auto operator ++(int) -> iterator {
26577         iterator prev(*this);
26578         operator ++();
26579         return prev;
26580     }
26581
26582     auto operator ==(iterator const& other) const -> bool {
26583         return
26584             m_pos == other.m_pos &&

```

```

26585         m_stringIndex == other.m_stringIndex &&
26586         &m_column == &other.m_column;
26587     }
26588     auto operator !=(iterator const& other) const -> bool {
26589         return !operator==(other);
26590     }
26591 };
26592 using const_iterator = iterator;
26593
26594 explicit Column(std::string const& text) { m_strings.push_back(text); }
26595
26596 auto width(size_t newWidth) -> Column& {
26597     assert(newWidth > 0);
26598     m_width = newWidth;
26599     return *this;
26600 }
26601 auto indent(size_t newIndent) -> Column& {
26602     m_indent = newIndent;
26603     return *this;
26604 }
26605 auto initialIndent(size_t newIndent) -> Column& {
26606     m_initialIndent = newIndent;
26607     return *this;
26608 }
26609
26610 auto width() const -> size_t { return m_width; }
26611 auto begin() const -> iterator { return iterator(*this); }
26612 auto end() const -> iterator { return { *this, m_strings.size() }; }
26613
26614 inline friend std::ostream& operator << (std::ostream& os, Column const& col) {
26615     bool first = true;
26616     for (auto line : col) {
26617         if (first)
26618             first = false;
26619         else
26620             os << "\n";
26621         os << line;
26622     }
26623     return os;
26624 }
26625
26626 auto operator + (Column const& other) -> Columns;
26627
26628 auto toString() const -> std::string {
26629     std::ostringstream oss;
26630     oss << *this;
26631     return oss.str();
26632 }
26633 };
26634
26635 class Spacer : public Column {
26636 public:
26637     explicit Spacer(size_t spaceWidth) : Column("") {
26638         width(spaceWidth);
26639     }
26640 };
26641
26642 class Columns {
26643     std::vector<Column> m_columns;
26644 public:
26645     class iterator {
26646     friend Columns;
26647     struct EndTag {};
26648
26649     std::vector<Column> const& m_columns;
26650     std::vector<Column::iterator> m_iterators;
26651     size_t m_activeIterators;
26652
26653     iterator(Columns const& columns, EndTag)
26654         : m_columns(columns.m_columns),
26655         m_activeIterators(0) {
26656         m_iterators.reserve(m_columns.size());
26657
26658         for (auto const& col : m_columns)
26659             m_iterators.push_back(col.end());
26660     }
26661
26662 public:
26663     using difference_type = std::ptrdiff_t;
26664     using value_type = std::string;
26665     using pointer = value_type * ;
26666     using reference = value_type & ;
26667     using iterator_category = std::forward_iterator_tag;

```

```

26672     explicit iterator(Columns const& columns)
26673     : m_columns(columns.m_columns),
26674       m_activeIterators(m_columns.size()) {
26675         m_iterators.reserve(m_columns.size());
26676
26677         for (auto const& col : m_columns)
26678             m_iterators.push_back(col.begin());
26679     }
26680
26681     auto operator ==(iterator const& other) const -> bool {
26682         return m_iterators == other.m_iterators;
26683     }
26684     auto operator !=(iterator const& other) const -> bool {
26685         return m_iterators != other.m_iterators;
26686     }
26687     auto operator *() const -> std::string {
26688         std::string row, padding;
26689
26690         for (size_t i = 0; i < m_columns.size(); ++i) {
26691             auto width = m_columns[i].width();
26692             if (m_iterators[i] != m_columns[i].end()) {
26693                 std::string col = *m_iterators[i];
26694                 row += padding + col;
26695                 if (col.size() < width)
26696                     padding = std::string(width - col.size(), ' ');
26697             }
26698             else
26699                 padding = "";
26700             padding += std::string(width, ' ');
26701         }
26702         return row;
26703     }
26704
26705     auto operator ++() -> iterator& {
26706         for (size_t i = 0; i < m_columns.size(); ++i) {
26707             if (m_iterators[i] != m_columns[i].end())
26708                 ++m_iterators[i];
26709         }
26710         return *this;
26711     }
26712     auto operator ++(int) -> iterator {
26713         iterator prev(*this);
26714         operator++;
26715         return prev;
26716     }
26717 };
26718 using const_iterator = iterator;
26719
26720 auto begin() const -> iterator { return iterator(*this); }
26721 auto end() const -> iterator { return { *this, iterator::EndTag() }; }
26722
26723 auto operator += (Column const& col) -> Columns& {
26724     m_columns.push_back(col);
26725     return *this;
26726 }
26727 auto operator + (Column const& col) -> Columns {
26728     Columns combined = *this;
26729     combined += col;
26730     return combined;
26731 }
26732
26733 inline friend std::ostream& operator << (std::ostream& os, Columns const& cols) {
26734
26735     bool first = true;
26736     for (auto line : cols) {
26737         if (first)
26738             first = false;
26739         else
26740             os << "\n";
26741         os << line;
26742     }
26743     return os;
26744 }
26745
26746 auto toString() const -> std::string {
26747     std::ostringstream oss;
26748     oss << *this;
26749     return oss.str();
26750 }
26751 };
26752
26753 inline auto Column::operator + (Column const& other) -> Columns {
26754     Columns cols;
26755     cols += *this;
26756     cols += other;
26757     return cols;
26758 }

```

```

26759 }
26760
26761 }
26762 }
26763
26764 // ----- end of #include from clara_textflow.hpp -----
26765 // ..... back in clara.hpp
26766
26767 #include <cctype>
26768 #include <string>
26769 #include <memory>
26770 #include <set>
26771 #include <algorithm>
26772
26773 #if !defined(CATCH_PLATFORM_WINDOWS) && ( defined(WIN32) || defined(__WIN32__) || defined(_WIN32) ||
    defined(_MSC_VER) )
26774 #define CATCH_PLATFORM_WINDOWS
26775 #endif
26776
26777 namespace Catch { namespace clara {
26778 namespace detail {
26779
26780     // Traits for extracting arg and return type of lambdas (for single argument lambdas)
26781     template<typename L>
26782     struct UnaryLambdaTraits : UnaryLambdaTraits<decltype( &L::operator() )> {};
26783
26784     template<typename ClassT, typename ReturnT, typename... Args>
26785     struct UnaryLambdaTraits<ReturnT( ClassT::* )( Args... ) const> {
26786         static const bool isValid = false;
26787     };
26788
26789     template<typename ClassT, typename ReturnT, typename ArgT>
26790     struct UnaryLambdaTraits<ReturnT( ClassT::* )( ArgT ) const> {
26791         static const bool isValid = true;
26792         using ArgType = typename std::remove_const<typename std::remove_reference<ArgT>::type>::type;
26793         using ReturnT = ReturnT;
26794     };
26795
26796     class TokenStream;
26797
26798     // Transport for raw args (copied from main args, or supplied via init list for testing)
26799     class Args {
26800     friend TokenStream;
26801     std::string m_exeName;
26802     std::vector<std::string> m_args;
26803
26804     public:
26805         Args( int argc, char const* const* argv )
26806             : m_exeName(argv[0]),
26807               m_args(argv + 1, argv + argc) {}
26808
26809         Args( std::initializer_list<std::string> args )
26810             : m_exeName( *args.begin() ),
26811               m_args( args.begin()+1, args.end() )
26812         {}
26813
26814         auto exeName() const -> std::string {
26815             return m_exeName;
26816         }
26817     };
26818
26819     // Wraps a token coming from a token stream. These may not directly correspond to strings as a
26820     // single string
26821     // may encode an option + its argument if the : or = form is used
26822     enum class TokenType {
26823         Option, Argument
26824     };
26825     struct Token {
26826         TokenType type;
26827         std::string token;
26828     };
26829
26830     inline auto isOptPrefix( char c ) -> bool {
26831         return c == '-' || c == '/'
26832     }
26833 #ifdef CATCH_PLATFORM_WINDOWS
26834     || c == '/'
26835 #endif
26836 #endif
26837
26838     // Abstracts iterators into args as a stream of tokens, with option arguments uniformly handled
26839     class TokenStream {
26840     using Iterator = std::vector<std::string>::const_iterator;
26841     Iterator it;
26842     Iterator itEnd;
26843     std::vector<Token> m_tokenBuffer;
26844

```

```

26844     void loadBuffer() {
26845         m_tokenBuffer.resize( 0 );
26846
26847         // Skip any empty strings
26848         while( it != itEnd && it->empty() )
26849             ++it;
26850
26851         if( it != itEnd ) {
26852             auto const &next = *it;
26853             if( isOptPrefix( next[0] ) ) {
26854                 auto delimiterPos = next.find_first_of( " :=" );
26855                 if( delimiterPos != std::string::npos ) {
26856                     m_tokenBuffer.push_back( { TokenType::Option, next.substr( 0, delimiterPos ) }
26857 );
26858                     m_tokenBuffer.push_back( { TokenType::Argument, next.substr( delimiterPos + 1
26859 ) } );
26860                 } else {
26861                     if( next[1] != '-' && next.size() > 2 ) {
26862                         std::string opt = "- ";
26863                         for( size_t i = 1; i < next.size(); ++i ) {
26864                             opt[i] = next[i];
26865                             m_tokenBuffer.push_back( { TokenType::Option, opt } );
26866                         }
26867                     } else {
26868                         m_tokenBuffer.push_back( { TokenType::Option, next } );
26869                     }
26870                 } else {
26871                     m_tokenBuffer.push_back( { TokenType::Argument, next } );
26872                 }
26873             }
26874         }
26875     public:
26876     explicit TokenStream( Args const &args ) : TokenStream( args.m_args.begin(), args.m_args.end()
26877 ) {}
26878
26879     TokenStream( Iterator it, Iterator itEnd ) : it( it ), itEnd( itEnd ) {
26880         loadBuffer();
26881     }
26882
26883     explicit operator bool() const {
26884         return !m_tokenBuffer.empty() || it != itEnd;
26885     }
26886
26887     auto count() const -> size_t { return m_tokenBuffer.size() + (itEnd - it); }
26888
26889     auto operator*() const -> Token {
26890         assert( !m_tokenBuffer.empty() );
26891         return m_tokenBuffer.front();
26892     }
26893
26894     auto operator->() const -> Token const * {
26895         assert( !m_tokenBuffer.empty() );
26896         return &m_tokenBuffer.front();
26897     }
26898
26899     auto operator++() -> TokenStream & {
26900         if( m_tokenBuffer.size() >= 2 ) {
26901             m_tokenBuffer.erase( m_tokenBuffer.begin() );
26902         } else {
26903             if( it != itEnd )
26904                 ++it;
26905             loadBuffer();
26906         }
26907         return *this;
26908     }
26909
26910     class ResultBase {
26911     public:
26912         enum Type {
26913             Ok, LogicError, RuntimeError
26914         };
26915
26916     protected:
26917         ResultBase( Type type ) : m_type( type ) {}
26918         virtual ~ResultBase() = default;
26919
26920         virtual void enforceOk() const = 0;
26921
26922         Type m_type;
26923     };
26924
26925     template<typename T>
26926     class ResultValueBase : public ResultBase {
26927     public:

```



```

26928     auto value() const -> T const & {
26929         enforceOk();
26930         return m_value;
26931     }
26932
26933 protected:
26934     ResultValueBase( Type type ) : ResultBase( type ) {}
26935
26936     ResultValueBase( ResultValueBase const &other ) : ResultBase( other ) {
26937         if( m_type == ResultBase::Ok )
26938             new( &m_value ) T( other.m_value );
26939     }
26940
26941     ResultValueBase( Type, T const &value ) : ResultBase( Ok ) {
26942         new( &m_value ) T( value );
26943     }
26944
26945     auto operator=( ResultValueBase const &other ) -> ResultValueBase & {
26946         if( m_type == ResultBase::Ok )
26947             m_value.~T();
26948         ResultBase::operator=(other);
26949         if( m_type == ResultBase::Ok )
26950             new( &m_value ) T( other.m_value );
26951         return *this;
26952     }
26953
26954     ~ResultValueBase() override {
26955         if( m_type == Ok )
26956             m_value.~T();
26957     }
26958
26959     union {
26960         T m_value;
26961     };
26962 };
26963
26964 template<>
26965 class ResultValueBase<void> : public ResultBase {
26966 protected:
26967     using ResultBase::ResultBase;
26968 };
26969
26970 template<typename T = void>
26971 class BasicResult : public ResultValueBase<T> {
26972 public:
26973     template<typename U>
26974     explicit BasicResult( BasicResult<U> const &other )
26975         : ResultValueBase<T>( other.type() ),
26976           m_errorMessage( other.errorMessage() )
26977     {
26978         assert( type() != ResultBase::Ok );
26979     }
26980
26981     template<typename U>
26982     static auto ok( U const &value ) -> BasicResult { return { ResultBase::Ok, value }; }
26983     static auto ok() -> BasicResult { return { ResultBase::Ok }; }
26984     static auto logicError( std::string const &message ) -> BasicResult { return {
ResultBase::LogicError, message }; }
26985     static auto runtimeError( std::string const &message ) -> BasicResult { return {
ResultBase::RuntimeError, message }; }
26986
26987     explicit operator bool() const { return m_type == ResultBase::Ok; }
26988     auto type() const -> ResultBase::Type { return m_type; }
26989     auto errorMessage() const -> std::string { return m_errorMessage; }
26990
26991 protected:
26992     void enforceOk() const override {
26993
26994         // Errors shouldn't reach this point, but if they do
26995         // the actual error message will be in m_errorMessage
26996         assert( m_type != ResultBase::LogicError );
26997         assert( m_type != ResultBase::RuntimeError );
26998         if( m_type != ResultBase::Ok )
26999             std::abort();
27000     }
27001
27002     std::string m_errorMessage; // Only populated if resultType is an error
27003
27004     BasicResult( ResultBase::Type type, std::string const &message )
27005         : ResultValueBase<T>(type),
27006           m_errorMessage(message)
27007     {
27008         assert( m_type != ResultBase::Ok );
27009     }
27010
27011     using ResultValueBase<T>::ResultValueBase;
27012     using ResultBase::m_type;

```

```

27013     };
27014
27015     enum class ParseResultType {
27016         Matched, NoMatch, ShortCircuitAll, ShortCircuitSame
27017     };
27018
27019     class ParseState {
27020     public:
27021
27022         ParseState( ParseResultType type, TokenStream const &remainingTokens )
27023             : m_type(type),
27024               m_remainingTokens( remainingTokens )
27025         {}
27026
27027         auto type() const -> ParseResultType { return m_type; }
27028         auto remainingTokens() const -> TokenStream { return m_remainingTokens; }
27029
27030     private:
27031         ParseResultType m_type;
27032         TokenStream m_remainingTokens;
27033     };
27034
27035     using Result = BasicResult<void>;
27036     using ParserResult = BasicResult<ParseResultType>;
27037     using InternalParseResult = BasicResult<ParseState>;
27038
27039     struct HelpColumns {
27040         std::string left;
27041         std::string right;
27042     };
27043
27044     template<typename T>
27045     inline auto convertInto( std::string const &source, T& target ) -> ParserResult {
27046         std::stringstream ss;
27047         ss << source;
27048         ss >> target;
27049         if( ss.fail() )
27050             return ParserResult::runtimeError( "Unable to convert '" + source + "' to destination
type" );
27051         else
27052             return ParserResult::ok( ParseResultType::Matched );
27053     }
27054     inline auto convertInto( std::string const &source, std::string& target ) -> ParserResult {
27055         target = source;
27056         return ParserResult::ok( ParseResultType::Matched );
27057     }
27058     inline auto convertInto( std::string const &source, bool &target ) -> ParserResult {
27059         std::string srcLC = source;
27060         std::transform( srcLC.begin(), srcLC.end(), srcLC.begin(), []( unsigned char c ) { return
static_cast<char>( std::tolower(c) ); } );
27061         if (srcLC == "y" || srcLC == "1" || srcLC == "true" || srcLC == "yes" || srcLC == "on")
27062             target = true;
27063         else if (srcLC == "n" || srcLC == "0" || srcLC == "false" || srcLC == "no" || srcLC == "off")
27064             target = false;
27065         else
27066             return ParserResult::runtimeError( "Expected a boolean value but did not recognise: '" +
source + "'" );
27067         return ParserResult::ok( ParseResultType::Matched );
27068     }
27069     #ifndef CLARA_CONFIG_OPTIONAL_TYPE
27070     template<typename T>
27071     inline auto convertInto( std::string const &source, CLARA_CONFIG_OPTIONAL_TYPE<T>& target ) ->
ParserResult {
27072         T temp;
27073         auto result = convertInto( source, temp );
27074         if( result )
27075             target = std::move(temp);
27076         return result;
27077     }
27078     #endif // CLARA_CONFIG_OPTIONAL_TYPE
27079
27080     struct NonCopyable {
27081         NonCopyable() = default;
27082         NonCopyable( NonCopyable const & ) = delete;
27083         NonCopyable( NonCopyable && ) = delete;
27084         NonCopyable &operator=( NonCopyable const & ) = delete;
27085         NonCopyable &operator=( NonCopyable && ) = delete;
27086     };
27087
27088     struct BoundRef : NonCopyable {
27089         virtual ~BoundRef() = default;
27090         virtual auto isContainer() const -> bool { return false; }
27091         virtual auto isFlag() const -> bool { return false; }
27092     };
27093     struct BoundValueRefBase : BoundRef {
27094         virtual auto setValue( std::string const &arg ) -> ParserResult = 0;
27095     };

```

```

27096     struct BoundFlagRefBase : BoundRef {
27097         virtual auto setFlag( bool flag ) -> ParserResult = 0;
27098         virtual auto isFlag() const -> bool { return true; }
27099     };
27100
27101     template<typename T>
27102     struct BoundValueRef : BoundValueRefBase {
27103         T &m_ref;
27104
27105         explicit BoundValueRef( T &ref ) : m_ref( ref ) {}
27106
27107         auto setValue( std::string const &arg ) -> ParserResult override {
27108             return convertInto( arg, m_ref );
27109         }
27110     };
27111
27112     template<typename T>
27113     struct BoundValueRef<std::vector<T> > : BoundValueRefBase {
27114         std::vector<T> &m_ref;
27115
27116         explicit BoundValueRef( std::vector<T> &ref ) : m_ref( ref ) {}
27117
27118         auto isContainer() const -> bool override { return true; }
27119
27120         auto setValue( std::string const &arg ) -> ParserResult override {
27121             T temp;
27122             auto result = convertInto( arg, temp );
27123             if( result )
27124                 m_ref.push_back( temp );
27125             return result;
27126         }
27127     };
27128
27129     struct BoundFlagRef : BoundFlagRefBase {
27130         bool &m_ref;
27131
27132         explicit BoundFlagRef( bool &ref ) : m_ref( ref ) {}
27133
27134         auto setFlag( bool flag ) -> ParserResult override {
27135             m_ref = flag;
27136             return ParserResult::ok( ParseResultType::Matched );
27137         }
27138     };
27139
27140     template<typename Return>
27141     struct LambdaInvoker {
27142         static_assert( std::is_same<Return, ParserResult>::value, "Lambda must return void or
27143         clara::ParserResult" );
27144
27145         template<typename L, typename ArgType>
27146         static auto invoke( L const &lambda, ArgType const &arg ) -> ParserResult {
27147             return lambda( arg );
27148         }
27149     };
27150
27151     template<>
27152     struct LambdaInvoker<void> {
27153         template<typename L, typename ArgType>
27154         static auto invoke( L const &lambda, ArgType const &arg ) -> ParserResult {
27155             lambda( arg );
27156             return ParserResult::ok( ParseResultType::Matched );
27157         }
27158     };
27159
27160     template<typename ArgType, typename L>
27161     inline auto invokeLambda( L const &lambda, std::string const &arg ) -> ParserResult {
27162         ArgType temp{};
27163         auto result = convertInto( arg, temp );
27164         return !result
27165             ? result
27166             : LambdaInvoker<typename UnaryLambdaTraits<L>::Return>::invoke( lambda, temp );
27167     }
27168
27169     template<typename L>
27170     struct BoundLambda : BoundValueRefBase {
27171         L m_lambda;
27172
27173         static_assert( UnaryLambdaTraits<L>::isValid, "Supplied lambda must take exactly one argument" );
27174
27175         explicit BoundLambda( L const &lambda ) : m_lambda( lambda ) {}
27176
27177         auto setValue( std::string const &arg ) -> ParserResult override {
27178             return invokeLambda<typename UnaryLambdaTraits<L>::ArgType>( m_lambda, arg );
27179         }
27180     };
27181
27182     template<typename L>

```

```

27181     struct BoundFlagLambda : BoundFlagRefBase {
27182         L m_lambda;
27183
27184         static_assert( UnaryLambdaTraits<L>::isValid, "Supplied lambda must take exactly one argument"
27185 );
27186         static_assert( std::is_same<typename UnaryLambdaTraits<L>::ArgType, bool>::value, "flags must
be boolean" );
27187
27188         explicit BoundFlagLambda( L const &lambda ) : m_lambda( lambda ) {}
27189
27190         auto setFlag( bool flag ) -> ParserResult override {
27191             return LambdaInvoker<typename UnaryLambdaTraits<L>::ReturnType>::invoke( m_lambda, flag );
27192         }
27193     };
27194
27195     enum class Optionality { Optional, Required };
27196
27197     struct Parser;
27198
27199     class ParserBase {
27200     public:
27201         virtual ~ParserBase() = default;
27202         virtual auto validate() const -> Result { return Result::ok(); }
27203         virtual auto parse( std::string const& exeName, TokenStream const &tokens) const ->
InternalParseResult = 0;
27204         virtual auto cardinality() const -> size_t { return 1; }
27205
27206         auto parse( Args const &args ) const -> InternalParseResult {
27207             return parse( args.exeName(), TokenStream( args ) );
27208         }
27209     };
27210
27211     template<typename DerivedT>
27212     class ComposableParserImpl : public ParserBase {
27213     public:
27214         template<typename T>
27215         auto operator|( T const &other ) const -> Parser;
27216
27217         template<typename T>
27218         auto operator+( T const &other ) const -> Parser;
27219     };
27220
27221     // Common code and state for Args and Opts
27222     template<typename DerivedT>
27223     class ParserRefImpl : public ComposableParserImpl<DerivedT> {
27224     protected:
27225         Optionality m_optionality = Optionality::Optional;
27226         std::shared_ptr<BoundRef> m_ref;
27227         std::string m_hint;
27228         std::string m_description;
27229
27230         explicit ParserRefImpl( std::shared_ptr<BoundRef> const &ref ) : m_ref( ref ) {}
27231     public:
27232         template<typename T>
27233         ParserRefImpl( T &ref, std::string const &hint )
27234             : m_ref( std::make_shared<BoundValueRef<T>>( ref ) ),
27235               m_hint( hint )
27236         {}
27237
27238         template<typename LambdaT>
27239         ParserRefImpl( LambdaT const &ref, std::string const &hint )
27240             : m_ref( std::make_shared<BoundLambda<LambdaT>>( ref ) ),
27241               m_hint( hint )
27242         {}
27243
27244         auto operator()( std::string const &description ) -> DerivedT & {
27245             m_description = description;
27246             return static_cast<DerivedT &>( *this );
27247         }
27248
27249         auto optional() -> DerivedT & {
27250             m_optionality = Optionality::Optional;
27251             return static_cast<DerivedT &>( *this );
27252         };
27253
27254         auto required() -> DerivedT & {
27255             m_optionality = Optionality::Required;
27256             return static_cast<DerivedT &>( *this );
27257         };
27258
27259         auto isOptional() const -> bool {
27260             return m_optionality == Optionality::Optional;
27261         }
27262
27263         auto cardinality() const -> size_t override {
27264             if( m_ref->isContainer() )

```

```

27265         return 0;
27266     else
27267         return 1;
27268     }
27269
27270     auto hint() const -> std::string { return m_hint; }
27271 };
27272
27273 class ExeName : public ComposableParserImpl<ExeName> {
27274     std::shared_ptr<std::string> m_name;
27275     std::shared_ptr<BoundValueRefBase> m_ref;
27276
27277     template<typename LambdaT>
27278     static auto makeRef(LambdaT const& lambda) -> std::shared_ptr<BoundValueRefBase> {
27279         return std::make_shared<BoundLambda<LambdaT>>( lambda );
27280     }
27281
27282 public:
27283     ExeName() : m_name( std::make_shared<std::string>( "<executable>" ) ) {}
27284
27285     explicit ExeName( std::string& ref ) : ExeName() {
27286         m_ref = std::make_shared<BoundValueRef<std::string>>( ref );
27287     }
27288
27289     template<typename LambdaT>
27290     explicit ExeName( LambdaT const& lambda ) : ExeName() {
27291         m_ref = std::make_shared<BoundLambda<LambdaT>>( lambda );
27292     }
27293
27294     // The exe name is not parsed out of the normal tokens, but is handled specially
27295     auto parse( std::string const&, TokenStream const& tokens ) const -> InternalParseResult
27296 override {
27297         return InternalParseResult::ok( ParseState( ParseResultType::NoMatch, tokens ) );
27298     }
27299
27300     auto name() const -> std::string { return *m_name; }
27301     auto set( std::string const& newName ) -> ParserResult {
27302         auto lastSlash = newName.find_last_of( "\\/" );
27303         auto filename = ( lastSlash == std::string::npos )
27304             ? newName
27305             : newName.substr( lastSlash+1 );
27306
27307         *m_name = filename;
27308         if( m_ref )
27309             return m_ref->setValue( filename );
27310         else
27311             return ParserResult::ok( ParseResultType::Matched );
27312     }
27313 };
27314
27315 class Arg : public ParserRefImpl<Arg> {
27316 public:
27317     using ParserRefImpl::ParserRefImpl;
27318
27319     auto parse( std::string const&, TokenStream const& tokens ) const -> InternalParseResult
27320 override {
27321         auto validationResult = validate();
27322         if( !validationResult )
27323             return InternalParseResult( validationResult );
27324
27325         auto remainingTokens = tokens;
27326         auto const& token = *remainingTokens;
27327         if( token.type != TokenType::Argument )
27328             return InternalParseResult::ok( ParseState( ParseResultType::NoMatch, remainingTokens
27329 ) );
27330
27331         assert( !m_ref->isFlag() );
27332         auto valueRef = static_cast<detail::BoundValueRefBase*>( m_ref.get() );
27333
27334         auto result = valueRef->setValue( remainingTokens->token );
27335         if( !result )
27336             return InternalParseResult( result );
27337         else
27338             return InternalParseResult::ok( ParseState( ParseResultType::Matched,
27339 ++remainingTokens ) );
27340     }
27341 };
27342
27343 inline auto normaliseOpt( std::string const& optName ) -> std::string {
27344 #ifdef CATCH_PLATFORM_WINDOWS
27345     if( optName[0] == '/' )
27346         return "-" + optName.substr( 1 );
27347     else
27348         return optName;
27349 #endif
27350 }

```

```

27348
27349     class Opt : public ParserRefImpl<Opt> {
27350     protected:
27351         std::vector<std::string> m_optNames;
27352
27353     public:
27354         template<typename LambdaT>
27355         explicit Opt( LambdaT const &ref ) : ParserRefImpl( std::make_shared<BoundFlagLambda<LambdaT>(
ref ) ) {}
27356
27357         explicit Opt( bool &ref ) : ParserRefImpl( std::make_shared<BoundFlagRef>( ref ) ) {}
27358
27359         template<typename LambdaT>
27360         Opt( LambdaT const &ref, std::string const &hint ) : ParserRefImpl( ref, hint ) {}
27361
27362         template<typename T>
27363         Opt( T &ref, std::string const &hint ) : ParserRefImpl( ref, hint ) {}
27364
27365         auto operator[]( std::string const &optName ) -> Opt & {
27366             m_optNames.push_back( optName );
27367             return *this;
27368         }
27369
27370         auto getHelpColumns() const -> std::vector<HelpColumns> {
27371             std::ostringstream oss;
27372             bool first = true;
27373             for( auto const &opt : m_optNames ) {
27374                 if (first)
27375                     first = false;
27376                 else
27377                     oss << ", ";
27378                 oss << opt;
27379             }
27380             if( !m_hint.empty() )
27381                 oss << " <" << m_hint << ">";
27382             return { { oss.str(), m_description } };
27383         }
27384
27385         auto isMatch( std::string const &optToken ) const -> bool {
27386             auto normalisedToken = normaliseOpt( optToken );
27387             for( auto const &name : m_optNames ) {
27388                 if( normaliseOpt( name ) == normalisedToken )
27389                     return true;
27390             }
27391             return false;
27392         }
27393
27394         using ParserBase::parse;
27395
27396         auto parse( std::string const&, TokenStream const &tokens ) const -> InternalParseResult
27397         override {
27398             auto validationResult = validate();
27399             if( !validationResult )
27400                 return InternalParseResult( validationResult );
27401
27402             auto remainingTokens = tokens;
27403             if( remainingTokens && remainingTokens->type == TokenType::Option ) {
27404                 auto const &token = *remainingTokens;
27405                 if( isMatch(token.token) ) {
27406                     if( m_ref->isFlag() ) {
27407                         auto flagRef = static_cast<detail::BoundFlagRefBase*>( m_ref.get() );
27408                         auto result = flagRef->setFlag( true );
27409                         if( !result )
27410                             return InternalParseResult( result );
27411                         if( result.value() == ParseResultType::ShortCircuitAll )
27412                             return InternalParseResult::ok( ParseState( result.value(),
remainingTokens ) );
27413                     } else {
27414                         auto valueRef = static_cast<detail::BoundValueRefBase*>( m_ref.get() );
27415                         ++remainingTokens;
27416                         if( !remainingTokens )
27417                             return InternalParseResult::runtimeError( "Expected argument following " +
token.token );
27418                         auto const &argToken = *remainingTokens;
27419                         if( argToken.type != TokenType::Argument )
27420                             return InternalParseResult::runtimeError( "Expected argument following " +
token.token );
27421                         auto result = valueRef->setValue( argToken.token );
27422                         if( !result )
27423                             return InternalParseResult( result );
27424                         if( result.value() == ParseResultType::ShortCircuitAll )
27425                             return InternalParseResult::ok( ParseState( result.value(),
remainingTokens ) );
27426                     }
27427                     return InternalParseResult::ok( ParseState( ParseResultType::Matched,
++remainingTokens ) );
27428                 }
27429             }

```

```

27428         }
27429         return InternalParseResult::ok( ParseState( ParseResultType::NoMatch, remainingTokens ) );
27430     }
27431
27432     auto validate() const -> Result override {
27433         if( m_optNames.empty() )
27434             return Result::logicError( "No options supplied to Opt" );
27435         for( auto const &name : m_optNames ) {
27436             if( name.empty() )
27437                 return Result::logicError( "Option name cannot be empty" );
27438 #ifndef CATCH_PLATFORM_WINDOWS
27439             if( name[0] != '-' && name[0] != '/' )
27440                 return Result::logicError( "Option name must begin with '-' or '/'" );
27441 #else
27442             if( name[0] != '-' )
27443                 return Result::logicError( "Option name must begin with '-'" );
27444 #endif
27445         }
27446         return ParserRefImpl::validate();
27447     }
27448 };
27449
27450 struct Help : Opt {
27451     Help( bool &showHelpFlag )
27452     :   Opt([&]( bool flag ) {
27453         showHelpFlag = flag;
27454         return ParserResult::ok( ParseResultType::ShortCircuitAll );
27455     })
27456     {
27457         static_cast<Opt &>( *this )
27458             ("display usage information")
27459             ["-?"] ["-h"] ["--help"]
27460             .optional();
27461     }
27462 };
27463
27464 struct Parser : ParserBase {
27465
27466     mutable ExeName m_exeName;
27467     std::vector<Opt> m_options;
27468     std::vector<Arg> m_args;
27469
27470     auto operator|=( ExeName const &exeName ) -> Parser & {
27471         m_exeName = exeName;
27472         return *this;
27473     }
27474
27475     auto operator|=( Arg const &arg ) -> Parser & {
27476         m_args.push_back(arg);
27477         return *this;
27478     }
27479
27480     auto operator|=( Opt const &opt ) -> Parser & {
27481         m_options.push_back(opt);
27482         return *this;
27483     }
27484
27485     auto operator|=( Parser const &other ) -> Parser & {
27486         m_options.insert(m_options.end(), other.m_options.begin(), other.m_options.end());
27487         m_args.insert(m_args.end(), other.m_args.begin(), other.m_args.end());
27488         return *this;
27489     }
27490
27491     template<typename T>
27492     auto operator|( T const &other ) const -> Parser {
27493         return Parser( *this ) |= other;
27494     }
27495
27496     // Forward deprecated interface with '+' instead of '|'
27497     template<typename T>
27498     auto operator+=( T const &other ) -> Parser & { return operator|=( other ); }
27499     template<typename T>
27500     auto operator+( T const &other ) const -> Parser { return operator|( other ); }
27501
27502     auto getHelpColumns() const -> std::vector<HelpColumns> {
27503         std::vector<HelpColumns> cols;
27504         for( auto const &o : m_options ) {
27505             auto childCols = o.getHelpColumns();
27506             cols.insert( cols.end(), childCols.begin(), childCols.end() );
27507         }
27508         return cols;
27509     }
27510
27511     void writeToStream( std::ostream &os ) const {
27512         if (!m_exeName.name().empty()) {
27513             os << "usage:\n" << " " << m_exeName.name() << " ";
27514             bool required = true, first = true;

```

```

27515         for( auto const &arg : m_args ) {
27516             if (first)
27517                 first = false;
27518             else
27519                 os << " ";
27520             if( arg.isOptional() && required ) {
27521                 os << "[";
27522                 required = false;
27523             }
27524             os << "<" << arg.hint() << ">";
27525             if( arg.cardinality() == 0 )
27526                 os << " ... ";
27527         }
27528         if( !required )
27529             os << "]";
27530         if( !m_options.empty() )
27531             os << " options";
27532         os << "\n\nwhere options are:" << std::endl;
27533     }
27534
27535     auto rows = getHelpColumns();
27536     size_t consoleWidth = CATCH_CLARA_CONFIG_CONSOLE_WIDTH;
27537     size_t optWidth = 0;
27538     for( auto const &cols : rows )
27539         optWidth = (std::max)(optWidth, cols.left.size() + 2);
27540
27541     optWidth = (std::min)(optWidth, consoleWidth/2);
27542
27543     for( auto const &cols : rows ) {
27544         auto row =
27545             TextFlow::Column( cols.left ).width( optWidth ).indent( 2 ) +
27546             TextFlow::Spacer(4) +
27547             TextFlow::Column( cols.right ).width( consoleWidth - 7 - optWidth );
27548         os << row << std::endl;
27549     }
27550 }
27551
27552 friend auto operator<( std::ostream &os, Parser const &parser ) -> std::ostream& {
27553     parser.writeToStream( os );
27554     return os;
27555 }
27556
27557 auto validate() const -> Result override {
27558     for( auto const &opt : m_options ) {
27559         auto result = opt.validate();
27560         if( !result )
27561             return result;
27562     }
27563     for( auto const &arg : m_args ) {
27564         auto result = arg.validate();
27565         if( !result )
27566             return result;
27567     }
27568     return Result::ok();
27569 }
27570
27571 using ParserBase::parse;
27572
27573 auto parse( std::string const& exeName, TokenStream const &tokens ) const ->
InternalParseResult override {
27574
27575     struct ParserInfo {
27576         ParserBase const* parser = nullptr;
27577         size_t count = 0;
27578     };
27579     const size_t totalParsers = m_options.size() + m_args.size();
27580     assert( totalParsers < 512 );
27581     // ParserInfo parseInfos[totalParsers]; // <-- this is what we really want to do
27582     ParserInfo parseInfos[512];
27583
27584     {
27585         size_t i = 0;
27586         for (auto const &opt : m_options) parseInfos[i++].parser = &opt;
27587         for (auto const &arg : m_args) parseInfos[i++].parser = &arg;
27588     }
27589
27590     m_exeName.set( exeName );
27591
27592     auto result = InternalParseResult::ok( ParseState( ParseResultType::NoMatch, tokens ) );
27593     while( result.value().remainingTokens() ) {
27594         bool tokenParsed = false;
27595
27596         for( size_t i = 0; i < totalParsers; ++i ) {
27597             auto& parseInfo = parseInfos[i];
27598             if( parseInfo.parser->cardinality() == 0 || parseInfo.count <
parseInfo.parser->cardinality() ) {
27599                 result = parseInfo.parser->parse( exeName, result.value().remainingTokens() );

```



```

27600             if (!result)
27601                 return result;
27602             if (result.value().type() != ParseResultType::NoMatch) {
27603                 tokenParsed = true;
27604                 ++parseInfo.count;
27605                 break;
27606             }
27607         }
27608     }
27609
27610     if( result.value().type() == ParseResultType::ShortCircuitAll )
27611         return result;
27612     if( !tokenParsed )
27613         return InternalParseResult::runtimeError( "Unrecognised token: " +
result.value().remainingTokens()->token );
27614     }
27615     // !TBD Check missing required options
27616     return result;
27617 }
27618 };
27619
27620 template<typename DerivedT>
27621 template<typename T>
27622 auto ComposableParserImpl<DerivedT>::operator|( T const &other ) const -> Parser {
27623     return Parser() | static_cast<DerivedT const &>( *this ) | other;
27624 }
27625 } // namespace detail
27626
27627 // A Combined parser
27628 using detail::Parser;
27629
27630 // A parser for options
27631 using detail::Opt;
27632
27633 // A parser for arguments
27634 using detail::Arg;
27635
27636 // Wrapper for argc, argv from main()
27637 using detail::Args;
27638
27639 // Specifies the name of the executable
27640 using detail::ExeName;
27641
27642 // Convenience wrapper for option parser that specifies the help option
27643 using detail::Help;
27644
27645 // enum of result types from a parse
27646 using detail::ParseResultType;
27647
27648 // Result type for parser operation
27649 using detail::ParserResult;
27650
27651 }} // namespace Catch::clara
27652
27653 // end clara.hpp
27654 #ifdef __clang__
27655 #pragma clang diagnostic pop
27656 #endif
27657
27658 // Restore Clara's value for console width, if present
27659 #ifdef CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
27660 #define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
27661 #undef CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
27662 #endif
27663
27664 // end catch_clara.h
27665 namespace Catch {
27666
27667     clara::Parser makeCommandLineParser( ConfigData& config );
27668
27669 } // end namespace Catch
27670
27671 // end catch_commandline.h
27672 #include <fstream>
27673 #include <ctime>
27674
27675 namespace Catch {
27676
27677     clara::Parser makeCommandLineParser( ConfigData& config ) {
27678
27679         using namespace clara;
27680
27681         auto const setWarning = [&]( std::string const& warning ) {
27682             auto warningSet = [&]() {
27683                 if( warning == "NoAssertions" )
27684                     return WarnAbout::NoAssertions;
27685

```

```

27686         if ( warning == "NoTests" )
27687             return WarnAbout::NoTests;
27688
27689         return WarnAbout::Nothing;
27690     }();
27691
27692     if (warningSet == WarnAbout::Nothing)
27693         return ParserResult::runtimeError( "Unrecognised warning: '" + warning + "'" );
27694     config.warnings = static_cast<WarnAbout::What>( config.warnings | warningSet );
27695     return ParserResult::ok( ParseResultType::Matched );
27696 };
27697 auto const loadTestNamesFromFile = [&]( std::string const& filename ) {
27698     std::ifstream f( filename.c_str() );
27699     if( !f.is_open() )
27700         return ParserResult::runtimeError( "Unable to load input file: '" + filename + "'"
);
27701
27702     std::string line;
27703     while( std::getline( f, line ) ) {
27704         line = trim(line);
27705         if( !line.empty() && !startsWith( line, '#' ) ) {
27706             if( !startsWith( line, '"' ) )
27707                 line = '"' + line + '"';
27708             config.testsOrTags.push_back( line );
27709             config.testsOrTags.emplace_back( "," );
27710         }
27711     }
27712     //Remove comma in the end
27713     if(!config.testsOrTags.empty())
27714         config.testsOrTags.erase( config.testsOrTags.end()-1 );
27715
27716     return ParserResult::ok( ParseResultType::Matched );
27717 };
27718 auto const setTestOrder = [&]( std::string const& order ) {
27719     if( startsWith( "declared", order ) )
27720         config.runOrder = RunTests::InDeclarationOrder;
27721     else if( startsWith( "lexical", order ) )
27722         config.runOrder = RunTests::InLexicographicalOrder;
27723     else if( startsWith( "random", order ) )
27724         config.runOrder = RunTests::InRandomOrder;
27725     else
27726         return clara::ParserResult::runtimeError( "Unrecognised ordering: '" + order + "'"
);
27727     return ParserResult::ok( ParseResultType::Matched );
27728 };
27729 auto const setRngSeed = [&]( std::string const& seed ) {
27730     if( seed != "time" )
27731         return clara::detail::convertInto( seed, config.rngSeed );
27732     config.rngSeed = static_cast<unsigned int>( std::time(nullptr) );
27733     return ParserResult::ok( ParseResultType::Matched );
27734 };
27735 auto const setColourUsage = [&]( std::string const& useColour ) {
27736     auto mode = toLower( useColour );
27737
27738     if( mode == "yes" )
27739         config.useColour = UseColour::Yes;
27740     else if( mode == "no" )
27741         config.useColour = UseColour::No;
27742     else if( mode == "auto" )
27743         config.useColour = UseColour::Auto;
27744     else
27745         return ParserResult::runtimeError( "colour mode must be one of: auto, yes or
no. '" + useColour + "' not recognised" );
27746     return ParserResult::ok( ParseResultType::Matched );
27747 };
27748 auto const setWaitForKeypress = [&]( std::string const& keypress ) {
27749     auto keypressLc = toLower( keypress );
27750     if( keypressLc == "never" )
27751         config.waitForKeypress = WaitForKeypress::Never;
27752     else if( keypressLc == "start" )
27753         config.waitForKeypress = WaitForKeypress::BeforeStart;
27754     else if( keypressLc == "exit" )
27755         config.waitForKeypress = WaitForKeypress::BeforeExit;
27756     else if( keypressLc == "both" )
27757         config.waitForKeypress = WaitForKeypress::BeforeStartAndExit;
27758     else
27759         return ParserResult::runtimeError( "keypress argument must be one of: never,
start, exit or both. '" + keypress + "' not recognised" );
27760     return ParserResult::ok( ParseResultType::Matched );
27761 };
27762 auto const setVerbosity = [&]( std::string const& verbosity ) {
27763     auto lcVerbosity = toLower( verbosity );
27764     if( lcVerbosity == "quiet" )
27765         config.verbosity = Verbosity::Quiet;
27766     else if( lcVerbosity == "normal" )
27767         config.verbosity = Verbosity::Normal;
27768     else if( lcVerbosity == "high" )

```

```

27769         config.verbosity = Verbosity::High;
27770     else
27771         return ParserResult::runtimeError( "Unrecognised verbosity, '" + verbosity + "'" );
27772     return ParserResult::ok( ParseResultType::Matched );
27773 };
27774 auto const setReporter = [&]( std::string const& reporter ) {
27775     IReporterRegistry::FactoryMap const& factories =
27776         getRegistryHub().getReporterRegistry().getFactories();
27777     auto lcReporter = toLower( reporter );
27778     auto result = factories.find( lcReporter );
27779     if( factories.end() != result )
27780         config.reporterName = lcReporter;
27781     else
27782         return ParserResult::runtimeError( "Unrecognized reporter, '" + reporter + "'. Check
27783         available with --list-reporters" );
27784     return ParserResult::ok( ParseResultType::Matched );
27785 };
27786
27787 auto cli
27788     = ExeName( config.processName )
27789     | Help( config.showHelp )
27790     | Opt( config.listTests )
27791         [ "-l" ][ "--list-tests" ]
27792         ( "list all/matching test cases" )
27793     | Opt( config.listTags )
27794         [ "-t" ][ "--list-tags" ]
27795         ( "list all/matching tags" )
27796     | Opt( config.showSuccessfulTests )
27797         [ "-s" ][ "--success" ]
27798         ( "include successful tests in output" )
27799     | Opt( config.shouldDebugBreak )
27800         [ "-b" ][ "--break" ]
27801         ( "break into debugger on failure" )
27802     | Opt( config.noThrow )
27803         [ "-e" ][ "--nothrow" ]
27804         ( "skip exception tests" )
27805     | Opt( config.showInvisibles )
27806         [ "-i" ][ "--invisibles" ]
27807         ( "show invisibles (tabs, newlines)" )
27808     | Opt( config.outputFilename, "filename" )
27809         [ "-o" ][ "--out" ]
27810         ( "output filename" )
27811     | Opt( setReporter, "name" )
27812         [ "-r" ][ "--reporter" ]
27813         ( "reporter to use (defaults to console)" )
27814     | Opt( config.name, "name" )
27815         [ "-n" ][ "--name" ]
27816         ( "suite name" )
27817     | Opt( [&]( bool ){ config.abortAfter = 1; } )
27818         [ "-a" ][ "--abort" ]
27819         ( "abort at first failure" )
27820     | Opt( [&]( int x ){ config.abortAfter = x; }, "no. failures" )
27821         [ "-x" ][ "--abortx" ]
27822         ( "abort after x failures" )
27823     | Opt( setWarning, "warning name" )
27824         [ "-w" ][ "--warn" ]
27825         ( "enable warnings" )
27826     | Opt( [&]( bool flag ) { config.showDurations = flag ? ShowDurations::Always :
27827         ShowDurations::Never; }, "yes|no" )
27828         [ "-d" ][ "--durations" ]
27829         ( "show test durations" )
27830     | Opt( config.minDuration, "seconds" )
27831         [ "-D" ][ "--min-duration" ]
27832         ( "show test durations for tests taking at least the given number of seconds" )
27833     | Opt( loadTestNamesFromFile, "filename" )
27834         [ "-f" ][ "--input-file" ]
27835         ( "load test names to run from a file" )
27836     | Opt( config.fileNamesAsTags )
27837         [ "-#" ][ "--filenames-as-tags" ]
27838         ( "adds a tag for the filename" )
27839     | Opt( config.sectionsToRun, "section name" )
27840         [ "-c" ][ "--section" ]
27841         ( "specify section to run" )
27842     | Opt( setVerbosity, "quiet|normal|high" )
27843         [ "-v" ][ "--verbosity" ]
27844         ( "set output verbosity" )
27845     | Opt( config.listTestNamesOnly )
27846         [ "--list-test-names-only" ]
27847         ( "list all/matching test cases names only" )
27848     | Opt( config.listReporters )
27849         [ "--list-reporters" ]
27850         ( "list all reporters" )
27851     | Opt( setTestOrder, "decl|lex|rand" )
27852         [ "--order" ]
27853         ( "test case order (defaults to decl)" )

```

```

27853         | Opt( setRngSeed, "time'|number" )
27854             [ "--rng-seed" ]
27855             ( "set a specific seed for random numbers" )
27856         | Opt( setColourUsage, "yes|no" )
27857             [ "--use-colour" ]
27858             ( "should output be coloured" )
27859         | Opt( config.libIdentify )
27860             [ "--libidentify" ]
27861             ( "report name and version according to libidentify standard" )
27862         | Opt( setWaitForKeypress, "never|start|exit|both" )
27863             [ "--wait-for-keypress" ]
27864             ( "waits for a keypress before exiting" )
27865         | Opt( config.benchmarkSamples, "samples" )
27866             [ "--benchmark-samples" ]
27867             ( "number of samples to collect (default: 100)" )
27868         | Opt( config.benchmarkResamples, "resamples" )
27869             [ "--benchmark-resamples" ]
27870             ( "number of resamples for the bootstrap (default: 100000)" )
27871         | Opt( config.benchmarkConfidenceInterval, "confidence interval" )
27872             [ "--benchmark-confidence-interval" ]
27873             ( "confidence interval for the bootstrap (between 0 and 1, default: 0.95)" )
27874         | Opt( config.benchmarkNoAnalysis )
27875             [ "--benchmark-no-analysis" ]
27876             ( "perform only measurements; do not perform any analysis" )
27877         | Opt( config.benchmarkWarmupTime, "benchmarkWarmupTime" )
27878             [ "--benchmark-warmup-time" ]
27879             ( "amount of time in milliseconds spent on warming up each test (default: 100)" )
27880         | Arg( config.testsOrTags, "test name|pattern|tags" )
27881             ( "which test or tests to use" );
27882
27883     return cli;
27884 }
27885
27886 } // end namespace Catch
27887 // end catch_commandline.cpp
27888 // start catch_common.cpp
27889
27890 #include <cstring>
27891 #include <ostream>
27892
27893 namespace Catch {
27894
27895     bool SourceLineInfo::operator == ( SourceLineInfo const& other ) const noexcept {
27896         return line == other.line && ( file == other.file || std::strcmp(file, other.file) == 0 );
27897     }
27898     bool SourceLineInfo::operator < ( SourceLineInfo const& other ) const noexcept {
27899         // We can assume that the same file will usually have the same pointer.
27900         // Thus, if the pointers are the same, there is no point in calling the strcmp
27901         return line < other.line || ( line == other.line && file != other.file && (std::strcmp(file,
27902 other.file) < 0) );
27903     }
27904
27905     std::ostream& operator << ( std::ostream& os, SourceLineInfo const& info ) {
27906 #ifndef __GNUG__
27907         os << info.file << '(' << info.line << ')';
27908 #else
27909         os << info.file << ':' << info.line;
27910 #endif
27911         return os;
27912     }
27913
27914     std::string StreamEndStop::operator+() const {
27915         return std::string();
27916     }
27917
27918     NonCopyable::NonCopyable() = default;
27919     NonCopyable::~NonCopyable() = default;
27920 }
27921 // end catch_common.cpp
27922 // start catch_config.cpp
27923
27924 namespace Catch {
27925
27926     Config::Config( ConfigData const& data )
27927     :   m_data( data ),
27928         m_stream( openStream() )
27929     {
27930         // We need to trim filter specs to avoid trouble with superfluous
27931         // whitespace (esp. important for bdd macros, as those are manually
27932         // aligned with whitespace).
27933
27934         for ( auto& elem : m_data.testsOrTags ) {
27935             elem = trim(elem);
27936         }
27937         for ( auto& elem : m_data.sectionsToRun ) {
27938             elem = trim(elem);

```

```

27939     }
27940
27941     TestSpecParser parser(ITagAliasRegistry::get());
27942     if (!m_data.testsOrTags.empty()) {
27943         m_hasTestFilters = true;
27944         for (auto const& testOrTags : m_data.testsOrTags) {
27945             parser.parse(testOrTags);
27946         }
27947     }
27948     m_testSpec = parser.testSpec();
27949 }
27950
27951 std::string const& Config::getFilename() const {
27952     return m_data.outputFilename;
27953 }
27954
27955 bool Config::listTests() const { return m_data.listTests; }
27956 bool Config::listTestNamesOnly() const { return m_data.listTestNamesOnly; }
27957 bool Config::listTags() const { return m_data.listTags; }
27958 bool Config::listReporters() const { return m_data.listReporters; }
27959
27960 std::string Config::getProcessName() const { return m_data.processName; }
27961 std::string const& Config::getReporterName() const { return m_data.reporterName; }
27962
27963 std::vector<std::string> const& Config::getTestsOrTags() const { return m_data.testsOrTags; }
27964 std::vector<std::string> const& Config::getSectionsToRun() const { return m_data.sectionsToRun; }
27965
27966 TestSpec const& Config::testSpec() const { return m_testSpec; }
27967 bool Config::hasTestFilters() const { return m_hasTestFilters; }
27968
27969 bool Config::showHelp() const { return m_data.showHelp; }
27970
27971 // IConfig interface
27972 bool Config::allowThrows() const { return !m_data.noThrow; }
27973 std::ostream& Config::stream() const { return m_stream->stream(); }
27974 std::string Config::name() const { return m_data.name.empty() ?
m_data.processName : m_data.name; }
27975 bool Config::includeSuccessfulResults() const { return m_data.showSuccessfulTests; }
27976 bool Config::warnAboutMissingAssertions() const { return !(m_data.warnings &
WarnAbout::NoAssertions); }
27977 bool Config::warnAboutNoTests() const { return !(m_data.warnings &
WarnAbout::NoTests); }
27978 ShowDurations::OrNot Config::showDurations() const { return m_data.showDurations; }
27979 double Config::minDuration() const { return m_data.minDuration; }
27980 RunTests::InWhatOrder Config::runOrder() const { return m_data.runOrder; }
27981 unsigned int Config::rngSeed() const { return m_data.rngSeed; }
27982 UseColour::YesOrNo Config::useColour() const { return m_data.useColour; }
27983 bool Config::shouldDebugBreak() const { return m_data.shouldDebugBreak; }
27984 int Config::abortAfter() const { return m_data.abortAfter; }
27985 bool Config::showInvisibles() const { return m_data.showInvisibles; }
27986 Verbosity Config::verbosity() const { return m_data.verbosity; }
27987
27988 bool Config::benchmarkNoAnalysis() const { return m_data.benchmarkNoAnalysis; }
27989 int Config::benchmarkSamples() const { return m_data.benchmarkSamples; }
27990 double Config::benchmarkConfidenceInterval() const { return m_data.benchmarkConfidenceInterval; }
27991 unsigned int Config::benchmarkResamples() const { return m_data.benchmarkResamples; }
27992
27993 std::chrono::milliseconds Config::benchmarkWarmupTime() const { return
std::chrono::milliseconds(m_data.benchmarkWarmupTime); }
27994
27995 IStream const* Config::openStream() {
27996     return Catch::makeStream(m_data.outputFilename);
27997 }
27998 } // end namespace Catch
27999 // end catch_config.cpp
28000 // start catch_console_colour.cpp
28001
28002 #if defined(__clang__)
28003 # pragma clang diagnostic push
28004 # pragma clang diagnostic ignored "-Wexit-time-destructors"
28005 #endif
28006
28007 // start catch_errno_guard.h
28008
28009 namespace Catch {
28010
28011     class ErrnoGuard {
28012     public:
28013         ErrnoGuard();
28014         ~ErrnoGuard();
28015     private:
28016         int m_oldErrno;
28017     };
28018

```

```

28019 }
28020
28021 // end catch_errno_guard.h
28022 // start catch_windows_h_proxy.h
28023
28024
28025 #if defined(CATCH_PLATFORM_WINDOWS)
28026
28027 #if !defined(NOMINMAX) && !defined(CATCH_CONFIG_NO_NOMINMAX)
28028 #   define CATCH_DEFINED_NOMINMAX
28029 #   define NOMINMAX
28030 #endif
28031 #if !defined(WIN32_LEAN_AND_MEAN) && !defined(CATCH_CONFIG_NO_WIN32_LEAN_AND_MEAN)
28032 #   define CATCH_DEFINED_WIN32_LEAN_AND_MEAN
28033 #   define WIN32_LEAN_AND_MEAN
28034 #endif
28035
28036 #ifdef __AFXDLL
28037 #include <AfxWin.h>
28038 #else
28039 #include <windows.h>
28040 #endif
28041
28042 #ifdef CATCH_DEFINED_NOMINMAX
28043 #   undef NOMINMAX
28044 #endif
28045 #ifdef CATCH_DEFINED_WIN32_LEAN_AND_MEAN
28046 #   undef WIN32_LEAN_AND_MEAN
28047 #endif
28048
28049 #endif // defined(CATCH_PLATFORM_WINDOWS)
28050
28051 // end catch_windows_h_proxy.h
28052 #include <sstream>
28053
28054 namespace Catch {
28055     namespace {
28056
28057         struct IColourImpl {
28058             virtual ~IColourImpl() = default;
28059             virtual void use( Colour::Code _colourCode ) = 0;
28060         };
28061
28062         struct NoColourImpl : IColourImpl {
28063             void use( Colour::Code ) override {}
28064
28065             static IColourImpl* instance() {
28066                 static NoColourImpl s_instance;
28067                 return &s_instance;
28068             }
28069         };
28070
28071     } // anon namespace
28072 } // namespace Catch
28073
28074 #if !defined( CATCH_CONFIG_COLOUR_NONE ) && !defined( CATCH_CONFIG_COLOUR_WINDOWS ) && !defined(
CATCH_CONFIG_COLOUR_ANSI )
28075 #   ifdef CATCH_PLATFORM_WINDOWS
28076 #       define CATCH_CONFIG_COLOUR_WINDOWS
28077 #   else
28078 #       define CATCH_CONFIG_COLOUR_ANSI
28079 #   endif
28080 #endif
28081
28082 #if defined ( CATCH_CONFIG_COLOUR_WINDOWS )
28083
28084 namespace Catch {
28085     namespace {
28086
28087         class Win32ColourImpl : public IColourImpl {
28088         public:
28089             Win32ColourImpl() : stdoutHandle( GetStdHandle(STD_OUTPUT_HANDLE) )
28090             {
28091                 CONSOLE_SCREEN_BUFFER_INFO csbiInfo;
28092                 GetConsoleScreenBufferInfo( stdoutHandle, &csbiInfo );
28093                 originalForegroundAttributes = csbiInfo.wAttributes & ~( BACKGROUND_GREEN | BACKGROUND_RED
| BACKGROUND_BLUE | BACKGROUND_INTENSITY );
28094                 originalBackgroundAttributes = csbiInfo.wAttributes & ~( FOREGROUND_GREEN | FOREGROUND_RED
| FOREGROUND_BLUE | FOREGROUND_INTENSITY );
28095             }
28096
28097             void use( Colour::Code _colourCode ) override {
28098                 switch( _colourCode ) {
28099                     case Colour::None:         return setTextAttribute( originalForegroundAttributes );
28100                     case Colour::White:        return setTextAttribute( FOREGROUND_GREEN | FOREGROUND_RED |
28101 FOREGROUND_BLUE );
28102                     case Colour::Red:          return setTextAttribute( FOREGROUND_RED );

```

```

28102         case Colour::Green:      return setTextAttribute( FOREGROUND_GREEN );
28103         case Colour::Blue:       return setTextAttribute( FOREGROUND_BLUE );
28104         case Colour::Cyan:       return setTextAttribute( FOREGROUND_BLUE | FOREGROUND_GREEN );
28105         case Colour::Yellow:     return setTextAttribute( FOREGROUND_RED | FOREGROUND_GREEN );
28106         case Colour::Grey:       return setTextAttribute( 0 );
28107
28108         case Colour::LightGrey:   return setTextAttribute( FOREGROUND_INTENSITY );
28109         case Colour::BrightRed:   return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_RED );
28110         case Colour::BrightGreen: return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_GREEN );
28111         case Colour::BrightWhite: return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_GREEN | FOREGROUND_RED | FOREGROUND_BLUE );
28112         case Colour::BrightYellow: return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_RED | FOREGROUND_GREEN );
28113
28114         case Colour::Bright: CATCH_INTERNAL_ERROR( "not a colour" );
28115
28116         default:
28117             CATCH_ERROR( "Unknown colour requested" );
28118     }
28119 }
28120
28121 private:
28122     void setTextAttribute( WORD _textAttribute ) {
28123         SetConsoleTextAttribute( stdoutHandle, _textAttribute | originalBackgroundAttributes );
28124     }
28125     HANDLE stdoutHandle;
28126     WORD originalForegroundAttributes;
28127     WORD originalBackgroundAttributes;
28128 };
28129
28130 IColourImpl* platformColourInstance() {
28131     static Win32ColourImpl s_instance;
28132
28133     IConfigPtr config = getCurrentContext().getConfig();
28134     UseColour::YesOrNo colourMode = config
? config->useColour()
: UseColour::Auto;
28136     if( colourMode == UseColour::Auto )
28137         colourMode = UseColour::Yes;
28138     return colourMode == UseColour::Yes
? &s_instance
: NoColourImpl::instance();
28141 }
28142
28143
28144 } // end anon namespace
28145 } // end namespace Catch
28146
28147 #elif defined( CATCH_CONFIG_COLOUR_ANSI )
28148
28149 #include <unistd.h>
28150
28151 namespace Catch {
28152 namespace {
28153
28154     // use POSIX/ ANSI console terminal codes
28155     // Thanks to Adam Strzelecki for original contribution
28156     // (http://github.com/nanoant)
28157     // https://github.com/philsquared/Catch/pull/131
28158     class PosixColourImpl : public IColourImpl {
28159     public:
28160         void use( Colour::Code _colourCode ) override {
28161             switch( _colourCode ) {
28162                 case Colour::None:
28163                 case Colour::White:      return setColour( "[0m" );
28164                 case Colour::Red:       return setColour( "[0;31m" );
28165                 case Colour::Green:     return setColour( "[0;32m" );
28166                 case Colour::Blue:      return setColour( "[0;34m" );
28167                 case Colour::Cyan:      return setColour( "[0;36m" );
28168                 case Colour::Yellow:    return setColour( "[0;33m" );
28169                 case Colour::Grey:      return setColour( "[1;30m" );
28170
28171                 case Colour::LightGrey: return setColour( "[0;37m" );
28172                 case Colour::BrightRed: return setColour( "[1;31m" );
28173                 case Colour::BrightGreen: return setColour( "[1;32m" );
28174                 case Colour::BrightWhite: return setColour( "[1;37m" );
28175                 case Colour::BrightYellow: return setColour( "[1;33m" );
28176
28177                 case Colour::Bright: CATCH_INTERNAL_ERROR( "not a colour" );
28178                 default: CATCH_INTERNAL_ERROR( "Unknown colour requested" );
28179             }
28180         }
28181         static IColourImpl* instance() {
28182             static PosixColourImpl s_instance;
28183             return &s_instance;
28184         }
28185     };

```

```

28185
28186     private:
28187         void setColour( const char* _escapeCode ) {
28188             getCurrentContext().getConfig()->stream()
28189                 « '\033' « _escapeCode;
28190         }
28191     };
28192
28193     bool useColourOnPlatform() {
28194         return
28195 #if defined(CATCH_PLATFORM_MAC) || defined(CATCH_PLATFORM_IPHONE)
28196             !isDebuggerActive() &&
28197 #endif
28198 #if !(defined(__DJGPP__) && defined(__STRICT_ANSI__))
28199             isatty(STDOUT_FILENO)
28200 #else
28201             false
28202 #endif
28203         ;
28204     }
28205     IColourImpl* platformColourInstance() {
28206         ErrnoGuard guard;
28207         IConfigPtr config = getCurrentContext().getConfig();
28208         UseColour::YesOrNo colourMode = config
28209             ? config->useColour()
28210             : UseColour::Auto;
28211         if( colourMode == UseColour::Auto )
28212             colourMode = useColourOnPlatform()
28213                 ? UseColour::Yes
28214                 : UseColour::No;
28215         return colourMode == UseColour::Yes
28216             ? PosixColourImpl::instance()
28217             : NoColourImpl::instance();
28218     }
28219 } // end anon namespace
28220 } // end namespace Catch
28221
28222 #else // not Windows or ANSI //////////////////////////////////////
28223 namespace Catch {
28224
28225     static IColourImpl* platformColourInstance() { return NoColourImpl::instance(); }
28226 } // end namespace Catch
28227
28228 #endif // Windows/ ANSI/ None
28229
28230 namespace Catch {
28231
28232     Colour::Colour( Code _colourCode ) { use( _colourCode ); }
28233     Colour::Colour( Colour&& other ) noexcept {
28234         m_moved = other.m_moved;
28235         other.m_moved = true;
28236     }
28237     Colour& Colour::operator=( Colour&& other ) noexcept {
28238         m_moved = other.m_moved;
28239         other.m_moved = true;
28240         return *this;
28241     }
28242
28243     Colour::~Colour(){ if( !m_moved ) use( None ); }
28244
28245     void Colour::use( Code _colourCode ) {
28246         static IColourImpl* impl = platformColourInstance();
28247         // Strictly speaking, this cannot possibly happen.
28248         // However, under some conditions it does happen (see #1626),
28249         // and this change is small enough that we can let practicality
28250         // triumph over purity in this case.
28251         if (impl != nullptr) {
28252             impl->use( _colourCode );
28253         }
28254     }
28255
28256     std::ostream& operator << ( std::ostream& os, Colour const& ) {
28257         return os;
28258     }
28259 } // end namespace Catch
28260
28261 #if defined(__clang__)
28262 #pragma clang diagnostic pop
28263 #endif
28264 // end catch_console_colour.cpp
28265 // start catch_context.cpp
28266

```



```

28272 namespace Catch {
28273
28274     class Context : public IMutableContext, NonCopyable {
28275
28276     public: // IContext
28277         IResultCapture* getResultCapture() override {
28278             return m_resultCapture;
28279         }
28280         IRunner* getRunner() override {
28281             return m_runner;
28282         }
28283
28284         IConfigPtr const& getConfig() const override {
28285             return m_config;
28286         }
28287
28288         ~Context() override;
28289
28290     public: // IMutableContext
28291         void setResultCapture( IResultCapture* resultCapture ) override {
28292             m_resultCapture = resultCapture;
28293         }
28294         void setRunner( IRunner* runner ) override {
28295             m_runner = runner;
28296         }
28297         void setConfig( IConfigPtr const& config ) override {
28298             m_config = config;
28299         }
28300
28301         friend IMutableContext& getCurrentMutableContext();
28302
28303     private:
28304         IConfigPtr m_config;
28305         IRunner* m_runner = nullptr;
28306         IResultCapture* m_resultCapture = nullptr;
28307     };
28308
28309     IMutableContext *IMutableContext::currentContext = nullptr;
28310
28311     void IMutableContext::createContext()
28312     {
28313         currentContext = new Context();
28314     }
28315
28316     void cleanUpContext() {
28317         delete IMutableContext::currentContext;
28318         IMutableContext::currentContext = nullptr;
28319     }
28320     IContext::~IContext() = default;
28321     IMutableContext::~IMutableContext() = default;
28322     Context::~Context() = default;
28323
28324     SimplePcg32& rng() {
28325         static SimplePcg32 s_rng;
28326         return s_rng;
28327     }
28328
28329 }
28330 // end catch_context.cpp
28331 // start catch_debug_console.cpp
28332
28333 // start catch_debug_console.h
28334
28335 #include <string>
28336
28337 namespace Catch {
28338     void writeToDebugConsole( std::string const& text );
28339 }
28340
28341 // end catch_debug_console.h
28342 #if defined(CATCH_CONFIG_ANDROID_LOGWRITE)
28343 #include <android/log.h>
28344
28345     namespace Catch {
28346         void writeToDebugConsole( std::string const& text ) {
28347             __android_log_write( ANDROID_LOG_DEBUG, "Catch", text.c_str() );
28348         }
28349     }
28350
28351 #elif defined(CATCH_PLATFORM_WINDOWS)
28352
28353     namespace Catch {
28354         void writeToDebugConsole( std::string const& text ) {
28355             ::OutputDebugStringA( text.c_str() );
28356         }
28357     }
28358

```

```

28359 #else
28360
28361     namespace Catch {
28362         void writeToDebugConsole( std::string const& text ) {
28363             // !TBD: Need a version for Mac/ XCode and other IDEs
28364             Catch::cout() << text;
28365         }
28366     }
28367
28368 #endif // Platform
28369 // end catch_debug_console.cpp
28370 // start catch_debugger.cpp
28371
28372 #if defined(CATCH_PLATFORM_MAC) || defined(CATCH_PLATFORM_IPHONE)
28373
28374 # include <assert>
28375 # include <sys/types.h>
28376 # include <unistd.h>
28377 # include <cstdint>
28378 # include <ostream>
28379
28380 #ifdef __apple_build_version__
28381     // These headers will only compile with AppleClang (XCode)
28382     // For other compilers (Clang, GCC, ... ) we need to exclude them
28383 # include <sys/sysctl.h>
28384 #endif
28385
28386     namespace Catch {
28387         #ifdef __apple_build_version__
28388             // The following function is taken directly from the following technical note:
28389             // https://developer.apple.com/library/archive/qa/qa1361/_index.html
28390
28391             // Returns true if the current process is being debugged (either
28392             // running under the debugger or has a debugger attached post facto).
28393             bool isDebuggerActive() {
28394                 int mib[4];
28395                 struct kinfo_proc info;
28396                 std::size_t size;
28397
28398                 // Initialize the flags so that, if sysctl fails for some bizarre
28399                 // reason, we get a predictable result.
28400
28401                 info.kp_proc.p_flag = 0;
28402
28403                 // Initialize mib, which tells sysctl the info we want, in this case
28404                 // we're looking for information about a specific process ID.
28405
28406                 mib[0] = CTL_KERN;
28407                 mib[1] = KERN_PROC;
28408                 mib[2] = KERN_PROC_PID;
28409                 mib[3] = getpid();
28410
28411                 // Call sysctl.
28412
28413                 size = sizeof(info);
28414                 if( sysctl(mib, sizeof(mib) / sizeof(*mib), &info, &size, nullptr, 0) != 0 ) {
28415                     Catch::cerr() << "\n** Call to sysctl failed - unable to determine if debugger is
active **\n" << std::endl;
28416                     return false;
28417                 }
28418
28419                 // We're being debugged if the P_TRACED flag is set.
28420
28421                 return ( (info.kp_proc.p_flag & P_TRACED) != 0 );
28422             }
28423         #else
28424             bool isDebuggerActive() {
28425                 // We need to find another way to determine this for non-appleclang compilers on macOS
28426                 return false;
28427             }
28428         #endif
28429     } // namespace Catch
28430
28431 #elif defined(CATCH_PLATFORM_LINUX)
28432     #include <fstream>
28433     #include <string>
28434
28435     namespace Catch{
28436         // The standard POSIX way of detecting a debugger is to attempt to
28437         // ptrace() the process, but this needs to be done from a child and not
28438         // this process itself to still allow attaching to this process later
28439         // if wanted, so is rather heavy. Under Linux we have the PID of the
28440         // "debugger" (which doesn't need to be gdb, of course, it could also
28441         // be strace, for example) in /proc/$PID/status, so just get it from
28442         // there instead.
28443         bool isDebuggerActive(){
28444             // Libstdc++ has a bug, where std::ifstream sets errno to 0

```

```

28445         // This way our users can properly assert over errno values
28446         ErrnoGuard guard;
28447         std::ifstream in("/proc/self/status");
28448         for( std::string line; std::getline(in, line); ) {
28449             static const int PREFIX_LEN = 11;
28450             if( line.compare(0, PREFIX_LEN, "TracerPid:") == 0 ) {
28451                 // We're traced if the PID is not 0 and no other PID starts
28452                 // with 0 digit, so it's enough to check for just a single
28453                 // character.
28454                 return line.length() > PREFIX_LEN && line[PREFIX_LEN] != '0';
28455             }
28456         }
28457         return false;
28458     }
28459 } // namespace Catch
28460 #elif defined(_MSC_VER)
28461 extern "C" __declspec(dllimport) int __stdcall IsDebuggerPresent();
28462 namespace Catch {
28463     bool isDebuggerActive() {
28464         return IsDebuggerPresent() != 0;
28465     }
28466 }
28467 #elif defined(__MINGW32__)
28468 extern "C" __declspec(dllimport) int __stdcall IsDebuggerPresent();
28469 namespace Catch {
28470     bool isDebuggerActive() {
28471         return IsDebuggerPresent() != 0;
28472     }
28473 }
28474 #else
28475 namespace Catch {
28476     bool isDebuggerActive() { return false; }
28477 }
28478 #endif // Platform
28479 // end catch_debugger.cpp
28480 // start catch_decomposer.cpp
28481 namespace Catch {
28482     ITransientExpression::~ITransientExpression() = default;
28483     void formatReconstructedExpression( std::ostream &os, std::string const& lhs, StringRef op,
28484         std::string const& rhs ) {
28485         if( lhs.size() + rhs.size() < 40 &&
28486             lhs.find('\n') == std::string::npos &&
28487             rhs.find('\n') == std::string::npos )
28488             os << lhs << " " << op << " " << rhs;
28489         else
28490             os << lhs << "\n" << op << "\n" << rhs;
28491     }
28492 // end catch_decomposer.cpp
28493 // start catch_enforce.cpp
28494 #include <stdexcept>
28495 namespace Catch {
28496     #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS) &&
28497         !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS_CUSTOM_HANDLER)
28498     [[noreturn]]
28499     void throw_exception(std::exception const& e) {
28500         Catch::cerr() << "Catch will terminate because it needed to throw an exception.\n"
28501             << "The message was: " << e.what() << '\n';
28502         std::terminate();
28503     }
28504 #endif
28505 [[noreturn]]
28506 void throw_logic_error(std::string const& msg) {
28507     throw_exception(std::logic_error(msg));
28508 }
28509 [[noreturn]]
28510 void throw_domain_error(std::string const& msg) {
28511     throw_exception(std::domain_error(msg));
28512 }
28513 [[noreturn]]
28514 void throw_runtime_error(std::string const& msg) {
28515     throw_exception(std::runtime_error(msg));
28516 }
28517 } // namespace Catch;
28518 // end catch_enforce.cpp
28519 // start catch_enum_values_registry.cpp
28520 // start catch_enum_values_registry.h

```

```

28530
28531 #include <vector>
28532 #include <memory>
28533
28534 namespace Catch {
28535
28536     namespace Detail {
28537
28538         std::unique_ptr<EnumInfo> makeEnumInfo( StringRef enumName, StringRef allValueNames,
28539 std::vector<int> const& values );
28540
28541         class EnumValuesRegistry : public IMutableEnumValuesRegistry {
28542
28543             std::vector<std::unique_ptr<EnumInfo>> m_enumInfos;
28544
28545             EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums, std::vector<int>
28546 const& values) override;
28547
28548             std::vector<StringRef> parseEnums( StringRef enums );
28549         } // Detail
28550
28551     } // Catch
28552
28553 // end catch_enum_values_registry.h
28554
28555 #include <map>
28556 #include <cassert>
28557
28558 namespace Catch {
28559
28560     IMutableEnumValuesRegistry::~IMutableEnumValuesRegistry() {}
28561
28562     namespace Detail {
28563
28564         namespace {
28565             // Extracts the actual name part of an enum instance
28566             // In other words, it returns the Blue part of Bikeshed::Colour::Blue
28567             StringRef extractInstanceName(StringRef enumInstance) {
28568                 // Find last occurrence of ":"
28569                 size_t name_start = enumInstance.size();
28570                 while (name_start > 0 && enumInstance[name_start - 1] != ':') {
28571                     --name_start;
28572                 }
28573                 return enumInstance.substr(name_start, enumInstance.size() - name_start);
28574             }
28575         }
28576
28577         std::vector<StringRef> parseEnums( StringRef enums ) {
28578             auto enumValues = splitStringRef( enums, ',' );
28579             std::vector<StringRef> parsed;
28580             parsed.reserve( enumValues.size() );
28581             for( auto const& enumValue : enumValues ) {
28582                 parsed.push_back(trim(extractInstanceName(enumValue)));
28583             }
28584             return parsed;
28585         }
28586
28587         EnumInfo::~EnumInfo() {}
28588
28589         StringRef EnumInfo::lookup( int value ) const {
28590             for( auto const& valueToName : m_values ) {
28591                 if( valueToName.first == value )
28592                     return valueToName.second;
28593             }
28594             return "{** unexpected enum value **}"_sr;
28595         }
28596
28597         std::unique_ptr<EnumInfo> makeEnumInfo( StringRef enumName, StringRef allValueNames,
28598 std::vector<int> const& values ) {
28599             std::unique_ptr<EnumInfo> enumInfo( new EnumInfo );
28600             enumInfo->m_name = enumName;
28601             enumInfo->m_values.reserve( values.size() );
28602
28603             const auto valueNames = Catch::Detail::parseEnums( allValueNames );
28604             assert( valueNames.size() == values.size() );
28605             std::size_t i = 0;
28606             for( auto value : values )
28607                 enumInfo->m_values.emplace_back(value, valueNames[i++]);
28608
28609             return enumInfo;
28610         }
28611
28612         EnumInfo const& EnumValuesRegistry::registerEnum( StringRef enumName, StringRef allValueNames,
28613 std::vector<int> const& values ) {
28614             m_enumInfos.push_back( makeEnumInfo( enumName, allValueNames, values ) );

```

```

28613         return *m_enumInfos.back();
28614     }
28615
28616     } // Detail
28617 } // Catch
28618
28619 // end catch_enum_values_registry.cpp
28620 // start catch_errno_guard.cpp
28621
28622 #include <cerrno>
28623
28624 namespace Catch {
28625     ErrnoGuard::ErrnoGuard(): m_oldErrno(errno){}
28626     ErrnoGuard::~ErrnoGuard() { errno = m_oldErrno; }
28627 }
28628 // end catch_errno_guard.cpp
28629 // start catch_exception_translator_registry.cpp
28630
28631 // start catch_exception_translator_registry.h
28632
28633 #include <vector>
28634 #include <string>
28635 #include <memory>
28636
28637 namespace Catch {
28638
28639     class ExceptionTranslatorRegistry : public IExceptionTranslatorRegistry {
28640     public:
28641         ~ExceptionTranslatorRegistry();
28642         virtual void registerTranslator( const IExceptionTranslator* translator );
28643         std::string translateActiveException() const override;
28644         std::string tryTranslators() const;
28645
28646     private:
28647         std::vector<std::unique_ptr<IExceptionTranslator const> m_translators;
28648     };
28649 }
28650
28651 // end catch_exception_translator_registry.h
28652 #ifdef __OBJC__
28653 #import "Foundation/Foundation.h"
28654 #endif
28655
28656 namespace Catch {
28657
28658     ExceptionTranslatorRegistry::~ExceptionTranslatorRegistry() {
28659     }
28660
28661     void ExceptionTranslatorRegistry::registerTranslator( const IExceptionTranslator* translator ) {
28662         m_translators.push_back( std::unique_ptr<const IExceptionTranslator>( translator ) );
28663     }
28664
28665     #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
28666     std::string ExceptionTranslatorRegistry::translateActiveException() const {
28667         try {
28668             #ifdef __OBJC__
28669                 // In Objective-C try objective-c exceptions first
28670                 @try {
28671                     return tryTranslators();
28672                 }
28673                 @catch (NSEException *exception) {
28674                     return Catch::Detail::stringify( [exception description] );
28675                 }
28676             #else
28677                 // Compiling a mixed mode project with MSVC means that CLR
28678                 // exceptions will be caught in (...) as well. However, these
28679                 // do not fill-in std::current_exception and thus lead to crash
28680                 // when attempting rethrow.
28681                 // /EHa switch also causes structured exceptions to be caught
28682                 // here, but they fill-in current_exception properly, so
28683                 // at worst the output should be a little weird, instead of
28684                 // causing a crash.
28685                 if (std::current_exception() == nullptr) {
28686                     return "Non C++ exception. Possibly a CLR exception.";
28687                 }
28688                 return tryTranslators();
28689             #endif
28690         }
28691         catch( TestFailureException& ) {
28692             std::rethrow_exception(std::current_exception());
28693         }
28694         catch( std::exception& ex ) {
28695             return ex.what();
28696         }
28697         catch( std::string& msg ) {
28698             return msg;
28699         }
28700     }

```

```

28700         catch( const char* msg ) {
28701             return msg;
28702         }
28703         catch(...) {
28704             return "Unknown exception";
28705         }
28706     }
28707
28708     std::string ExceptionTranslatorRegistry::tryTranslators() const {
28709         if (m_translators.empty()) {
28710             std::rethrow_exception(std::current_exception());
28711         } else {
28712             return m_translators[0]->translate(m_translators.begin() + 1, m_translators.end());
28713         }
28714     }
28715
28716 #else // ^^ Exceptions are enabled // Exceptions are disabled vv
28717     std::string ExceptionTranslatorRegistry::translateActiveException() const {
28718         CATCH_INTERNAL_ERROR("Attempted to translate active exception under
28719         CATCH_CONFIG_DISABLE_EXCEPTIONS!");
28720     }
28721
28722     std::string ExceptionTranslatorRegistry::tryTranslators() const {
28723         CATCH_INTERNAL_ERROR("Attempted to use exception translators under
28724         CATCH_CONFIG_DISABLE_EXCEPTIONS!");
28725     }
28726 #endif
28727 // end catch_exception_translator_registry.cpp
28728 // start catch_fatal_condition.cpp
28729
28730 #include <algorithm>
28731
28732 #if !defined( CATCH_CONFIG_WINDOWS_SEH ) && !defined( CATCH_CONFIG_POSIX_SIGNALS )
28733
28734 namespace Catch {
28735
28736     // If neither SEH nor signal handling is required, the handler impls
28737     // do not have to do anything, and can be empty.
28738     void FatalConditionHandler::engage_platform() {}
28739     void FatalConditionHandler::disengage_platform() {}
28740     FatalConditionHandler::FatalConditionHandler() = default;
28741     FatalConditionHandler::~FatalConditionHandler() = default;
28742
28743 } // end namespace Catch
28744
28745 #endif // !CATCH_CONFIG_WINDOWS_SEH && !CATCH_CONFIG_POSIX_SIGNALS
28746
28747 #if defined( CATCH_CONFIG_WINDOWS_SEH ) && defined( CATCH_CONFIG_POSIX_SIGNALS )
28748 #error "Inconsistent configuration: Windows' SEH handling and POSIX signals cannot be enabled at the
28749 same time"
28750 #endif
28751 #if defined( CATCH_CONFIG_WINDOWS_SEH ) || defined( CATCH_CONFIG_POSIX_SIGNALS )
28752
28753 namespace {
28754     void reportFatal( char const * const message ) {
28755         Catch::getCurrentContext().getResultCapture()->handleFatalErrorCondition( message );
28756     }
28757 }
28758
28759 constexpr std::size_t minStackSizeForErrors = 32 * 1024;
28760 } // end unnamed namespace
28761
28762 #endif // CATCH_CONFIG_WINDOWS_SEH || CATCH_CONFIG_POSIX_SIGNALS
28763
28764 #if defined( CATCH_CONFIG_WINDOWS_SEH )
28765 namespace Catch {
28766
28767     struct SignalDefs { DWORD id; const char* name; };
28768
28769     // There is no 1-1 mapping between signals and windows exceptions.
28770     // Windows can easily distinguish between SO and SigSegV,
28771     // but SigInt, SigTerm, etc are handled differently.
28772     static SignalDefs signalDefs[] = {
28773         { static_cast<DWORD>(EXCEPTION_ILLEGAL_INSTRUCTION), "SIGILL - Illegal instruction signal" },
28774         { static_cast<DWORD>(EXCEPTION_STACK_OVERFLOW), "SIGSEGV - Stack overflow" },
28775         { static_cast<DWORD>(EXCEPTION_ACCESS_VIOLATION), "SIGSEGV - Segmentation violation signal" },
28776         { static_cast<DWORD>(EXCEPTION_INT_DIVIDE_BY_ZERO), "Divide by zero error" },
28777     };
28778
28779     static LONG CALLBACK handleVectoredException(PEXCEPTION_POINTERS ExceptionInfo) {
28780         for (auto const& def : signalDefs) {
28781             if (ExceptionInfo->ExceptionRecord->ExceptionCode == def.id) {
28782                 reportFatal(def.name);
28783             }
28784         }
28785     }

```

```

28788     }
28789     // If its not an exception we care about, pass it along.
28790     // This stops us from eating debugger breaks etc.
28791     return EXCEPTION_CONTINUE_SEARCH;
28792 }
28793
28794 // Since we do not support multiple instantiations, we put these
28795 // into global variables and rely on cleaning them up in outlined
28796 // constructors/destructors
28797 static PVOID exceptionHandlerHandle = nullptr;
28798
28799 // For MSVC, we reserve part of the stack memory for handling
28800 // memory overflow structured exception.
28801 FatalConditionHandler::FatalConditionHandler() {
28802     ULONG guaranteeSize = static_cast<ULONG>(minStackSizeForErrors);
28803     if (!SetThreadStackGuarantee(&guaranteeSize)) {
28804         // We do not want to fully error out, because needing
28805         // the stack reserve should be rare enough anyway.
28806         Catch::cerr()
28807             << "Failed to reserve piece of stack."
28808             << " Stack overflows will not be reported successfully.";
28809     }
28810 }
28811
28812 // We do not attempt to unset the stack guarantee, because
28813 // Windows does not support lowering the stack size guarantee.
28814 FatalConditionHandler::~FatalConditionHandler() = default;
28815
28816 void FatalConditionHandler::engage_platform() {
28817     // Register as first handler in current chain
28818     exceptionHandlerHandle = AddVectoredExceptionHandler(1, handleVectoredException);
28819     if (!exceptionHandlerHandle) {
28820         CATCH_RUNTIME_ERROR("Could not register vectored exception handler");
28821     }
28822 }
28823
28824 void FatalConditionHandler::disengage_platform() {
28825     if (!RemoveVectoredExceptionHandler(exceptionHandlerHandle)) {
28826         CATCH_RUNTIME_ERROR("Could not unregister vectored exception handler");
28827     }
28828     exceptionHandlerHandle = nullptr;
28829 }
28830
28831 } // end namespace Catch
28832
28833 #endif // CATCH_CONFIG_WINDOWS_SEH
28834
28835 #if defined( CATCH_CONFIG_POSIX_SIGNALS )
28836
28837 #include <signal.h>
28838
28839 namespace Catch {
28840
28841     struct SignalDefs {
28842         int id;
28843         const char* name;
28844     };
28845
28846     static SignalDefs signalDefs[] = {
28847         { SIGINT, "SIGINT - Terminal interrupt signal" },
28848         { SIGILL, "SIGILL - Illegal instruction signal" },
28849         { SIGFPE, "SIGFPE - Floating point error signal" },
28850         { SIGSEGV, "SIGSEGV - Segmentation violation signal" },
28851         { SIGTERM, "SIGTERM - Termination request signal" },
28852         { SIGABRT, "SIGABRT - Abort (abnormal termination) signal" }
28853     };
28854
28855     // Older GCCs trigger -Wmissing-field-initializers for T foo = {}
28856     // which is zero initialization, but not explicit. We want to avoid
28857     // that.
28858     #if defined(__GNUC__)
28859     # pragma GCC diagnostic push
28860     # pragma GCC diagnostic ignored "-Wmissing-field-initializers"
28861     #endif
28862
28863     static char* altStackMem = nullptr;
28864     static std::size_t altStackSize = 0;
28865     static stack_t oldSigStack{};
28866     static struct sigaction oldSigActions[sizeof(signalDefs) / sizeof(SignalDefs)]{};
28867
28868     static void restorePreviousSignalHandlers() {
28869         // We set signal handlers back to the previous ones. Hopefully
28870         // nobody overwrote them in the meantime, and doesn't expect
28871         // their signal handlers to live past ours given that they
28872         // installed them after ours..
28873         for (std::size_t i = 0; i < sizeof(signalDefs) / sizeof(SignalDefs); ++i) {
28874             sigaction(signalDefs[i].id, &oldSigActions[i], nullptr);

```

```

28875     }
28876     // Return the old stack
28877     sigaltstack(&oldSigStack, nullptr);
28878 }
28879
28880 static void handleSignal( int sig ) {
28881     char const * name = "<unknown signal>";
28882     for (auto const& def : signalDefs) {
28883         if (sig == def.id) {
28884             name = def.name;
28885             break;
28886         }
28887     }
28888     // We need to restore previous signal handlers and let them do
28889     // their thing, so that the users can have the debugger break
28890     // when a signal is raised, and so on.
28891     restorePreviousSignalHandlers();
28892     reportFatal( name );
28893     raise( sig );
28894 }
28895
28896 FatalConditionHandler::FatalConditionHandler() {
28897     assert(!altStackMem && "Cannot initialize POSIX signal handler when one already exists");
28898     if (altStackSize == 0) {
28899         altStackSize = std::max(static_cast<size_t>(SIGSTKSZ), minStackSizeForErrors);
28900     }
28901     altStackMem = new char[altStackSize]();
28902 }
28903
28904 FatalConditionHandler::~FatalConditionHandler() {
28905     delete[] altStackMem;
28906     // We signal that another instance can be constructed by zeroing
28907     // out the pointer.
28908     altStackMem = nullptr;
28909 }
28910
28911 void FatalConditionHandler::engage_platform() {
28912     stack_t sigStack;
28913     sigStack.ss_sp = altStackMem;
28914     sigStack.ss_size = altStackSize;
28915     sigStack.ss_flags = 0;
28916     sigaltstack(&sigStack, &oldSigStack);
28917     struct sigaction sa = { };
28918
28919     sa.sa_handler = handleSignal;
28920     sa.sa_flags = SA_ONSTACK;
28921     for (std::size_t i = 0; i < sizeof(signalDefs)/sizeof(SignalDefs); ++i) {
28922         sigaction(signalDefs[i].id, &sa, &oldSigActions[i]);
28923     }
28924 }
28925
28926 #if defined(__GNUC__)
28927 #pragma GCC diagnostic pop
28928 #endif
28929
28930 void FatalConditionHandler::disengage_platform() {
28931     restorePreviousSignalHandlers();
28932 }
28933
28934 } // end namespace Catch
28935
28936 #endif // CATCH_CONFIG_POSIX_SIGNALS
28937 // end catch_fatal_condition.cpp
28938 // start catch_generators.cpp
28939
28940 #include <limits>
28941 #include <set>
28942
28943 namespace Catch {
28944
28945 IGeneratorTracker::~IGeneratorTracker() {}
28946
28947 const char* GeneratorException::what() const noexcept {
28948     return m_msg;
28949 }
28950
28951 namespace Generators {
28952
28953 GeneratorUntypedBase::~GeneratorUntypedBase() {}
28954
28955 auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
IGeneratorTracker& {
28956     return getResultCapture().acquireGeneratorTracker( generatorName, lineInfo );
28957 }
28958
28959 } // namespace Generators
28960 } // namespace Catch

```



```

28961 // end catch_generators.cpp
28962 // start catch_interfaces_capture.cpp
28963
28964 namespace Catch {
28965     IResultCapture::~IResultCapture() = default;
28966 }
28967 // end catch_interfaces_capture.cpp
28968 // start catch_interfaces_config.cpp
28969
28970 namespace Catch {
28971     IConfig::~IConfig() = default;
28972 }
28973 // end catch_interfaces_config.cpp
28974 // start catch_interfaces_exception.cpp
28975
28976 namespace Catch {
28977     IExceptionTranslator::~IExceptionTranslator() = default;
28978     IExceptionTranslatorRegistry::~IExceptionTranslatorRegistry() = default;
28979 }
28980 // end catch_interfaces_exception.cpp
28981 // start catch_interfaces_registry_hub.cpp
28982
28983 namespace Catch {
28984     IRegistryHub::~IRegistryHub() = default;
28985     IMutableRegistryHub::~IMutableRegistryHub() = default;
28986 }
28987 // end catch_interfaces_registry_hub.cpp
28988 // start catch_interfaces_reporter.cpp
28989
28990 // start catch_reporter_listening.h
28991
28992 namespace Catch {
28993
28994     class ListeningReporter : public IStreamingReporter {
28995     public:
28996         using Reporters = std::vector<IStreamingReporterPtr>;
28997         Reporters m_listeners;
28998         IStreamingReporterPtr m_reporter = nullptr;
28999         ReporterPreferences m_preferences;
29000
29001         ListeningReporter();
29002
29003         void addListener( IStreamingReporterPtr&& listener );
29004         void addReporter( IStreamingReporterPtr&& reporter );
29005
29006     public: // IStreamingReporter
29007         ReporterPreferences getPreferences() const override;
29008
29009         void noMatchingTestCases( std::string const& spec ) override;
29010
29011         void reportInvalidArguments( std::string const& arg ) override;
29012
29013         static std::set<Verbosity> getSupportedVerbsities();
29014
29015     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
29016         void benchmarkPreparing( std::string const& name ) override;
29017         void benchmarkStarting( BenchmarkInfo const& benchmarkInfo ) override;
29018         void benchmarkEnded( BenchmarkStats<> const& benchmarkStats ) override;
29019         void benchmarkFailed( std::string const& ) override;
29020     #endif
29021     #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
29022
29023         void testRunStarting( TestRunInfo const& testRunInfo ) override;
29024         void testGroupStarting( GroupInfo const& groupInfo ) override;
29025         void testCaseStarting( TestCaseInfo const& testInfo ) override;
29026         void sectionStarting( SectionInfo const& sectionInfo ) override;
29027         void assertionStarting( AssertionInfo const& assertionInfo ) override;
29028
29029         // The return value indicates if the messages buffer should be cleared:
29030         bool assertionEnded( AssertionStats const& assertionStats ) override;
29031         void sectionEnded( SectionStats const& sectionStats ) override;
29032         void testCaseEnded( TestCaseStats const& testCaseStats ) override;
29033         void testGroupEnded( TestGroupStats const& testGroupStats ) override;
29034         void testRunEnded( TestRunStats const& testRunStats ) override;
29035
29036         void skipTest( TestCaseInfo const& testInfo ) override;
29037         bool isMulti() const override;
29038     };
29039
29040 }
29041 // end namespace Catch
29042
29043 // end catch_reporter_listening.h
29044 namespace Catch {
29045
29046     ReporterConfig::ReporterConfig( IConfigPtr const& _fullConfig )
29047     :   m_stream( &_fullConfig->stream() ), m_fullConfig( _fullConfig ) {}

```

```

29048
29049 ReporterConfig::ReporterConfig( IConfigPtr const& _fullConfig, std::ostream& _stream )
29050 :   m_stream( &_stream ), m_fullConfig( _fullConfig ) {}
29051
29052 std::ostream& ReporterConfig::stream() const { return *m_stream; }
29053 IConfigPtr ReporterConfig::fullConfig() const { return m_fullConfig; }
29054
29055 TestRunInfo::TestRunInfo( std::string const& _name ) : name( _name ) {}
29056
29057 GroupInfo::GroupInfo(   std::string const& _name,
29058                       std::size_t _groupIndex,
29059                       std::size_t _groupsCount )
29060 :   name( _name ),
29061   groupIndex( _groupIndex ),
29062   groupsCounts( _groupsCount )
29063 {}
29064
29065 AssertionStats::AssertionStats( AssertionResult const& _assertionResult,
29066                                std::vector<MessageInfo> const& _infoMessages,
29067                                Totals const& _totals )
29068 :   assertionResult( _assertionResult ),
29069   infoMessages( _infoMessages ),
29070   totals( _totals )
29071 {
29072     assertionResult.m_resultData.lazyExpression.m_transientExpression =
29073     _assertionResult.m_resultData.lazyExpression.m_transientExpression;
29074
29075     if( assertionResult.hasMessage() ) {
29076         // Copy message into messages list.
29077         // !TBD This should have been done earlier, somewhere
29078         MessageBuilder builder( assertionResult.getTestMacroName(),
29079                                assertionResult.getSourceInfo(), assertionResult.getResultType() );
29080         builder « assertionResult.getMessage();
29081         builder.m_info.message = builder.m_stream.str();
29082         infoMessages.push_back( builder.m_info );
29083     }
29084
29085     AssertionStats::~AssertionStats() = default;
29086
29087 SectionStats::SectionStats(   SectionInfo const& _sectionInfo,
29088                             Counts const& _assertions,
29089                             double _durationInSeconds,
29090                             bool _missingAssertions )
29091 :   sectionInfo( _sectionInfo ),
29092   assertions( _assertions ),
29093   durationInSeconds( _durationInSeconds ),
29094   missingAssertions( _missingAssertions )
29095 {}
29096
29097 SectionStats::~SectionStats() = default;
29098
29099 TestCaseStats::TestCaseStats(   TestCaseInfo const& _testInfo,
29100                               Totals const& _totals,
29101                               std::string const& _stdOut,
29102                               std::string const& _stdErr,
29103                               bool _aborting )
29104 :   testInfo( _testInfo ),
29105   totals( _totals ),
29106   stdOut( _stdOut ),
29107   stdErr( _stdErr ),
29108   aborting( _aborting )
29109 {}
29110
29111 TestCaseStats::~TestCaseStats() = default;
29112
29113 TestGroupStats::TestGroupStats(   GroupInfo const& _groupInfo,
29114                                 Totals const& _totals,
29115                                 bool _aborting )
29116 :   groupInfo( _groupInfo ),
29117   totals( _totals ),
29118   aborting( _aborting )
29119 {}
29120
29121 TestGroupStats::TestGroupStats(   GroupInfo const& _groupInfo )
29122 :   groupInfo( _groupInfo ),
29123   aborting( false )
29124 {}
29125
29126 TestGroupStats::~TestGroupStats() = default;
29127
29128 TestRunStats::TestRunStats(   TestRunInfo const& _runInfo,
29129                             Totals const& _totals,
29130                             bool _aborting )
29131 :   runInfo( _runInfo ),
29132   totals( _totals ),

```

```

29133         aborting( _aborting )
29134     {}
29135
29136     TestRunStats::~TestRunStats() = default;
29137
29138     void IStreamingReporter::fatalErrorEncountered( StringRef ) {}
29139     bool IStreamingReporter::isMulti() const { return false; }
29140
29141     IReporterFactory::~IReporterFactory() = default;
29142     IReporterRegistry::~IReporterRegistry() = default;
29143
29144 } // end namespace Catch
29145 // end catch_interfaces_reporter.cpp
29146 // start catch_interfaces_runner.cpp
29147
29148 namespace Catch {
29149     IRunner::~IRunner() = default;
29150 }
29151 // end catch_interfaces_runner.cpp
29152 // start catch_interfaces_testcase.cpp
29153
29154 namespace Catch {
29155     ITestInvoker::~ITestInvoker() = default;
29156     ITestCaseRegistry::~ITestCaseRegistry() = default;
29157 }
29158 // end catch_interfaces_testcase.cpp
29159 // start catch_leak_detector.cpp
29160
29161 #ifdef CATCH_CONFIG_WINDOWS_CRTDBG
29162 #include <crtdbg.h>
29163
29164 namespace Catch {
29165     LeakDetector::LeakDetector() {
29166         int flag = _CrtSetDbgFlag(_CRTDBG_REPORT_FLAG);
29167         flag |= _CRTDBG_LEAK_CHECK_DF;
29168         flag |= _CRTDBG_ALLOC_MEM_DF;
29169         _CrtSetDbgFlag(flag);
29170         _CrtSetReportMode(_CRT_WARN, _CRTDBG_MODE_FILE | _CRTDBG_MODE_DEBUG);
29171         _CrtSetReportFile(_CRT_WARN, _CRTDBG_FILE_STDERR);
29172         // Change this to leaking allocation's number to break there
29173         _CrtSetBreakAlloc(-1);
29174     }
29175 }
29176 #else
29177
29178 Catch::LeakDetector::LeakDetector() {}
29179 #endif
29180
29181 Catch::LeakDetector::~LeakDetector() {}
29182 #endif
29183
29184 Catch::LeakDetector::~LeakDetector() {
29185     Catch::cleanUp();
29186 }
29187 // end catch_leak_detector.cpp
29188 // start catch_list.cpp
29189
29190 // start catch_list.h
29191
29192 #include <set>
29193
29194 namespace Catch {
29195     std::size_t listTests( Config const& config );
29196
29197     std::size_t listTestsNamesOnly( Config const& config );
29198
29199     struct TagInfo {
29200         void add( std::string const& spelling );
29201         std::string all() const;
29202
29203         std::set<std::string> spellings;
29204         std::size_t count = 0;
29205     };
29206
29207     std::size_t listTags( Config const& config );
29208
29209     std::size_t listReporters();
29210
29211     Option<std::size_t> list( std::shared_ptr<Config> const& config );
29212 } // end namespace Catch
29213
29214 // end catch_list.h
29215 // start catch_text.h
29216
29217 namespace Catch {

```

```

29220     using namespace clara::TextFlow;
29221 }
29222
29223 // end catch_text.h
29224 #include <limits>
29225 #include <algorithm>
29226 #include <iomanip>
29227
29228 namespace Catch {
29229
29230     std::size_t listTests( Config const& config ) {
29231         TestSpec const& testSpec = config.testSpec();
29232         if( config.hasTestFilters() )
29233             Catch::cout() << "Matching test cases:\n";
29234         else {
29235             Catch::cout() << "All available test cases:\n";
29236         }
29237
29238         auto matchedTestCases = filterTests( getAllTestCasesSorted( config ), testSpec, config );
29239         for( auto const& testCaseInfo : matchedTestCases ) {
29240             Colour::Code colour = testCaseInfo.isHidden()
29241                 ? Colour::SecondaryText
29242                 : Colour::None;
29243             Colour colourGuard( colour );
29244
29245             Catch::cout() << Column( testCaseInfo.name ).initialIndent( 2 ).indent( 4 ) << "\n";
29246             if( config.verbosity() >= Verbosity::High ) {
29247                 Catch::cout() << Column( Catch::Detail::stringify( testCaseInfo.lineInfo ) ).indent(4)
29248                 << std::endl;
29249                 std::string description = testCaseInfo.description;
29250                 if( description.empty() )
29251                     description = "(NO DESCRIPTION)";
29252                 Catch::cout() << Column( description ).indent(4) << std::endl;
29253             }
29254             if( !testCaseInfo.tags.empty() )
29255                 Catch::cout() << Column( testCaseInfo.tagsAsString() ).indent( 6 ) << "\n";
29256         }
29257         if( !config.hasTestFilters() )
29258             Catch::cout() << pluralise( matchedTestCases.size(), "test case" ) << '\n' << std::endl;
29259         else
29260             Catch::cout() << pluralise( matchedTestCases.size(), "matching test case" ) << '\n' <<
29261             std::endl;
29262         return matchedTestCases.size();
29263     }
29264
29265     std::size_t listTestsNamesOnly( Config const& config ) {
29266         TestSpec const& testSpec = config.testSpec();
29267         std::size_t matchedTests = 0;
29268         std::vector<TestCases> matchedTestCases = filterTests( getAllTestCasesSorted( config ),
29269             testSpec, config );
29270         for( auto const& testCaseInfo : matchedTestCases ) {
29271             matchedTests++;
29272             if( startsWith( testCaseInfo.name, '#' ) )
29273                 Catch::cout() << "' " << testCaseInfo.name << "' ";
29274             else
29275                 Catch::cout() << testCaseInfo.name;
29276             if ( config.verbosity() >= Verbosity::High )
29277                 Catch::cout() << "\t@" << testCaseInfo.lineInfo;
29278             Catch::cout() << std::endl;
29279         }
29280         return matchedTests;
29281     }
29282
29283     void TagInfo::add( std::string const& spelling ) {
29284         ++count;
29285         spellings.insert( spelling );
29286     }
29287
29288     std::string TagInfo::all() const {
29289         std::size_t size = 0;
29290         for (auto const& spelling : spellings) {
29291             // Add 2 for the brackets
29292             size += spelling.size() + 2;
29293         }
29294         std::string out; out.reserve(size);
29295         for (auto const& spelling : spellings) {
29296             out += '[';
29297             out += spelling;
29298             out += ']';
29299         }
29300         return out;
29301     }
29302
29303     std::size_t listTags( Config const& config ) {
29304         TestSpec const& testSpec = config.testSpec();

```

```

29304         if( config.hasTestFilters() )
29305             Catch::cout() << "Tags for matching test cases:\n";
29306         else {
29307             Catch::cout() << "All available tags:\n";
29308         }
29309
29310         std::map<std::string, TagInfo> tagCounts;
29311
29312         std::vector<TestCase> matchedTestCases = filterTests( getAllTestCasesSorted( config ),
testSpec, config );
29313         for( auto const& testCase : matchedTestCases ) {
29314             for( auto const& tagName : testCase.getTestCaseInfo().tags ) {
29315                 std::string lcaseTagName = toLower( tagName );
29316                 auto countIt = tagCounts.find( lcaseTagName );
29317                 if( countIt == tagCounts.end() )
29318                     countIt = tagCounts.insert( std::make_pair( lcaseTagName, TagInfo() ) ).first;
29319                 countIt->second.add( tagName );
29320             }
29321         }
29322
29323         for( auto const& tagCount : tagCounts ) {
29324             ReusableStringStream rss;
29325             rss << " " << std::setw(2) << tagCount.second.count << " ";
29326             auto str = rss.str();
29327             auto wrapper = Column( tagCount.second.all() )
29328                 .initialIndent( 0 )
29329                 .indent( str.size() )
29330                 .width( CATCH_CONFIG_CONSOLE_WIDTH-10 );
29331             Catch::cout() << str << wrapper << '\n';
29332         }
29333         Catch::cout() << pluralise( tagCounts.size(), "tag" ) << '\n' << std::endl;
29334         return tagCounts.size();
29335     }
29336
29337     std::size_t listReporters() {
29338         Catch::cout() << "Available reporters:\n";
29339         IReporterRegistry::FactoryMap const& factories =
getRegistryHub().getReporterRegistry().getFactories();
29340         std::size_t maxNameLen = 0;
29341         for( auto const& factoryKvp : factories )
29342             maxNameLen = (std::max)( maxNameLen, factoryKvp.first.size() );
29343
29344         for( auto const& factoryKvp : factories ) {
29345             Catch::cout()
29346                 << Column( factoryKvp.first + ":" )
29347                     .indent( 2 )
29348                     .width( 5+maxNameLen )
29349                 + Column( factoryKvp.second->getDescription() )
29350                     .initialIndent( 0 )
29351                     .indent( 2 )
29352                     .width( CATCH_CONFIG_CONSOLE_WIDTH - maxNameLen-8 )
29353                 << "\n";
29354         }
29355         Catch::cout() << std::endl;
29356         return factories.size();
29357     }
29358
29359     Option<std::size_t> list( std::shared_ptr<Config> const& config ) {
29360         Option<std::size_t> listedCount;
29361         getCurrentMutableContext().setConfig( config );
29362         if( config->listTests() )
29363             listedCount = listedCount.valueOr(0) + listTests( *config );
29364         if( config->listTestNamesOnly() )
29365             listedCount = listedCount.valueOr(0) + listTestsNamesOnly( *config );
29366         if( config->listTags() )
29367             listedCount = listedCount.valueOr(0) + listTags( *config );
29368         if( config->listReporters() )
29369             listedCount = listedCount.valueOr(0) + listReporters();
29370         return listedCount;
29371     }
29372
29373 } // end namespace Catch
29374 // end catch_list.cpp
29375 // start catch_matchers.cpp
29376
29377 namespace Catch {
29378     namespace Matchers {
29379         namespace Impl {
29380
29381             std::string MatcherUntypedBase::toString() const {
29382                 if( m_cachedToString.empty() )
29383                     m_cachedToString = describe();
29384                 return m_cachedToString;
29385             }
29386
29387             MatcherUntypedBase::~MatcherUntypedBase() = default;
29388

```

```

29389     } // namespace Impl
29390 } // namespace Matchers
29391
29392 using namespace Matchers;
29393 using Matchers::Impl::MatcherBase;
29394
29395 } // namespace Catch
29396 // end catch_matchers.cpp
29397 // start catch_matchers_exception.cpp
29398
29399 namespace Catch {
29400 namespace Matchers {
29401 namespace Exception {
29402
29403 bool ExceptionMessageMatcher::match(std::exception const& ex) const {
29404     return ex.what() == m_message;
29405 }
29406
29407 std::string ExceptionMessageMatcher::describe() const {
29408     return "exception message matches \"" + m_message + "\"";
29409 }
29410
29411 }
29412 Exception::ExceptionMessageMatcher Message(std::string const& message) {
29413     return Exception::ExceptionMessageMatcher(message);
29414 }
29415
29416 // namespace Exception
29417 } // namespace Matchers
29418 } // namespace Catch
29419 // end catch_matchers_exception.cpp
29420 // start catch_matchers_floating.cpp
29421
29422 // start catch_polyfills.hpp
29423
29424 namespace Catch {
29425     bool isnan(float f);
29426     bool isnan(double d);
29427 }
29428
29429 // end catch_polyfills.hpp
29430 // start catch_to_string.hpp
29431
29432 #include <string>
29433
29434 namespace Catch {
29435     template <typename T>
29436     std::string to_string(T const& t) {
29437 #if defined(CATCH_CONFIG_CPP11_TO_STRING)
29438         return std::to_string(t);
29439 #else
29440         ReusableStringStream rss;
29441         rss << t;
29442         return rss.str();
29443 #endif
29444     }
29445 } // end namespace Catch
29446
29447 // end catch_to_string.hpp
29448 #include <algorithm>
29449 #include <cmath>
29450 #include <cstdlib>
29451 #include <stdint>
29452 #include <cstring>
29453 #include <sstream>
29454 #include <type_traits>
29455 #include <iomanip>
29456 #include <limits>
29457
29458 namespace Catch {
29459 namespace {
29460
29461     int32_t convert(float f) {
29462         static_assert(sizeof(float) == sizeof(int32_t), "Important ULP matcher assumption violated");
29463         int32_t i;
29464         std::memcpy(&i, &f, sizeof(f));
29465         return i;
29466     }
29467
29468     int64_t convert(double d) {
29469         static_assert(sizeof(double) == sizeof(int64_t), "Important ULP matcher assumption violated");
29470         int64_t i;
29471         std::memcpy(&i, &d, sizeof(d));
29472         return i;
29473     }
29474
29475     template <typename FP>

```

```

29476     bool almostEqualUlp(FP lhs, FP rhs, uint64_t maxUlpDiff) {
29477         // Comparison with NaN should always be false.
29478         // This way we can rule it out before getting into the ugly details
29479         if (Catch::isnan(lhs) || Catch::isnan(rhs)) {
29480             return false;
29481         }
29482
29483         auto lc = convert(lhs);
29484         auto rc = convert(rhs);
29485
29486         if ((lc < 0) != (rc < 0)) {
29487             // Potentially we can have +0 and -0
29488             return lhs == rhs;
29489         }
29490
29491         // static cast as a workaround for IBM XLC
29492         auto ulpDiff = std::abs(static_cast<FP>(lc - rc));
29493         return static_cast<uint64_t>(ulpDiff) <= maxUlpDiff;
29494     }
29495
29496 #if defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
29497     float nextafter(float x, float y) {
29498         return ::nextafterf(x, y);
29499     }
29500
29501     double nextafter(double x, double y) {
29502         return ::nextafter(x, y);
29503     }
29504 }
29505
29506 #endif // ^^^ CATCH_CONFIG_GLOBAL_NEXTAFTER ^^^
29507
29508 template <typename FP>
29509 FP step(FP start, FP direction, uint64_t steps) {
29510     for (uint64_t i = 0; i < steps; ++i) {
29511 #if defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
29512         start = Catch::nextafter(start, direction);
29513 #else
29514         start = std::nextafter(start, direction);
29515 #endif
29516     }
29517     return start;
29518 }
29519
29520 // Performs equivalent check of std::fabs(lhs - rhs) <= margin
29521 // But without the subtraction to allow for INFINITY in comparison
29522 bool marginComparison(double lhs, double rhs, double margin) {
29523     return (lhs + margin >= rhs) && (rhs + margin >= lhs);
29524 }
29525
29526 template <typename FloatingPoint>
29527 void write(std::ostream& out, FloatingPoint num) {
29528     out << std::scientific
29529         << std::setprecision(std::numeric_limits<FloatingPoint>::max_digits10 - 1)
29530         << num;
29531 }
29532
29533 } // end anonymous namespace
29534
29535 namespace Matchers {
29536 namespace Floating {
29537
29538     enum class FloatingPointKind : uint8_t {
29539         Float,
29540         Double
29541     };
29542
29543     WithinAbsMatcher::WithinAbsMatcher(double target, double margin)
29544         : m_target{ target }, m_margin{ margin } {
29545         CATCH_ENFORCE(margin >= 0, "Invalid margin: " << margin << ".")
29546         << " Margin has to be non-negative.";
29547     }
29548
29549     // Performs equivalent check of std::fabs(lhs - rhs) <= margin
29550     // But without the subtraction to allow for INFINITY in comparison
29551     bool WithinAbsMatcher::match(double const& matchee) const {
29552         return (matchee + m_margin >= m_target) && (m_target + m_margin >= matchee);
29553     }
29554
29555     std::string WithinAbsMatcher::describe() const {
29556         return "is within " + ::Catch::Detail::stringify(m_margin) + " of " +
29557             ::Catch::Detail::stringify(m_target);
29558     }
29559
29560     WithinUlpMatcher::WithinUlpMatcher(double target, uint64_t ulps, FloatingPointKind baseType)
29561         : m_target{ target }, m_ulps{ ulps }, m_type{ baseType } {
29562         CATCH_ENFORCE(m_type == FloatingPointKind::Double

```

```

29562         || m_ulps < (std::numeric_limits<uint32_t>::max)(),
29563         "Provided ULP is impossibly large for a float comparison.");
29564     }
29565
29566 #if defined(__clang__)
29567 #pragma clang diagnostic push
29568 // Clang <3.5 reports on the default branch in the switch below
29569 #pragma clang diagnostic ignored "-Wunreachable-code"
29570 #endif
29571
29572     bool WithinUlpMatcher::match(double const& matchee) const {
29573         switch (m_type) {
29574             case FloatingPointKind::Float:
29575                 return almostEqualUlp<float>(static_cast<float>(matchee), static_cast<float>(m_target),
m_ulps);
29576             case FloatingPointKind::Double:
29577                 return almostEqualUlp<double>(matchee, m_target, m_ulps);
29578             default:
29579                 CATCH_INTERNAL_ERROR( "Unknown FloatingPointKind value" );
29580         }
29581     }
29582
29583 #if defined(__clang__)
29584 #pragma clang diagnostic pop
29585 #endif
29586
29587     std::string WithinUlpMatcher::describe() const {
29588         std::stringstream ret;
29589
29590         ret << "is within " << m_ulps << " ULPs of ";
29591
29592         if (m_type == FloatingPointKind::Float) {
29593             write(ret, static_cast<float>(m_target));
29594             ret << 'f';
29595         } else {
29596             write(ret, m_target);
29597         }
29598
29599         ret << " [";
29600         if (m_type == FloatingPointKind::Double) {
29601             write(ret, step(m_target, static_cast<double>(-INFINITY), m_ulps));
29602             ret << ", ";
29603             write(ret, step(m_target, static_cast<double>( INFINITY), m_ulps));
29604         } else {
29605             // We have to cast INFINITY to float because of MinGW, see #1782
29606             write(ret, step(static_cast<float>(m_target), static_cast<float>(-INFINITY), m_ulps));
29607             ret << ", ";
29608             write(ret, step(static_cast<float>(m_target), static_cast<float>( INFINITY), m_ulps));
29609         }
29610         ret << " ]";
29611
29612         return ret.str();
29613     }
29614
29615     WithinRelMatcher::WithinRelMatcher(double target, double epsilon):
29616         m_target(target),
29617         m_epsilon(epsilon){
29618         CATCH_ENFORCE(m_epsilon >= 0., "Relative comparison with epsilon < 0 does not make sense.");
29619         CATCH_ENFORCE(m_epsilon < 1., "Relative comparison with epsilon >= 1 does not make sense.");
29620     }
29621
29622     bool WithinRelMatcher::match(double const& matchee) const {
29623         const auto relMargin = m_epsilon * (std::max)(std::fabs(matchee), std::fabs(m_target));
29624         return marginComparison(matchee, m_target,
29625                                 std::isinf(relMargin)? 0 : relMargin);
29626     }
29627
29628     std::string WithinRelMatcher::describe() const {
29629         Catch::ReusableStringStream sstr;
29630         sstr << "and " << m_target << " are within " << m_epsilon * 100. << "% of each other";
29631         return sstr.str();
29632     }
29633
29634 } // namespace Floating
29635
29636 Floating::WithinUlpMatcher WithinULP(double target, uint64_t maxUlpDiff) {
29637     return Floating::WithinUlpMatcher(target, maxUlpDiff, Floating::FloatingPointKind::Double);
29638 }
29639
29640 Floating::WithinUlpMatcher WithinULP(float target, uint64_t maxUlpDiff) {
29641     return Floating::WithinUlpMatcher(target, maxUlpDiff, Floating::FloatingPointKind::Float);
29642 }
29643
29644 Floating::WithinAbsMatcher WithinAbs(double target, double margin) {
29645     return Floating::WithinAbsMatcher(target, margin);
29646 }
29647

```



```

29648 Floating::WithinRelMatcher WithinRel(double target, double eps) {
29649     return Floating::WithinRelMatcher(target, eps);
29650 }
29651
29652 Floating::WithinRelMatcher WithinRel(double target) {
29653     return Floating::WithinRelMatcher(target, std::numeric_limits<double>::epsilon() * 100);
29654 }
29655
29656 Floating::WithinRelMatcher WithinRel(float target, float eps) {
29657     return Floating::WithinRelMatcher(target, eps);
29658 }
29659
29660 Floating::WithinRelMatcher WithinRel(float target) {
29661     return Floating::WithinRelMatcher(target, std::numeric_limits<float>::epsilon() * 100);
29662 }
29663
29664 } // namespace Matchers
29665 } // namespace Catch
29666 // end catch_matchers_floating.cpp
29667 // start catch_matchers_generic.cpp
29668
29669 std::string Catch::Matchers::Generic::Detail::finalizeDescription(const std::string& desc) {
29670     if (desc.empty()) {
29671         return "matches undescribed predicate";
29672     } else {
29673         return "matches predicate: \"" + desc + '"';
29674     }
29675 }
29676 // end catch_matchers_generic.cpp
29677 // start catch_matchers_string.cpp
29678
29679 #include <regex>
29680
29681 namespace Catch {
29682 namespace Matchers {
29683     namespace StdString {
29684
29685         CasedString::CasedString( std::string const& str, CaseSensitive::Choice caseSensitivity )
29686         :   m_caseSensitivity( caseSensitivity ),
29687             m_str( adjustString( str ) )
29688         {}
29689
29690         std::string CasedString::adjustString( std::string const& str ) const {
29691             return m_caseSensitivity == CaseSensitive::No
29692                 ? toLower( str )
29693                 : str;
29694         }
29695
29696         std::string CasedString::caseSensitivitySuffix() const {
29697             return m_caseSensitivity == CaseSensitive::No
29698                 ? " (case insensitive)"
29699                 : std::string();
29700         }
29701
29702         StringMatcherBase::StringMatcherBase( std::string const& operation, CasedString const&
comparator )
29703         :   m_comparator( comparator ),
29704             m_operation( operation ) {}
29705
29706         std::string StringMatcherBase::describe() const {
29707             std::string description;
29708             description.reserve( 5 + m_operation.size() + m_comparator.m_str.size() +
m_comparator.caseSensitivitySuffix().size() );
29709             description += m_operation;
29710             description += ": ";
29711             description += m_comparator.m_str;
29712             description += " ";
29713             description += m_comparator.caseSensitivitySuffix();
29714             return description;
29715         }
29716
29717         EqualsMatcher::EqualsMatcher( CasedString const& comparator ) : StringMatcherBase( "equals",
comparator ) {}
29718
29719         bool EqualsMatcher::match( std::string const& source ) const {
29720             return m_comparator.adjustString( source ) == m_comparator.m_str;
29721         }
29722
29723         ContainsMatcher::ContainsMatcher( CasedString const& comparator ) : StringMatcherBase(
"contains", comparator ) {}
29724
29725         bool ContainsMatcher::match( std::string const& source ) const {
29726             return contains( m_comparator.adjustString( source ), m_comparator.m_str );
29727         }
29728
29729         StartsWithMatcher::StartsWithMatcher( CasedString const& comparator ) : StringMatcherBase(
"starts with", comparator ) {}

```

```

29731
29732     bool StartsWithMatcher::match( std::string const& source ) const {
29733         return startsWith( m_comparator.adjustString( source ), m_comparator.m_str );
29734     }
29735
29736     EndsWithMatcher::EndsWithMatcher( CasedString const& comparator ) : StringMatcherBase( "ends
with", comparator ) {}
29737
29738     bool EndsWithMatcher::match( std::string const& source ) const {
29739         return endsWith( m_comparator.adjustString( source ), m_comparator.m_str );
29740     }
29741
29742     RegexMatcher::RegexMatcher( std::string regex, CaseSensitive::Choice caseSensitivity ):
m_regex( std::move( regex ), m_caseSensitivity( caseSensitivity ) ) {}
29743
29744     bool RegexMatcher::match( std::string const& matchee ) const {
29745         auto flags = std::regex::ECMAScript; // ECMAScript is the default syntax option anyway
29746         if ( m_caseSensitivity == CaseSensitive::Choice::No ) {
29747             flags |= std::regex::icase;
29748         }
29749         auto reg = std::regex( m_regex, flags );
29750         return std::regex_match( matchee, reg );
29751     }
29752
29753     std::string RegexMatcher::describe() const {
29754         return "matches " + ::Catch::Detail::stringify( m_regex ) + ( ( m_caseSensitivity ==
CaseSensitive::Choice::Yes ) ? " case sensitively" : " case insensitively" );
29755     }
29756
29757     } // namespace StdString
29758
29759     StdString::EqualsMatcher Equals( std::string const& str, CaseSensitive::Choice caseSensitivity ) {
29760         return StdString::EqualsMatcher( StdString::CasedString( str, caseSensitivity ) );
29761     }
29762     StdString::ContainsMatcher Contains( std::string const& str, CaseSensitive::Choice caseSensitivity
) {
29763         return StdString::ContainsMatcher( StdString::CasedString( str, caseSensitivity ) );
29764     }
29765     StdString::EndsWithMatcher EndsWith( std::string const& str, CaseSensitive::Choice caseSensitivity
) {
29766         return StdString::EndsWithMatcher( StdString::CasedString( str, caseSensitivity ) );
29767     }
29768     StdString::StartsWithMatcher StartsWith( std::string const& str, CaseSensitive::Choice
caseSensitivity ) {
29769         return StdString::StartsWithMatcher( StdString::CasedString( str, caseSensitivity ) );
29770     }
29771
29772     StdString::RegexMatcher Matches( std::string const& regex, CaseSensitive::Choice caseSensitivity ) {
29773         return StdString::RegexMatcher( regex, caseSensitivity );
29774     }
29775
29776 } // namespace Matchers
29777 } // namespace Catch
29778 // end catch_matchers_string.cpp
29779 // start catch_message.cpp
29780
29781 // start catch_uncaught_exceptions.h
29782
29783 namespace Catch {
29784     bool uncaught_exceptions();
29785 } // end namespace Catch
29786
29787 // end catch_uncaught_exceptions.h
29788 #include <cassert>
29789 #include <stack>
29790
29791 namespace Catch {
29792
29793     MessageInfo::MessageInfo( StringRef const& _macroName,
SourceLineInfo const& _lineInfo,
ResultWas::OfType _type )
29794 :     macroName( _macroName ),
29795     lineInfo( _lineInfo ),
29796     type( _type ),
29797     sequence( ++globalCount )
29800 {}
29801
29802     bool MessageInfo::operator==( MessageInfo const& other ) const {
29803         return sequence == other.sequence;
29804     }
29805
29806     bool MessageInfo::operator<( MessageInfo const& other ) const {
29807         return sequence < other.sequence;
29808     }
29809
29810     // This may need protecting if threading support is added
29811     unsigned int MessageInfo::globalCount = 0;

```

```

29812
29813
29814
29815     Catch::MessageBuilder::MessageBuilder( StringRef const& macroName,
29816                                           SourceLineInfo const& lineInfo,
29817                                           ResultWas::OfType type )
29818     : m_info(macroName, lineInfo, type) {}
29819
29820
29821
29822     ScopedMessage::ScopedMessage( MessageBuilder const& builder )
29823     : m_info( builder.m_info ), m_moved()
29824     {
29825         m_info.message = builder.m_stream.str();
29826         getResultCapture().pushScopedMessage( m_info );
29827     }
29828
29829     ScopedMessage::ScopedMessage( ScopedMessage&& old )
29830     : m_info( old.m_info ), m_moved()
29831     {
29832         old.m_moved = true;
29833     }
29834
29835     ScopedMessage::~ScopedMessage() {
29836         if ( !uncaught_exceptions() && !m_moved ){
29837             getResultCapture().popScopedMessage(m_info);
29838         }
29839     }
29840
29841     Capturer::Capturer( StringRef macroName, SourceLineInfo const& lineInfo, ResultWas::OfType
resultType, StringRef names ) {
29842         auto trimmed = [&] (size_t start, size_t end) {
29843             while (names[start] == ',' || isspace(static_cast<unsigned char>(names[start]))) {
29844                 ++start;
29845             }
29846             while (names[end] == ',' || isspace(static_cast<unsigned char>(names[end]))) {
29847                 --end;
29848             }
29849             return names.substr(start, end - start + 1);
29850         };
29851         auto skipq = [&] (size_t start, char quote) {
29852             for (auto i = start + 1; i < names.size(); ++i) {
29853                 if (names[i] == quote)
29854                     return i;
29855                 if (names[i] == '\\')
29856                     ++i;
29857             }
29858             CATCH_INTERNAL_ERROR("CAPTURE parsing encountered unmatched quote");
29859         };
29860
29861         size_t start = 0;
29862         std::stack<char> openings;
29863         for (size_t pos = 0; pos < names.size(); ++pos) {
29864             char c = names[pos];
29865             switch (c) {
29866                 case '{':
29867                 case '[':
29868                 case '(':
29869                     // It is basically impossible to disambiguate between
29870                     // comparison and start of template args in this context
29871                     case '<':
29872                         openings.push(c);
29873                         break;
29874                 case ']':
29875                 case ')':
29876                 case '>':
29877                     openings.pop();
29878                     break;
29879                 case '"':
29880                 case '\\':
29881                     pos = skipq(pos, c);
29882                     break;
29883                 case ',':
29884                     if (start != pos && openings.empty()) {
29885                         m_messages.emplace_back(macroName, lineInfo, resultType);
29886                         m_messages.back().message = static_cast<std::string>(trimmed(start, pos));
29887                         m_messages.back().message += " := ";
29888                         start = pos;
29889                     }
29890             }
29891         }
29892         assert(openings.empty() && "Mismatched openings");
29893         m_messages.emplace_back(macroName, lineInfo, resultType);
29894         m_messages.back().message = static_cast<std::string>(trimmed(start, names.size() - 1));
29895         m_messages.back().message += " := ";
29896     }
29897     Capturer::~Capturer() {
29898         if ( !uncaught_exceptions() ){
29899

```

```

29900         assert( m_captured == m_messages.size() );
29901         for( size_t i = 0; i < m_captured; ++i )
29902             m_resultCapture.popScopedMessage( m_messages[i] );
29903     }
29904 }
29905
29906 void Capturer::captureValue( size_t index, std::string const& value ) {
29907     assert( index < m_messages.size() );
29908     m_messages[index].message += value;
29909     m_resultCapture.pushScopedMessage( m_messages[index] );
29910     m_captured++;
29911 }
29912
29913 } // end namespace Catch
29914 // end catch_message.cpp
29915 // start catch_output_redirect.cpp
29916
29917 // start catch_output_redirect.h
29918 #ifndef TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
29919 #define TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
29920
29921 #include <cstdio>
29922 #include <iosfwd>
29923 #include <string>
29924
29925 namespace Catch {
29926
29927     class RedirectedStream {
29928     public:
29929         std::ostream& m_originalStream;
29930         std::ostream& m_redirectionStream;
29931         std::streambuf* m_prevBuf;
29932
29933         RedirectedStream( std::ostream& originalStream, std::ostream& redirectionStream );
29934         ~RedirectedStream();
29935     };
29936
29937     class RedirectedStdOut {
29938     public:
29939         ReusableStringStream m_rss;
29940         RedirectedStream m_cout;
29941
29942         RedirectedStdOut();
29943         auto str() const -> std::string;
29944     };
29945
29946     // StdErr has two constituent streams in C++, std::cerr and std::clog
29947     // This means that we need to redirect 2 streams into 1 to keep proper
29948     // order of writes
29949     class RedirectedStdErr {
29950     public:
29951         ReusableStringStream m_rss;
29952         RedirectedStream m_cerr;
29953         RedirectedStream m_clog;
29954
29955         RedirectedStdErr();
29956         auto str() const -> std::string;
29957     };
29958
29959     class RedirectedStreams {
29960     public:
29961         RedirectedStreams( RedirectedStreams const& ) = delete;
29962         RedirectedStreams& operator=( RedirectedStreams const& ) = delete;
29963         RedirectedStreams( RedirectedStreams&& ) = delete;
29964         RedirectedStreams& operator=( RedirectedStreams&& ) = delete;
29965
29966         RedirectedStreams( std::string& redirectedCout, std::string& redirectedCerr );
29967         ~RedirectedStreams();
29968     private:
29969         std::string& m_redirectedCout;
29970         std::string& m_redirectedCerr;
29971         RedirectedStdOut m_redirectedStdOut;
29972         RedirectedStdErr m_redirectedStdErr;
29973     };
29974
29975 #if defined( CATCH_CONFIG_NEW_CAPTURE )
29976
29977     // Windows's implementation of std::tmpfile is terrible (it tries
29978     // to create a file inside system folder, thus requiring elevated
29979     // privileges for the binary), so we have to use tmpnam(_s) and
29980     // create the file ourselves there.
29981     class TempFile {
29982     public:
29983         TempFile( TempFile const& ) = delete;
29984         TempFile& operator=( TempFile const& ) = delete;
29985         TempFile( TempFile&& ) = delete;
29986         TempFile& operator=( TempFile&& ) = delete;
29987
29988         TempFile();

```

```

29987     ~TempFile();
29988
29989     std::FILE* getFile();
29990     std::string getContents();
29991
29992 private:
29993     std::FILE* m_file = nullptr;
29994     #if defined(_MSC_VER)
29995         char m_buffer[L_tmpnam] = { 0 };
29996     #endif
29997 };
29998
29999 class OutputRedirect {
30000 public:
30001     OutputRedirect(OutputRedirect const&) = delete;
30002     OutputRedirect& operator=(OutputRedirect const&) = delete;
30003     OutputRedirect(OutputRedirect&&) = delete;
30004     OutputRedirect& operator=(OutputRedirect&&) = delete;
30005
30006     OutputRedirect(std::string& stdout_dest, std::string& stderr_dest);
30007     ~OutputRedirect();
30008
30009 private:
30010     int m_originalStdout = -1;
30011     int m_originalStderr = -1;
30012     TempFile m_stdoutFile;
30013     TempFile m_stderrFile;
30014     std::string& m_stdoutDest;
30015     std::string& m_stderrDest;
30016 };
30017
30018 #endif
30019
30020 } // end namespace Catch
30021
30022 #endif // TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
30023 // end catch_output_redirect.h
30024 #include <cstdio>
30025 #include <cstring>
30026 #include <fstream>
30027 #include <sstream>
30028 #include <stdexcept>
30029
30030 #if defined(CATCH_CONFIG_NEW_CAPTURE)
30031     #if defined(_MSC_VER)
30032         #include <io.h> // _dup and _dup2
30033         #define dup _dup
30034         #define dup2 _dup2
30035         #define fileno _fileno
30036     #else
30037         #include <unistd.h> // dup and dup2
30038     #endif
30039 #endif
30040
30041 namespace Catch {
30042
30043     RedirectedStream::RedirectedStream( std::ostream& originalStream, std::ostream& redirectionStream
30044 )
30045     :   m_originalStream( originalStream ),
30046         m_redirectionStream( redirectionStream ),
30047         m_prevBuf( m_originalStream.rdbuf() )
30048     {
30049         m_originalStream.rdbuf( m_redirectionStream.rdbuf() );
30050     }
30051
30052     RedirectedStream::~RedirectedStream() {
30053         m_originalStream.rdbuf( m_prevBuf );
30054     }
30055
30056     RedirectedStdOut::RedirectedStdOut() : m_cout( Catch::cout(), m_rss.get() ) {}
30057     auto RedirectedStdOut::str() const -> std::string { return m_rss.str(); }
30058
30059     RedirectedStdErr::RedirectedStdErr()
30060     :   m_cerr( Catch::cerr(), m_rss.get() ),
30061         m_clog( Catch::clog(), m_rss.get() )
30062     {}
30063     auto RedirectedStdErr::str() const -> std::string { return m_rss.str(); }
30064
30065     RedirectedStreams::RedirectedStreams(std::string& redirectedCout, std::string& redirectedCerr)
30066     :   m_redirectedCout(redirectedCout),
30067         m_redirectedCerr(redirectedCerr)
30068     {}
30069
30070     RedirectedStreams::~RedirectedStreams() {
30071         m_redirectedCout += m_redirectedStdOut.str();
30072         m_redirectedCerr += m_redirectedStdErr.str();
30073     }

```

```

30073
30074 #if defined(CATCH_CONFIG_NEW_CAPTURE)
30075
30076 #if defined(_MSC_VER)
30077     TempFile::TempFile() {
30078         if (tmpnam_s(m_buffer)) {
30079             CATCH_RUNTIME_ERROR("Could not get a temp filename");
30080         }
30081         if (fopen_s(&m_file, m_buffer, "w+")) {
30082             char buffer[100];
30083             if (strerror_s(buffer, errno)) {
30084                 CATCH_RUNTIME_ERROR("Could not translate errno to a string");
30085             }
30086             CATCH_RUNTIME_ERROR("Could not open the temp file: '" < m_buffer < "' because: " <
buffer);
30087         }
30088     }
30089 #else
30090     TempFile::TempFile() {
30091         m_file = std::tmpfile();
30092         if (!m_file) {
30093             CATCH_RUNTIME_ERROR("Could not create a temp file.");
30094         }
30095     }
30096 #endif
30097
30098     TempFile::~TempFile() {
30099         // TBD: What to do about errors here?
30100         std::fclose(m_file);
30101         // We manually create the file on Windows only, on Linux
30102         // it will be autodeleted
30103         #if defined(_MSC_VER)
30104             std::remove(m_buffer);
30105         #endif
30106     }
30107
30108     FILE* TempFile::getFile() {
30109         return m_file;
30110     }
30111
30112     std::string TempFile::getContents() {
30113         std::stringstream sstr;
30114         char buffer[100] = {};
30115         std::rewind(m_file);
30116         while (std::fgets(buffer, sizeof(buffer), m_file)) {
30117             sstr << buffer;
30118         }
30119         return sstr.str();
30120     }
30121
30122     OutputRedirect::OutputRedirect(std::string& stdout_dest, std::string& stderr_dest) :
30123         m_originalStdout(dup(1)),
30124         m_originalStderr(dup(2)),
30125         m_stdoutDest(stdout_dest),
30126         m_stderrDest(stderr_dest) {
30127         dup2(fileno(m_stdoutFile.getFile()), 1);
30128         dup2(fileno(m_stderrFile.getFile()), 2);
30129     }
30130
30131     OutputRedirect::~OutputRedirect() {
30132         Catch::cout() << std::flush;
30133         fflush(stdout);
30134         // Since we support overriding these streams, we flush cerr
30135         // even though std::cerr is unbuffered
30136         Catch::cerr() << std::flush;
30137         Catch::clog() << std::flush;
30138         fflush(stderr);
30139
30140         dup2(m_originalStdout, 1);
30141         dup2(m_originalStderr, 2);
30142
30143         m_stdoutDest += m_stdoutFile.getContents();
30144         m_stderrDest += m_stderrFile.getContents();
30145     }
30146 #endif // CATCH_CONFIG_NEW_CAPTURE
30147
30148 } // namespace Catch
30149
30150 #if defined(CATCH_CONFIG_NEW_CAPTURE)
30151 #if defined(_MSC_VER)
30152 #undef dup
30153 #undef dup2
30154 #undef fileno
30155 #endif
30156 #endif
30157
30158 #endif

```

```

30159 // end catch_output_redirect.cpp
30160 // start catch_polyfills.cpp
30161
30162 #include <cmath>
30163
30164 namespace Catch {
30165
30166 #if !defined(CATCH_CONFIG_POLYFILL_ISNAN)
30167     bool isnan(float f) {
30168         return std::isnan(f);
30169     }
30170     bool isnan(double d) {
30171         return std::isnan(d);
30172     }
30173 #else
30174     // For now we only use this for embarcadero
30175     bool isnan(float f) {
30176         return std::_isnan(f);
30177     }
30178     bool isnan(double d) {
30179         return std::_isnan(d);
30180     }
30181 #endif
30182
30183 } // end namespace Catch
30184 // end catch_polyfills.cpp
30185 // start catch_random_number_generator.cpp
30186
30187 namespace Catch {
30188
30189     namespace {
30190
30191         #if defined(_MSC_VER)
30192             #pragma warning(push)
30193             #pragma warning(disable:4146) // we negate uint32 during the rotate
30194         #endif
30195         // Safe rotr implementation thanks to John Regehr
30196         uint32_t rotate_right(uint32_t val, uint32_t count) {
30197             const uint32_t mask = 31;
30198             count &= mask;
30199             return (val >> count) | (val << (-count & mask));
30200         }
30201
30202         #if defined(_MSC_VER)
30203             #pragma warning(pop)
30204         #endif
30205     }
30206
30207     SimplePcg32::SimplePcg32(result_type seed_) {
30208         seed(seed_);
30209     }
30210
30211     void SimplePcg32::seed(result_type seed_) {
30212         m_state = 0;
30213         (*this)();
30214         m_state += seed_;
30215         (*this)();
30216     }
30217
30218     void SimplePcg32::discard(uint64_t skip) {
30219         // We could implement this to run in O(log n) steps, but this
30220         // should suffice for our use case.
30221         for (uint64_t s = 0; s < skip; ++s) {
30222             static_cast<void>((*this)());
30223         }
30224     }
30225
30226     SimplePcg32::result_type SimplePcg32::operator()() {
30227         // prepare the output value
30228         const uint32_t xorshifted = static_cast<uint32_t>(((m_state >> 18u) ^ m_state) >> 27u);
30229         const auto output = rotate_right(xorshifted, m_state >> 59u);
30230
30231         // advance state
30232         m_state = m_state * 6364136223846793005ULL + s_inc;
30233
30234         return output;
30235     }
30236
30237     bool operator==(SimplePcg32 const& lhs, SimplePcg32 const& rhs) {
30238         return lhs.m_state == rhs.m_state;
30239     }
30240
30241     bool operator!=(SimplePcg32 const& lhs, SimplePcg32 const& rhs) {
30242         return lhs.m_state != rhs.m_state;
30243     }
30244
30245 }

```

```

30246 // end catch_random_number_generator.cpp
30247 // start catch_registry_hub.cpp
30248
30249 // start catch_test_case_registry_impl.h
30250
30251 #include <vector>
30252 #include <set>
30253 #include <algorithm>
30254 #include <ios>
30255
30256 namespace Catch {
30257
30258     class TestCase;
30259     struct IConfig;
30260
30261     std::vector<TestCase> sortTests( IConfig const& config, std::vector<TestCase> const&
30262     unsortedTestCases );
30263
30264     bool isThrowSafe( TestCase const& testCase, IConfig const& config );
30265     bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config );
30266
30267     void enforceNoDuplicateTestCases( std::vector<TestCase> const& functions );
30268
30269     std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
30270     testSpec, IConfig const& config );
30271     std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config );
30272
30273     class TestRegistry : public ITestRegistry {
30274     public:
30275         virtual ~TestRegistry() = default;
30276
30277         virtual void registerTest( TestCase const& testCase );
30278
30279         std::vector<TestCase> const& getAllTests() const override;
30280         std::vector<TestCase> const& getAllTestsSorted( IConfig const& config ) const override;
30281
30282     private:
30283         std::vector<TestCase> m_functions;
30284         mutable RunTests::InWhatOrder m_currentSortOrder = RunTests::InDeclarationOrder;
30285         mutable std::vector<TestCase> m_sortedFunctions;
30286         std::size_t m_unnamedCount = 0;
30287         std::ios_base::Init m_ostreamInit; // Forces cout/ cerr to be initialised
30288     };
30289
30290     class TestInvokerAsFunction : public ITestInvoker {
30291     public:
30292         void(*m_testAsFunction)();
30293         TestInvokerAsFunction( void(*testAsFunction)() ) noexcept;
30294
30295         void invoke() const override;
30296     };
30297
30298     std::string extractClassName( StringRef const& classOrQualifiedMethodName );
30299
30300 } // end namespace Catch
30301
30302 // end catch_test_case_registry_impl.h
30303 // start catch_reporter_registry.h
30304
30305 #include <map>
30306
30307 namespace Catch {
30308
30309     class ReporterRegistry : public IReporterRegistry {
30310     public:
30311         ~ReporterRegistry() override;
30312
30313         IStreamingReporterPtr create( std::string const& name, IConfigPtr const& config ) const
30314         override;
30315
30316         void registerReporter( std::string const& name, IReporterFactoryPtr const& factory );
30317         void registerListener( IReporterFactoryPtr const& factory );
30318
30319         FactoryMap const& getFactories() const override;
30320         Listeners const& getListeners() const override;
30321
30322     private:
30323         FactoryMap m_factories;
30324         Listeners m_listeners;
30325     };
30326
30327 }
30328
30329 // end catch_reporter_registry.h

```



```

30332 // start catch_tag_alias_registry.h
30333
30334 // start catch_tag_alias.h
30335
30336 #include <string>
30337
30338 namespace Catch {
30339
30340     struct TagAlias {
30341         TagAlias(std::string const& _tag, SourceLineInfo _lineInfo);
30342
30343         std::string tag;
30344         SourceLineInfo lineInfo;
30345     };
30346
30347 } // end namespace Catch
30348
30349 // end catch_tag_alias.h
30350 #include <map>
30351
30352 namespace Catch {
30353
30354     class TagAliasRegistry : public ITagAliasRegistry {
30355     public:
30356         ~TagAliasRegistry() override;
30357         TagAlias const* find( std::string const& alias ) const override;
30358         std::string expandAliases( std::string const& unexpandedTestSpec ) const override;
30359         void add( std::string const& alias, std::string const& tag, SourceLineInfo const& lineInfo );
30360
30361     private:
30362         std::map<std::string, TagAlias> m_registry;
30363     };
30364
30365 } // end namespace Catch
30366
30367 // end catch_tag_alias_registry.h
30368 // start catch_startup_exception_registry.h
30369
30370 #include <vector>
30371 #include <exception>
30372
30373 namespace Catch {
30374
30375     class StartupExceptionRegistry {
30376     #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
30377     public:
30378         void add(std::exception_ptr const& exception) noexcept;
30379         std::vector<std::exception_ptr> const& getExceptions() const noexcept;
30380     private:
30381         std::vector<std::exception_ptr> m_exceptions;
30382     #endif
30383     };
30384
30385 } // end namespace Catch
30386
30387 // end catch_startup_exception_registry.h
30388 // start catch_singletons.hpp
30389
30390 namespace Catch {
30391
30392     struct ISingleton {
30393         virtual ~ISingleton();
30394     };
30395
30396     void addSingleton( ISingleton* singleton );
30397     void cleanupSingletons();
30398
30399     template<typename SingletonImplT, typename InterfaceT = SingletonImplT, typename MutableInterfaceT
= InterfaceT>
30400     class Singleton : SingletonImplT, public ISingleton {
30401
30402     public:
30403         static auto getInternal() -> Singleton* {
30404             static Singleton* s_instance = nullptr;
30405             if( !s_instance ) {
30406                 s_instance = new Singleton;
30407                 addSingleton( s_instance );
30408             }
30409             return s_instance;
30410         }
30411
30412     public:
30413         static auto get() -> InterfaceT const& {
30414             return *getInternal();
30415         }
30416         static auto getMutable() -> MutableInterfaceT& {
30417             return *getInternal();
30418         }
30419     };
30420

```

```

30418     };
30419
30420 } // namespace Catch
30421
30422 // end catch_singletons.hpp
30423 namespace Catch {
30424
30425     namespace {
30426
30427         class RegistryHub : public IRegistryHub, public IMutableRegistryHub,
30428                             private NonCopyable {
30429
30430         public: // IRegistryHub
30431             RegistryHub() = default;
30432             IReporterRegistry const& getReporterRegistry() const override {
30433                 return m_reporterRegistry;
30434             }
30435             ITestCaseRegistry const& getTestCaseRegistry() const override {
30436                 return m_testCaseRegistry;
30437             }
30438             IExceptionTranslatorRegistry const& getExceptionTranslatorRegistry() const override {
30439                 return m_exceptionTranslatorRegistry;
30440             }
30441             ITagAliasRegistry const& getTagAliasRegistry() const override {
30442                 return m_tagAliasRegistry;
30443             }
30444             StartupExceptionRegistry const& getStartupExceptionRegistry() const override {
30445                 return m_exceptionRegistry;
30446             }
30447
30448         public: // IMutableRegistryHub
30449             void registerReporter( std::string const& name, IReporterFactoryPtr const& factory )
30450             override {
30451                 m_reporterRegistry.registerReporter( name, factory );
30452             }
30453             void registerListener( IReporterFactoryPtr const& factory ) override {
30454                 m_reporterRegistry.registerListener( factory );
30455             }
30456             void registerTest( TestCase const& testInfo ) override {
30457                 m_testCaseRegistry.registerTest( testInfo );
30458             }
30459             void registerTranslator( const IExceptionTranslator* translator ) override {
30460                 m_exceptionTranslatorRegistry.registerTranslator( translator );
30461             }
30462             void registerTagAlias( std::string const& alias, std::string const& tag, SourceLineInfo
30463             const& lineInfo ) override {
30464                 m_tagAliasRegistry.add( alias, tag, lineInfo );
30465             }
30466             void registerStartupException() noexcept override {
30467                 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
30468                     m_exceptionRegistry.add(std::current_exception());
30469                 #else
30470                     CATCH_INTERNAL_ERROR("Attempted to register active exception under
30471                     CATCH_CONFIG_DISABLE_EXCEPTIONS!");
30472                 #endif
30473             }
30474             IMutableEnumValuesRegistry& getMutableEnumValuesRegistry() override {
30475                 return m_enumValuesRegistry;
30476             }
30477
30478         private:
30479             TestRegistry m_testCaseRegistry;
30480             ReporterRegistry m_reporterRegistry;
30481             ExceptionTranslatorRegistry m_exceptionTranslatorRegistry;
30482             TagAliasRegistry m_tagAliasRegistry;
30483             StartupExceptionRegistry m_exceptionRegistry;
30484             Detail::EnumValuesRegistry m_enumValuesRegistry;
30485
30486         };
30487
30488         using RegistryHubSingleton = Singleton<RegistryHub, IRegistryHub, IMutableRegistryHub>;
30489
30490         IRegistryHub const& getRegistryHub() {
30491             return RegistryHubSingleton::get();
30492         }
30493         IMutableRegistryHub& getMutableRegistryHub() {
30494             return RegistryHubSingleton::getMutable();
30495         }
30496         void cleanup() {
30497             cleanupSingletons();
30498             cleanupContext();
30499         }
30500         std::string translateActiveException() {
30501             return getRegistryHub().getExceptionTranslatorRegistry().translateActiveException();
30502         }
30503     } // end namespace Catch
30504 } // end namespace Catch

```

```

30502 // end catch_registry_hub.cpp
30503 // start catch_reporter_registry.cpp
30504
30505 namespace Catch {
30506     ReporterRegistry::~ReporterRegistry() = default;
30507
30508     IStreamingReporterPtr ReporterRegistry::create( std::string const& name, IConfigPtr const& config
30509 ) const {
30510         auto it = m_factories.find( name );
30511         if( it == m_factories.end() )
30512             return nullptr;
30513         return it->second->create( ReporterConfig( config ) );
30514     }
30515
30516     void ReporterRegistry::registerReporter( std::string const& name, IReporterFactoryPtr const&
30517 factory ) {
30518         m_factories.emplace(name, factory);
30519     }
30520     void ReporterRegistry::registerListener( IReporterFactoryPtr const& factory ) {
30521         m_listeners.push_back( factory );
30522     }
30523     IReporterRegistry::FactoryMap const& ReporterRegistry::getFactories() const {
30524         return m_factories;
30525     }
30526     IReporterRegistry::Listeners const& ReporterRegistry::getListeners() const {
30527         return m_listeners;
30528     }
30529 }
30530
30531 // end catch_reporter_registry.cpp
30532 // start catch_result_type.cpp
30533
30534 namespace Catch {
30535     bool isOk( ResultWas::OfType resultType ) {
30536         return ( resultType & ResultWas::FailureBit ) == 0;
30537     }
30538     bool isJustInfo( int flags ) {
30539         return flags == ResultWas::Info;
30540     }
30541
30542     ResultDisposition::Flags operator | ( ResultDisposition::Flags lhs, ResultDisposition::Flags rhs )
30543     {
30544         return static_cast<ResultDisposition::Flags>( static_cast<int>( lhs ) | static_cast<int>( rhs
30545 ) );
30546     }
30547     bool shouldContinueOnFailure( int flags ) { return ( flags &
30548 ResultDisposition::ContinueOnFailure ) != 0; }
30549     bool shouldSuppressFailure( int flags ) { return ( flags & ResultDisposition::SuppressFail )
30550 != 0; }
30551 } // end namespace Catch
30552 // end catch_result_type.cpp
30553 // start catch_run_context.cpp
30554
30555 #include <cassert>
30556 #include <algorithm>
30557 #include <sstream>
30558
30559 namespace Catch {
30560     namespace Generators {
30561         struct GeneratorTracker : TestCaseTracking::TrackerBase, IGeneratorTracker {
30562             GeneratorBasePtr m_generator;
30563
30564             GeneratorTracker( TestCaseTracking::NameAndLocation const& nameAndLocation,
30565 TrackerContext& ctx, ITracker* parent )
30566             : TrackerBase( nameAndLocation, ctx, parent )
30567             {}
30568             ~GeneratorTracker();
30569
30570             static GeneratorTracker& acquire( TrackerContext& ctx, TestCaseTracking::NameAndLocation
30571 const& nameAndLocation ) {
30572                 std::shared_ptr<GeneratorTracker> tracker;
30573
30574                 ITracker& currentTracker = ctx.currentTracker();
30575                 // Under specific circumstances, the generator we want
30576                 // to acquire is also the current tracker. If this is
30577                 // the case, we have to avoid looking through current
30578                 // tracker's children, and instead return the current
30579                 // tracker.
30580                 // A case where this check is important is e.g.
30581                 // for (int i = 0; i < 5; ++i) {
30582                     int n = GENERATE(1, 2);

```

```

30581         //      }
30582         //
30583         // without it, the code above creates 5 nested generators.
30584         if (currentTracker.nameAndLocation() == nameAndLocation) {
30585             auto thisTracker = currentTracker.parent().findChild(nameAndLocation);
30586             assert(thisTracker);
30587             assert(thisTracker->isGeneratorTracker());
30588             tracker = std::static_pointer_cast<GeneratorTracker>(thisTracker);
30589         } else if ( TestCaseTracking::ITrackerPtr childTracker = currentTracker.findChild(
nameAndLocation ) ) {
30590             assert( childTracker );
30591             assert( childTracker->isGeneratorTracker() );
30592             tracker = std::static_pointer_cast<GeneratorTracker>( childTracker );
30593         } else {
30594             tracker = std::make_shared<GeneratorTracker>( nameAndLocation, ctx,
&currentTracker );
30595             currentTracker.addChild( tracker );
30596         }
30597
30598         if( !tracker->isComplete() ) {
30599             tracker->open();
30600         }
30601
30602         return *tracker;
30603     }
30604
30605     // TrackerBase interface
30606     bool isGeneratorTracker() const override { return true; }
30607     auto hasGenerator() const -> bool override {
30608         return !m_generator;
30609     }
30610     void close() override {
30611         TrackerBase::close();
30612         // If a generator has a child (it is followed by a section)
30613         // and none of its children have started, then we must wait
30614         // until later to start consuming its values.
30615         // This catches cases where `GENERATE` is placed between two
30616         // `SECTION`s.
30617         // **The check for m_children.empty cannot be removed**.
30618         // doing so would break `GENERATE` `_not_` followed by `SECTION`s.
30619         const bool should_wait_for_child = [&]() {
30620             // No children -> nobody to wait for
30621             if ( m_children.empty() ) {
30622                 return false;
30623             }
30624             // If at least one child started executing, don't wait
30625             if ( std::find_if(
30626                 m_children.begin(),
30627                 m_children.end(),
30628                 []( TestCaseTracking::ITrackerPtr tracker ) {
30629                     return tracker->hasStarted();
30630                 } ) != m_children.end() ) {
30631                 return false;
30632             }
30633
30634             // No children have started. We need to check if they _can_
30635             // start, and thus we should wait for them, or they cannot
30636             // start (due to filters), and we shouldn't wait for them
30637             auto* parent = m_parent;
30638             // This is safe: there is always at least one section
30639             // tracker in a test case tracking tree
30640             while ( !parent->isSectionTracker() ) {
30641                 parent = &( parent->parent() );
30642             }
30643             assert( parent &&
30644                 "Missing root (test case) level section" );
30645
30646             auto const& parentSection =
30647                 static_cast<SectionTracker&>( *parent );
30648             auto const& filters = parentSection.getFilters();
30649             // No filters -> no restrictions on running sections
30650             if ( filters.empty() ) {
30651                 return true;
30652             }
30653
30654             for ( auto const& child : m_children ) {
30655                 if ( child->isSectionTracker() &&
30656                     std::find( filters.begin(),
30657                         filters.end(),
30658                         static_cast<SectionTracker&>( *child )
30659                             .trimmedName() ) !=
30660                     filters.end() ) {
30661                     return true;
30662                 }
30663             }
30664             return false;
30665         }();

```

```

30666
30667 // This check is a bit tricky, because m_generator->next()
30668 // has a side-effect, where it consumes generator's current
30669 // value, but we do not want to invoke the side-effect if
30670 // this generator is still waiting for any child to start.
30671 if ( should_wait_for_child ||
30672      ( m_runState == CompletedSuccessfully &&
30673        m_generator->next() ) ) {
30674     m_children.clear();
30675     m_runState = Executing;
30676 }
30677 }
30678
30679 // IGeneratorTracker interface
30680 auto getGenerator() const -> GeneratorBasePtr const& override {
30681     return m_generator;
30682 }
30683 void setGenerator( GeneratorBasePtr&& generator ) override {
30684     m_generator = std::move( generator );
30685 }
30686 };
30687 GeneratorTracker::~GeneratorTracker() {}
30688 }
30689
30690 RunContext::RunContext( IConfigPtr const& _config, IStreamingReporterPtr&& reporter)
30691 :   m_runInfo( _config->name() ),
30692   m_context( getCurrentMutableContext() ),
30693   m_config( _config ),
30694   m_reporter( std::move( reporter ) ),
30695   m_lastAssertionInfo{ StringRef(), SourceLineInfo("",0), StringRef(), ResultDisposition::Normal
30696 },
30697   m_includeSuccessfulResults( m_config->includeSuccessfulResults() ||
30698                               m_reporter->getPreferences().shouldReportAllAssertions )
30699 {
30700     m_context.setRunner( this );
30701     m_context.setConfig( m_config );
30702     m_context.setResultCapture( this );
30703     m_reporter->testRunStarting( m_runInfo );
30704 }
30705
30706 RunContext::~RunContext() {
30707     m_reporter->testRunEnded( TestRunStats( m_runInfo, m_totals, aborting() ) );
30708 }
30709
30710 void RunContext::testGroupStarting( std::string const& testSpec, std::size_t groupIndex,
30711                                     std::size_t groupsCount ) {
30712     m_reporter->testGroupStarting( GroupInfo( testSpec, groupIndex, groupsCount ) );
30713 }
30714
30715 void RunContext::testGroupEnded( std::string const& testSpec, Totals const& totals, std::size_t
30716 groupIndex, std::size_t groupsCount ) {
30717     m_reporter->testGroupEnded( TestGroupStats( GroupInfo( testSpec, groupIndex, groupsCount ),
30718         totals, aborting() ) );
30719 }
30720
30721 Totals RunContext::runTest( TestCase const& testCase ) {
30722     Totals prevTotals = m_totals;
30723
30724     std::string redirectedCout;
30725     std::string redirectedCerr;
30726
30727     auto const& testInfo = testCase.getTestCaseInfo();
30728
30729     m_reporter->testCaseStarting( testInfo );
30730
30731     m_activeTestCase = &testCase;
30732
30733     ITracker& rootTracker = m_trackerContext.startRun();
30734     assert( rootTracker.isSectionTracker() );
30735     static_cast<SectionTracker&>( rootTracker ).addInitialFilters( m_config->getSectionsToRun() );
30736     do {
30737         m_trackerContext.startCycle();
30738         m_testCaseTracker = &SectionTracker::acquire( m_trackerContext,
30739             TestCaseTracking::NameAndLocation( testInfo.name, testInfo.lineInfo ) );
30740         runCurrentTest( redirectedCout, redirectedCerr );
30741     } while ( !m_testCaseTracker->isSuccessfullyCompleted() && !aborting() );
30742
30743     Totals deltaTotals = m_totals.delta( prevTotals );
30744     if ( testInfo.expectedToFail() && deltaTotals.testCases.passed > 0 ) {
30745         deltaTotals.assertions.failed++;
30746         deltaTotals.testCases.passed--;
30747         deltaTotals.testCases.failed++;
30748     }
30749     m_totals.testCases += deltaTotals.testCases;
30750     m_reporter->testCaseEnded( TestCaseStats( testInfo,
30751         deltaTotals,
30752         redirectedCout,

```

```

30747             redirectedCerr,
30748             aborting()));
30749
30750     m_activeTestCase = nullptr;
30751     m_testCaseTracker = nullptr;
30752
30753     return deltaTotals;
30754 }
30755
30756 IConfigPtr RunContext::config() const {
30757     return m_config;
30758 }
30759
30760 IStreamingReporter& RunContext::reporter() const {
30761     return *m_reporter;
30762 }
30763
30764 void RunContext::assertionEnded(AssertionResult const & result) {
30765     if (result.getResultType() == ResultWas::Ok) {
30766         m_totals.assertions.passed++;
30767         m_lastAssertionPassed = true;
30768     } else if (!result.isOk()) {
30769         m_lastAssertionPassed = false;
30770         if (m_activeTestCase->getTestCaseInfo().okToFail() )
30771             m_totals.assertions.failedButOk++;
30772         else
30773             m_totals.assertions.failed++;
30774     }
30775     else {
30776         m_lastAssertionPassed = true;
30777     }
30778
30779     // We have no use for the return value (whether messages should be cleared), because messages
were made scoped
30780     // and should be let to clear themselves out.
30781     static_cast<void>(m_reporter->assertionEnded(AssertionStats(result, m_messages, m_totals)));
30782
30783     if (result.getResultType() != ResultWas::Warning)
30784         m_messageScopes.clear();
30785
30786     // Reset working state
30787     resetAssertionInfo();
30788     m_lastResult = result;
30789 }
30790 void RunContext::resetAssertionInfo() {
30791     m_lastAssertionInfo.macroName = StringRef();
30792     m_lastAssertionInfo.capturedExpression = "{Unknown expression after the reported line}"_sr;
30793 }
30794
30795 bool RunContext::sectionStarted(SectionInfo const & sectionInfo, Counts & assertions) {
30796     ITracker& sectionTracker = SectionTracker::acquire(m_trackerContext,
TestCaseTracking::NameAndLocation(sectionInfo.name, sectionInfo.lineInfo));
30797     if (!sectionTracker.isOpen())
30798         return false;
30799     m_activeSections.push_back(&sectionTracker);
30800
30801     m_lastAssertionInfo.lineInfo = sectionInfo.lineInfo;
30802
30803     m_reporter->sectionStarting(sectionInfo);
30804
30805     assertions = m_totals.assertions;
30806
30807     return true;
30808 }
30809 auto RunContext::acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo
) -> IGeneratorTracker& {
30810     using namespace Generators;
30811     GeneratorTracker& tracker = GeneratorTracker::acquire(m_trackerContext,
30812         TestCaseTracking::NameAndLocation(
static_cast<std::string>(generatorName), lineInfo ) );
30813     m_lastAssertionInfo.lineInfo = lineInfo;
30814     return tracker;
30815 }
30816
30817 bool RunContext::testForMissingAssertions(Counts& assertions) {
30818     if (assertions.total() != 0)
30819         return false;
30820     if (!m_config->warnAboutMissingAssertions())
30821         return false;
30822     if (m_trackerContext.currentTracker().hasChildren())
30823         return false;
30824     m_totals.assertions.failed++;
30825     assertions.failed++;
30826     return true;
30827 }
30828
30829 void RunContext::sectionEnded(SectionEndInfo const & endInfo) {

```

```

30830         Counts assertions = m_totals.assertions - endInfo.prevAssertions;
30831         bool missingAssertions = testForMissingAssertions(assertions);
30832
30833         if (!m_activeSections.empty()) {
30834             m_activeSections.back()->close();
30835             m_activeSections.pop_back();
30836         }
30837
30838         m_reporter->sectionEnded(SectionStats(endInfo.sectionInfo, assertions,
endInfo.durationInSeconds, missingAssertions));
30839         m_messages.clear();
30840         m_messageScopes.clear();
30841     }
30842
30843     void RunContext::sectionEndedEarly(SectionEndInfo const & endInfo) {
30844         if (m_unfinishedSections.empty())
30845             m_activeSections.back()->fail();
30846         else
30847             m_activeSections.back()->close();
30848         m_activeSections.pop_back();
30849
30850         m_unfinishedSections.push_back(endInfo);
30851     }
30852
30853 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
30854     void RunContext::benchmarkPreparing(std::string const& name) {
30855         m_reporter->benchmarkPreparing(name);
30856     }
30857     void RunContext::benchmarkStarting( BenchmarkInfo const& info ) {
30858         m_reporter->benchmarkStarting( info );
30859     }
30860     void RunContext::benchmarkEnded( BenchmarkStats<> const& stats ) {
30861         m_reporter->benchmarkEnded( stats );
30862     }
30863     void RunContext::benchmarkFailed(std::string const & error) {
30864         m_reporter->benchmarkFailed(error);
30865     }
30866 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
30867
30868     void RunContext::pushScopedMessage(MessageInfo const & message) {
30869         m_messages.push_back(message);
30870     }
30871
30872     void RunContext::popScopedMessage(MessageInfo const & message) {
30873         m_messages.erase(std::remove(m_messages.begin(), m_messages.end(), message),
m_messages.end());
30874     }
30875
30876     void RunContext::emplaceUnscopedMessage( MessageBuilder const& builder ) {
30877         m_messageScopes.emplace_back( builder );
30878     }
30879
30880     std::string RunContext::getCurrentTestName() const {
30881         return m_activeTestCase
            ? m_activeTestCase->getTestCaseInfo().name
            : std::string();
30882     }
30883
30884     const AssertionResult * RunContext::getLastResult() const {
30885         return &(*m_lastResult);
30886     }
30887
30888     void RunContext::exceptionEarlyReported() {
30889         m_shouldReportUnexpected = false;
30890     }
30891
30892     void RunContext::handleFatalErrorCondition( StringRef message ) {
30893         // First notify reporter that bad things happened
30894         m_reporter->fatalErrorEncountered(message);
30895
30896         // Don't rebuild the result -- the stringification itself can cause more fatal errors
30897         // Instead, fake a result data.
30898         AssertionResultData tempResult( ResultWas::FatalErrorCondition, { false } );
30899         tempResult.message = static_cast<std::string>(message);
30900         AssertionResult result(m_lastAssertionInfo, tempResult);
30901
30902         assertionEnded(result);
30903
30904         handleUnfinishedSections();
30905
30906         // Recreate section for test case (as we will lose the one that was in scope)
30907         auto const& testCaseInfo = m_activeTestCase->getTestCaseInfo();
30908         SectionInfo testCaseSection(testCaseInfo.lineInfo, testCaseInfo.name);
30909
30910         Counts assertions;
30911         assertions.failed = 1;
30912         SectionStats testCaseSectionStats(testCaseSection, assertions, 0, false);
30913
30914

```

```

30915     m_reporter->sectionEnded(testCaseSectionStats);
30916
30917     auto const& testInfo = m_activeTestCase->getTestCaseInfo();
30918
30919     Totals deltaTotals;
30920     deltaTotals.testCases.failed = 1;
30921     deltaTotals.assertions.failed = 1;
30922     m_reporter->testCaseEnded(TestCaseStats(testInfo,
30923                                           deltaTotals,
30924                                           std::string(),
30925                                           std::string(),
30926                                           false));
30927     m_totals.testCases.failed++;
30928     testGroupEnded(std::string(), m_totals, 1, 1);
30929     m_reporter->testRunEnded(TestRunStats(m_runInfo, m_totals, false));
30930 }
30931
30932 bool RunContext::lastAssertionPassed() {
30933     return m_lastAssertionPassed;
30934 }
30935
30936 void RunContext::assertionPassed() {
30937     m_lastAssertionPassed = true;
30938     ++m_totals.assertions.passed;
30939     resetAssertionInfo();
30940     m_messageScopes.clear();
30941 }
30942
30943 bool RunContext::aborting() const {
30944     return m_totals.assertions.failed >= static_cast<std::size_t>(m_config->abortAfter());
30945 }
30946
30947 void RunContext::runCurrentTest(std::string & redirectedCout, std::string & redirectedCerr) {
30948     auto const& testCaseInfo = m_activeTestCase->getTestCaseInfo();
30949     SectionInfo testCaseSection(testCaseInfo.lineInfo, testCaseInfo.name);
30950     m_reporter->sectionStarting(testCaseSection);
30951     Counts prevAssertions = m_totals.assertions;
30952     double duration = 0;
30953     m_shouldReportUnexpected = true;
30954     m_lastAssertionInfo = { "TEST_CASE"_sr, testCaseInfo.lineInfo, StringRef(),
ResultDisposition::Normal };
30955
30956     seedRng(*m_config);
30957
30958     Timer timer;
30959     CATCH_TRY {
30960         if (m_reporter->getPreferences().shouldRedirectStdOut) {
30961             #if !defined(CATCH_CONFIG_EXPERIMENTAL_REDIRECT)
30962                 RedirectedStreams redirectedStreams(redirectedCout, redirectedCerr);
30963
30964                 timer.start();
30965                 invokeActiveTestCase();
30966             #else
30967                 OutputRedirect r(redirectedCout, redirectedCerr);
30968                 timer.start();
30969                 invokeActiveTestCase();
30970             #endif
30971         } else {
30972             timer.start();
30973             invokeActiveTestCase();
30974         }
30975         duration = timer.getElapsedSeconds();
30976     } CATCH_CATCH_ANON (TestFailureException&) {
30977         // This just means the test was aborted due to failure
30978     } CATCH_CATCH_ALL {
30979         // Under CATCH_CONFIG_FAST_COMPILE, unexpected exceptions under REQUIRE assertions
30980         // are reported without translation at the point of origin.
30981         if (m_shouldReportUnexpected) {
30982             AssertionReaction dummyReaction;
30983             handleUnexpectedInflightException(m_lastAssertionInfo, translateActiveException(),
dummyReaction );
30984         }
30985     }
30986     Counts assertions = m_totals.assertions - prevAssertions;
30987     bool missingAssertions = testForMissingAssertions(assertions);
30988
30989     m_testCaseTracker->close();
30990     handleUnfinishedSections();
30991     m_messages.clear();
30992     m_messageScopes.clear();
30993
30994     SectionStats testCaseSectionStats(testCaseSection, assertions, duration, missingAssertions);
30995     m_reporter->sectionEnded(testCaseSectionStats);
30996 }
30997
30998 void RunContext::invokeActiveTestCase() {
30999     FatalConditionHandlerGuard _(&m_fatalConditionhandler);

```



```

31000     m_activeTestCase->invoke();
31001 }
31002
31003 void RunContext::handleUnfinishedSections() {
31004     // If sections ended prematurely due to an exception we stored their
31005     // infos here so we can tear them down outside the unwind process.
31006     for (auto it = m_unfinishedSections.rbegin(),
31007          itEnd = m_unfinishedSections.rend();
31008          it != itEnd;
31009          ++it)
31010         sectionEnded(*it);
31011     m_unfinishedSections.clear();
31012 }
31013
31014 void RunContext::handleExpr(
31015     AssertionInfo const& info,
31016     ITransientExpression const& expr,
31017     AssertionReaction& reaction
31018 ) {
31019     m_reporter->assertionStarting( info );
31020
31021     bool negated = isFalseTest( info.resultDisposition );
31022     bool result = expr.getResult() != negated;
31023
31024     if( result ) {
31025         if (!m_includeSuccessfulResults) {
31026             assertionPassed();
31027         }
31028         else {
31029             reportExpr(info, ResultWas::Ok, &expr, negated);
31030         }
31031     }
31032     else {
31033         reportExpr(info, ResultWas::ExpressionFailed, &expr, negated );
31034         populateReaction( reaction );
31035     }
31036 }
31037 void RunContext::reportExpr(
31038     AssertionInfo const& info,
31039     ResultWas::OfType resultType,
31040     ITransientExpression const& *expr,
31041     bool negated ) {
31042
31043     m_lastAssertionInfo = info;
31044     AssertionResultData data( resultType, LazyExpression( negated ) );
31045
31046     AssertionResult assertionResult{ info, data };
31047     assertionResult.m_resultData.lazyExpression.m_transientExpression = expr;
31048
31049     assertionEnded( assertionResult );
31050 }
31051
31052 void RunContext::handleMessage(
31053     AssertionInfo const& info,
31054     ResultWas::OfType resultType,
31055     StringRef const& message,
31056     AssertionReaction& reaction
31057 ) {
31058     m_reporter->assertionStarting( info );
31059
31060     m_lastAssertionInfo = info;
31061
31062     AssertionResultData data( resultType, LazyExpression( false ) );
31063     data.message = static_cast<std::string>(message);
31064     AssertionResult assertionResult{ m_lastAssertionInfo, data };
31065     assertionEnded( assertionResult );
31066     if( !assertionResult.isOk() )
31067         populateReaction( reaction );
31068 }
31069 void RunContext::handleUnexpectedExceptionNotThrown(
31070     AssertionInfo const& info,
31071     AssertionReaction& reaction
31072 ) {
31073     handleNonExpr(info, Catch::ResultWas::DidntThrowException, reaction);
31074 }
31075
31076 void RunContext::handleUnexpectedInflightException(
31077     AssertionInfo const& info,
31078     std::string const& message,
31079     AssertionReaction& reaction
31080 ) {
31081     m_lastAssertionInfo = info;
31082
31083     AssertionResultData data( ResultWas::ThrowException, LazyExpression( false ) );
31084     data.message = message;
31085     AssertionResult assertionResult{ info, data };
31086     assertionEnded( assertionResult );

```

```

31087     populateReaction( reaction );
31088 }
31089
31090 void RunContext::populateReaction( AssertionReaction& reaction ) {
31091     reaction.shouldDebugBreak = m_config->shouldDebugBreak();
31092     reaction.shouldThrow = aborting() || (m_lastAssertionInfo.resultDisposition &
ResultDisposition::Normal);
31093 }
31094
31095 void RunContext::handleIncomplete(
31096     AssertionInfo const& info
31097 ) {
31098     m_lastAssertionInfo = info;
31099
31100     AssertionResultData data( ResultWas::ThrowException, LazyExpression( false ) );
31101     data.message = "Exception translation was disabled by CATCH_CONFIG_FAST_COMPILE";
31102     AssertionResult assertionResult{ info, data };
31103     assertionEnded( assertionResult );
31104 }
31105 void RunContext::handleNonExpr(
31106     AssertionInfo const& info,
31107     ResultWas::OfType resultType,
31108     AssertionReaction &reaction
31109 ) {
31110     m_lastAssertionInfo = info;
31111
31112     AssertionResultData data( resultType, LazyExpression( false ) );
31113     AssertionResult assertionResult{ info, data };
31114     assertionEnded( assertionResult );
31115
31116     if( !assertionResult.isOk() )
31117         populateReaction( reaction );
31118 }
31119
31120 IResultCapture& getResultCapture() {
31121     if (auto* capture = getCurrentContext().getResultCapture())
31122         return *capture;
31123     else
31124         CATCH_INTERNAL_ERROR("No result capture instance");
31125 }
31126
31127 void seedRng(IConfig const& config) {
31128     if (config.rngSeed() != 0) {
31129         std::srand(config.rngSeed());
31130         rng().seed(config.rngSeed());
31131     }
31132 }
31133
31134 unsigned int rngSeed() {
31135     return getCurrentContext().getConfig()->rngSeed();
31136 }
31137
31138 }
31139 // end catch_run_context.cpp
31140 // start catch_section.cpp
31141
31142 namespace Catch {
31143
31144     Section::Section( SectionInfo const& info )
31145     :   m_info( info ),
31146         m_sectionIncluded( getResultCapture().sectionStarted( m_info, m_assertions ) )
31147     {
31148         m_timer.start();
31149     }
31150
31151     Section::~Section() {
31152         if( m_sectionIncluded ) {
31153             SectionEndInfo endInfo{ m_info, m_assertions, m_timer.getElapsedSeconds() };
31154             if( uncaught_exceptions() )
31155                 getResultCapture().sectionEndedEarly( endInfo );
31156             else
31157                 getResultCapture().sectionEnded( endInfo );
31158         }
31159     }
31160
31161     // This indicates whether the section should be executed or not
31162     Section::operator bool() const {
31163         return m_sectionIncluded;
31164     }
31165
31166 } // end namespace Catch
31167 // end catch_section.cpp
31168 // start catch_section_info.cpp
31169
31170 namespace Catch {
31171
31172     SectionInfo::SectionInfo

```

```

31173         (   SourceLineInfo const& _lineInfo,
31174             std::string const& _name )
31175     :   name( _name ),
31176         lineInfo( _lineInfo )
31177     {}
31178
31179 } // end namespace Catch
31180 // end catch_section_info.cpp
31181 // start catch_session.cpp
31182
31183 // start catch_session.h
31184
31185 #include <memory>
31186
31187 namespace Catch {
31188
31189     class Session : NonCopyable {
31190     public:
31191
31192         Session();
31193         ~Session() override;
31194
31195         void showHelp() const;
31196         void libIdentify();
31197
31198         int applyCommandLine( int argc, char const * const * argv );
31199 #if defined(CATCH_CONFIG_WCHAR) && defined(_WIN32) && defined(UNICODE)
31200         int applyCommandLine( int argc, wchar_t const * const * argv );
31201 #endif
31202
31203         void useConfigData( ConfigData const& configData );
31204
31205         template<typename CharT>
31206         int run(int argc, CharT const * const argv[]) {
31207             if (m_startupExceptions)
31208                 return 1;
31209             int returnCode = applyCommandLine(argc, argv);
31210             if (returnCode == 0)
31211                 returnCode = run();
31212             return returnCode;
31213         }
31214
31215         int run();
31216
31217         clara::Parser const& cli() const;
31218         void cli( clara::Parser const& newParser );
31219         ConfigData& configData();
31220         Config& config();
31221     private:
31222         int runInternal();
31223
31224         clara::Parser m_cli;
31225         ConfigData m_configData;
31226         std::shared_ptr<Config> m_config;
31227         bool m_startupExceptions = false;
31228     };
31229
31230 } // end namespace Catch
31231
31232 // end catch_session.h
31233 // start catch_version.h
31234
31235 #include <iosfwd>
31236
31237 namespace Catch {
31238
31239     // Versioning information
31240     struct Version {
31241         Version( Version const& ) = delete;
31242         Version& operator=( Version const& ) = delete;
31243         Version(      unsigned int _majorVersion,
31244                   unsigned int _minorVersion,
31245                   unsigned int _patchNumber,
31246                   char const * _branchName,
31247                   unsigned int _buildNumber );
31248
31249         unsigned int const majorVersion;
31250         unsigned int const minorVersion;
31251         unsigned int const patchNumber;
31252
31253         // buildNumber is only used if branchName is not null
31254         char const * const branchName;
31255         unsigned int const buildNumber;
31256
31257         friend std::ostream& operator << ( std::ostream& os, Version const& version );
31258     };
31259

```

```

31260     Version const& libraryVersion();
31261 }
31262
31263 // end catch_version.h
31264 #include <cstdlib>
31265 #include <iomanip>
31266 #include <set>
31267 #include <iterator>
31268
31269 namespace Catch {
31270
31271     namespace {
31272         const int MaxExitCode = 255;
31273
31274         IStreamingReporterPtr createReporter(std::string const& reporterName, IConfigPtr const&
31275 config) {
31276             auto reporter = Catch::getRegistryHub().getReporterRegistry().create(reporterName,
31277 config);
31278             CATCH_ENFORCE(reporter, "No reporter registered with name: '" << reporterName << "'");
31279             return reporter;
31280         }
31281
31282         IStreamingReporterPtr makeReporter(std::shared_ptr<Config> const& config) {
31283             if (Catch::getRegistryHub().getReporterRegistry().getListeners().empty()) {
31284                 return createReporter(config->getReporterName(), config);
31285             }
31286             // On older platforms, returning std::unique_ptr<ListeningReporter>
31287             // when the return type is std::unique_ptr<IStreamingReporter>
31288             // doesn't compile without a std::move call. However, this causes
31289             // a warning on newer platforms. Thus, we have to work around
31290             // it a bit and downcast the pointer manually.
31291             auto ret = std::unique_ptr<IStreamingReporter>(new ListeningReporter);
31292             auto& multi = static_cast<ListeningReporter&>(*ret);
31293             auto const& listeners = Catch::getRegistryHub().getReporterRegistry().getListeners();
31294             for (auto const& listener : listeners) {
31295                 multi.addListener(listener->create(Catch::ReporterConfig(config)));
31296             }
31297             multi.addReporter(createReporter(config->getReporterName(), config));
31298             return ret;
31299         }
31300
31301         class TestGroup {
31302         public:
31303             explicit TestGroup(std::shared_ptr<Config> const& config)
31304                 : m_config{config}
31305                 , m_context{config, makeReporter(config)}
31306             {
31307                 auto const& allTestCases = getAllTestCasesSorted(*m_config);
31308                 m_matches = m_config->testSpec().matchesByFilter(allTestCases, *m_config);
31309                 auto const& invalidArgs = m_config->testSpec().getInvalidArgs();
31310
31311                 if (m_matches.empty() && invalidArgs.empty()) {
31312                     for (auto const& test : allTestCases)
31313                         if (!test.isHidden())
31314                             m_tests.emplace(&test);
31315                 } else {
31316                     for (auto const& match : m_matches)
31317                         m_tests.insert(match.tests.begin(), match.tests.end());
31318                 }
31319             }
31320
31321             Totals execute() {
31322                 auto const& invalidArgs = m_config->testSpec().getInvalidArgs();
31323                 Totals totals;
31324                 m_context.testGroupStarting(m_config->name(), 1, 1);
31325                 for (auto const& testCase : m_tests) {
31326                     if (!m_context.aborting())
31327                         totals += m_context.runTest(*testCase);
31328                     else
31329                         m_context.reporter().skipTest(*testCase);
31330                 }
31331
31332                 for (auto const& match : m_matches) {
31333                     if (match.tests.empty()) {
31334                         m_context.reporter().noMatchingTestCases(match.name);
31335                         totals.error = -1;
31336                     }
31337                 }
31338
31339                 if (!invalidArgs.empty()) {
31340                     for (auto const& invalidArg : invalidArgs)
31341                         m_context.reporter().reportInvalidArguments(invalidArg);
31342                 }
31343
31344                 m_context.testGroupEnded(m_config->name(), totals, 1, 1);

```

```

31345         return totals;
31346     }
31347
31348 private:
31349     using Tests = std::set<TestCase const*>;
31350
31351     std::shared_ptr<Config> m_config;
31352     RunContext m_context;
31353     Tests m_tests;
31354     TestSpec::Matches m_matches;
31355 };
31356
31357 void applyFileNamesAsTags(Catch::IConfig const& config) {
31358     auto& tests = const_cast<std::vector<TestCase>&>(getAllTestCasesSorted(config));
31359     for (auto& testCase : tests) {
31360         auto tags = testCase.tags;
31361
31362         std::string filename = testCase.lineInfo.file;
31363         auto lastSlash = filename.find_last_of("\\");
31364         if (lastSlash != std::string::npos) {
31365             filename.erase(0, lastSlash);
31366             filename[0] = '#';
31367         }
31368         else
31369         {
31370             filename.insert(0, "#");
31371         }
31372
31373         auto lastDot = filename.find_last_of('.');
31374         if (lastDot != std::string::npos) {
31375             filename.erase(lastDot);
31376         }
31377
31378         tags.push_back(std::move(filename));
31379         setTags(testCase, tags);
31380     }
31381 }
31382
31383 } // anon namespace
31384
31385 Session::Session() {
31386     static bool alreadyInstantiated = false;
31387     if (alreadyInstantiated) {
31388         CATCH_TRY { CATCH_INTERNAL_ERROR( "Only one instance of Catch::Session can ever be used"
31389 ); }
31390         CATCH_CATCH_ALL { getMutableRegistryHub().registerStartupException(); }
31391     }
31392
31393     // There cannot be exceptions at startup in no-exception mode.
31394 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
31395     const auto& exceptions = getRegistryHub().getStartupExceptionRegistry().getExceptions();
31396     if (!exceptions.empty()) {
31397         config();
31398         getCurrentMutableContext().setConfig(m_config);
31399
31400         m_startupExceptions = true;
31401         Colour colourGuard( Colour::Red );
31402         Catch::cerr() << "Errors occurred during startup!" << '\n';
31403         // iterate over all exceptions and notify user
31404         for ( const auto& ex_ptr : exceptions ) {
31405             try {
31406                 std::rethrow_exception(ex_ptr);
31407             } catch ( std::exception const& ex ) {
31408                 Catch::cerr() << Column( ex.what() ).indent(2) << '\n';
31409             }
31410         }
31411 #endif
31412
31413     alreadyInstantiated = true;
31414     m_cli = makeCommandLineParser( m_configData );
31415 }
31416
31417 Session::~Session() {
31418     Catch::cleanUp();
31419 }
31420
31421 void Session::showHelp() const {
31422     Catch::cout()
31423         << "\nCatch v" << libraryVersion() << "\n"
31424         << m_cli << std::endl
31425         << "For more detailed usage please see the project docs\n" << std::endl;
31426 }
31427
31428 void Session::libIdentify() {
31429     Catch::cout()
31430         << std::left << std::setw(16) << "description: " << "A Catch2 test executable\n"
31431         << std::left << std::setw(16) << "category: " << "testframework\n"
31432         << std::left << std::setw(16) << "framework: " << "Catch Test\n"

```

```

31431         « std::left « std::setw(16) « "version: " « libraryVersion() « std::endl;
31432     }
31433
31434     int Session::applyCommandLine( int argc, char const * const * argv ) {
31435         if( m_startupExceptions )
31436             return 1;
31437
31438         auto result = m_cli.parse( clara::Args( argc, argv ) );
31439         if( !result ) {
31440             config();
31441             getCurrentMutableContext().setConfig(m_config);
31442             Catch::cerr()
31443                 « Colour( Colour::Red )
31444                 « "\nError(s) in input:\n"
31445                 « Column( result.errorMessage() ).indent( 2 )
31446                 « "\n\n";
31447             Catch::cerr() « "Run with -? for usage\n" « std::endl;
31448             return MaxExitCode;
31449         }
31450
31451         if( m_configData.showHelp )
31452             showHelp();
31453         if( m_configData.libIdentify )
31454             libIdentify();
31455         m_config.reset();
31456         return 0;
31457     }
31458
31459 #if defined(CATCH_CONFIG_WCHAR) && defined(_WIN32) && defined(UNICODE)
31460     int Session::applyCommandLine( int argc, wchar_t const * const * argv ) {
31461
31462         char **utf8Argv = new char * [ argc ];
31463
31464         for ( int i = 0; i < argc; ++i ) {
31465             int bufSize = WideCharToMultiByte( CP_UTF8, 0, argv[i], -1, nullptr, 0, nullptr, nullptr );
31466         };
31467
31468         utf8Argv[ i ] = new char[ bufSize ];
31469
31470         WideCharToMultiByte( CP_UTF8, 0, argv[i], -1, utf8Argv[i], bufSize, nullptr, nullptr );
31471     }
31472
31473     int returnCode = applyCommandLine( argc, utf8Argv );
31474
31475     for ( int i = 0; i < argc; ++i )
31476         delete [] utf8Argv[ i ];
31477
31478     delete [] utf8Argv;
31479
31480     return returnCode;
31481 #endif
31482
31483     void Session::useConfigData( ConfigData const& configData ) {
31484         m_configData = configData;
31485         m_config.reset();
31486     }
31487
31488     int Session::run() {
31489         if( ( m_configData.waitForKeypress & WaitForKeypress::BeforeStart ) != 0 ) {
31490             Catch::cout() « "...waiting for enter/ return before starting" « std::endl;
31491             static_cast<void>(std::getchar());
31492         }
31493         int exitCode = runInternal();
31494         if( ( m_configData.waitForKeypress & WaitForKeypress::BeforeExit ) != 0 ) {
31495             Catch::cout() « "...waiting for enter/ return before exiting, with code: " « exitCode «
std::endl;
31496             static_cast<void>(std::getchar());
31497         }
31498         return exitCode;
31499     }
31500
31501     clara::Parser const& Session::cli() const {
31502         return m_cli;
31503     }
31504     void Session::cli( clara::Parser const& newParser ) {
31505         m_cli = newParser;
31506     }
31507     ConfigData& Session::configData() {
31508         return m_configData;
31509     }
31510     Config& Session::config() {
31511         if( !m_config )
31512             m_config = std::make_shared<Config>( m_configData );
31513         return *m_config;
31514     }
31515

```

```

31516     int Session::runInternal() {
31517         if( m_startupExceptions )
31518             return 1;
31519
31520         if (m_configData.showHelp || m_configData.libIdentify) {
31521             return 0;
31522         }
31523
31524         CATCH_TRY {
31525             config(); // Force config to be constructed
31526
31527             seedRng( *m_config );
31528
31529             if( m_configData_filenamesAsTags )
31530                 applyFilenamesAsTags( *m_config );
31531
31532             // Handle list request
31533             if( Option<std::size_t> listed = list( m_config ) )
31534                 return (std::min)(MaxExitCode, static_cast<int>(*listed));
31535
31536             TestGroup tests { m_config };
31537             auto const totals = tests.execute();
31538
31539             if( m_config->warnAboutNoTests() && totals.error == -1 )
31540                 return 2;
31541
31542             // Note that on unices only the lower 8 bits are usually used, clamping
31543             // the return value to 255 prevents false negative when some multiple
31544             // of 256 tests has failed
31545             return (std::min)(MaxExitCode, (std::max)(totals.error,
31546                 static_cast<int>(totals.assertions.failed)));
31547         }
31548         #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
31549         catch( std::exception& ex ) {
31550             Catch::cerr() << ex.what() << std::endl;
31551             return MaxExitCode;
31552         }
31553         #endif
31554     } // end namespace Catch
31555 // end catch_session.cpp
31556 // start catch_singletons.cpp
31557 #include <vector>
31558 namespace Catch {
31559     namespace {
31560         static auto getSingletons() -> std::vector<ISingleton*>*& {
31561             static std::vector<ISingleton*>* g_singletons = nullptr;
31562             if( !g_singletons )
31563                 g_singletons = new std::vector<ISingleton*>();
31564             return g_singletons;
31565         }
31566     }
31567     ISingleton::~ISingleton() {}
31568     void addSingleton(ISingleton* singleton) {
31569         getSingletons()->push_back( singleton );
31570     }
31571     void cleanupSingletons() {
31572         auto& singletons = getSingletons();
31573         for( auto singleton : *singletons )
31574             delete singleton;
31575         delete singletons;
31576         singletons = nullptr;
31577     }
31578 } // namespace Catch
31579 // end catch_singletons.cpp
31580 // start catch_startup_exception_registry.cpp
31581 #if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
31582 namespace Catch {
31583     void StartupExceptionRegistry::add( std::exception_ptr const& exception ) noexcept {
31584         CATCH_TRY {
31585             m_exceptions.push_back(exception);
31586         } CATCH_CATCH_ALL {
31587             // If we run out of memory during start-up there's really not a lot more we can do about
31588             it
31589             std::terminate();
31590         }
31591     }
31592     std::vector<std::exception_ptr> const& StartupExceptionRegistry::getExceptions() const noexcept {

```

```

31601         return m_exceptions;
31602     }
31603
31604 } // end namespace Catch
31605 #endif
31606 // end catch_startup_exception_registry.cpp
31607 // start catch_stream.cpp
31608
31609 #include <cstdio>
31610 #include <iostream>
31611 #include <fstream>
31612 #include <sstream>
31613 #include <vector>
31614 #include <memory>
31615
31616 namespace Catch {
31617     Catch::IStream::~IStream() = default;
31618
31619     namespace Detail { namespace {
31620         template<typename WriterF, std::size_t bufferSize=256>
31621         class StreamBufImpl : public std::streambuf {
31622             char data[bufferSize];
31623             WriterF m_writer;
31624
31625         public:
31626             StreamBufImpl() {
31627                 setp( data, data + sizeof(data) );
31628             }
31629
31630             ~StreamBufImpl() noexcept {
31631                 StreamBufImpl::sync();
31632             }
31633
31634         private:
31635             int overflow( int c ) override {
31636                 sync();
31637
31638                 if( c != EOF ) {
31639                     if( pbase() == epptr() )
31640                         m_writer( std::string( 1, static_cast<char>( c ) ) );
31641                     else
31642                         sputc( static_cast<char>( c ) );
31643                 }
31644                 return 0;
31645             }
31646
31647             int sync() override {
31648                 if( pbase() != pptr() ) {
31649                     m_writer( std::string( pbase(), static_cast<std::string::size_type>( pptr() -
31650 pbase() ) ) );
31651                     setp( pbase(), epptr() );
31652                 }
31653                 return 0;
31654             }
31655         };
31656
31657         struct OutputDebugWriter {
31658             void operator()( std::string const&str ) {
31659                 writeToDebugConsole( str );
31660             }
31661         };
31662
31663         class FileStream : public IStream {
31664             mutable std::ofstream m_ofs;
31665         public:
31666             FileStream( StringRef filename ) {
31667                 m_ofs.open( filename.c_str() );
31668                 CATCH_ENFORCE( !m_ofs.fail(), "Unable to open file: " << filename << " " );
31669             }
31670             ~FileStream() override = default;
31671         public: // IStream
31672             std::ostream& stream() const override {
31673                 return m_ofs;
31674             }
31675         };
31676
31677         class CoutStream : public IStream {
31678             mutable std::ostream m_os;
31679         public:
31680             // Store the streambuf from cout up-front because
31681             // cout may get redirected when running tests
31682             CoutStream() : m_os( Catch::cout().rdbuf() ) {}
31683         }
31684     } }

```



```

31690         ~CoutStream() override = default;
31691
31692     public: // IStream
31693         std::ostream& stream() const override { return m_os; }
31694     };
31695
31696     class DebugOutStream : public IStream {
31697     public:
31698         std::unique_ptr<StreamBufImpl<OutputDebugWriter>> m_streamBuf;
31699         mutable std::ostream m_os;
31700     public:
31701         DebugOutStream()
31702         : m_streamBuf( new StreamBufImpl<OutputDebugWriter>() ),
31703           m_os( m_streamBuf.get() )
31704         {}
31705
31706         ~DebugOutStream() override = default;
31707
31708     public: // IStream
31709         std::ostream& stream() const override { return m_os; }
31710     };
31711
31712 } // namespace anon::detail
31713
31714 auto makeStream( StringRef const &filename ) -> IStream const* {
31715     if( filename.empty() )
31716         return new Detail::CoutStream();
31717     else if( filename[0] == '%' ) {
31718         if( filename == "%debug" )
31719             return new Detail::DebugOutStream();
31720         else
31721             CATCH_ERROR( "Unrecognised stream: '" << filename << "'" );
31722     }
31723     else
31724         return new Detail::FileStream( filename );
31725 }
31726
31727 // This class encapsulates the idea of a pool of ostringstreams that can be reused.
31728 struct StringStreams {
31729     std::vector<std::unique_ptr<std::ostringstream>> m_streams;
31730     std::vector<std::size_t> m_unused;
31731     std::ostringstream m_referenceStream; // Used for copy state/ flags from
31732
31733     auto add() -> std::size_t {
31734         if( m_unused.empty() ) {
31735             m_streams.push_back( std::unique_ptr<std::ostringstream>( new std::ostringstream ) );
31736             return m_streams.size()-1;
31737         }
31738         else {
31739             auto index = m_unused.back();
31740             m_unused.pop_back();
31741             return index;
31742         }
31743     }
31744
31745     void release( std::size_t index ) {
31746         m_streams[index]->copyfmt( m_referenceStream ); // Restore initial flags and other state
31747         m_unused.push_back(index);
31748     }
31749 };
31750
31751 ReusableStringStream::ReusableStringStream()
31752 : m_index( Singleton<StringStreams>::getMutable().add() ),
31753   m_oss( Singleton<StringStreams>::getMutable().m_streams[m_index].get() )
31754 {}
31755
31756 ReusableStringStream::~ReusableStringStream() {
31757     static_cast<std::ostringstream*>( m_oss )->str("");
31758     m_oss->clear();
31759     Singleton<StringStreams>::getMutable().release( m_index );
31760 }
31761
31762 auto ReusableStringStream::str() const -> std::string {
31763     return static_cast<std::ostringstream*>( m_oss )->str();
31764 }
31765
31766 #ifndef CATCH_CONFIG_NOSTDOUT // If you #define this you must implement these functions
31767 std::ostream& cout() { return std::cout; }
31768 std::ostream& cerr() { return std::cerr; }
31769 std::ostream& clog() { return std::clog; }
31770 #endif
31771
31772 // end catch_stream.cpp
31773 // start catch_string_manip.cpp
31774

```

```

31780 #include <algorithm>
31781 #include <ostream>
31782 #include <cstring>
31783 #include <cctype>
31784 #include <vector>
31785
31786 namespace Catch {
31787
31788     namespace {
31789         char toLowerCh(char c) {
31790             return static_cast<char>( std::tolower( static_cast<unsigned char>(c) ) );
31791         }
31792     }
31793
31794     bool startsWith( std::string const& s, std::string const& prefix ) {
31795         return s.size() >= prefix.size() && std::equal(prefix.begin(), prefix.end(), s.begin());
31796     }
31797     bool startsWith( std::string const& s, char prefix ) {
31798         return !s.empty() && s[0] == prefix;
31799     }
31800     bool endsWith( std::string const& s, std::string const& suffix ) {
31801         return s.size() >= suffix.size() && std::equal(suffix.rbegin(), suffix.rend(), s.rbegin());
31802     }
31803     bool endsWith( std::string const& s, char suffix ) {
31804         return !s.empty() && s[s.size()-1] == suffix;
31805     }
31806     bool contains( std::string const& s, std::string const& infix ) {
31807         return s.find( infix ) != std::string::npos;
31808     }
31809     void toLowerInPlace( std::string& s ) {
31810         std::transform( s.begin(), s.end(), s.begin(), toLowerCh );
31811     }
31812     std::string toLower( std::string const& s ) {
31813         std::string lc = s;
31814         toLowerInPlace( lc );
31815         return lc;
31816     }
31817     std::string trim( std::string const& str ) {
31818         static char const* whitespaceChars = "\n\r\t ";
31819         std::string::size_type start = str.find_first_not_of( whitespaceChars );
31820         std::string::size_type end = str.find_last_not_of( whitespaceChars );
31821
31822         return start != std::string::npos ? str.substr( start, 1+end-start ) : std::string();
31823     }
31824
31825     StringRef trim(StringRef ref) {
31826         const auto is_ws = [](char c) {
31827             return c == ' ' || c == '\t' || c == '\n' || c == '\r';
31828         };
31829         size_t real_begin = 0;
31830         while (real_begin < ref.size() && is_ws(ref[real_begin])) { ++real_begin; }
31831         size_t real_end = ref.size();
31832         while (real_end > real_begin && is_ws(ref[real_end - 1])) { --real_end; }
31833
31834         return ref.substr(real_begin, real_end - real_begin);
31835     }
31836
31837     bool replaceInPlace( std::string& str, std::string const& replaceThis, std::string const& withThis
31838 ) {
31839         bool replaced = false;
31840         std::size_t i = str.find( replaceThis );
31841         while( i != std::string::npos ) {
31842             replaced = true;
31843             str = str.substr( 0, i ) + withThis + str.substr( i+replaceThis.size() );
31844             i = str.find( replaceThis, i+withThis.size() );
31845         }
31846         i = std::string::npos;
31847         return replaced;
31848     }
31849
31850     std::vector<StringRef> splitStringRef( StringRef str, char delimiter ) {
31851         std::vector<StringRef> subStrings;
31852         std::size_t start = 0;
31853         for( std::size_t pos = 0; pos < str.size(); ++pos ) {
31854             if( str[pos] == delimiter ) {
31855                 if( pos - start > 1 )
31856                     subStrings.push_back( str.substr( start, pos-start ) );
31857                 start = pos+1;
31858             }
31859         }
31860         if( start < str.size() )
31861             subStrings.push_back( str.substr( start, str.size()-start ) );
31862         return subStrings;
31863     }
31864 }
31865

```

```

31866     pluralise::pluralise( std::size_t count, std::string const& label )
31867     :   m_count( count ),
31868         m_label( label )
31869     {}
31870
31871     std::ostream& operator << ( std::ostream& os, pluralise const& pluraliser ) {
31872         os << pluraliser.m_count << ' ' << pluraliser.m_label;
31873         if( pluraliser.m_count != 1 )
31874             os << 's';
31875         return os;
31876     }
31877
31878 }
31879 // end catch_string_manip.cpp
31880 // start catch_stringref.cpp
31881
31882 #include <algorithm>
31883 #include <ostream>
31884 #include <cstring>
31885 #include <cstdint>
31886
31887 namespace Catch {
31888     StringRef::StringRef( char const* rawChars ) noexcept
31889     : StringRef( rawChars, static_cast<StringRef::size_type>(std::strlen(rawChars) ) )
31890     {}
31891
31892     auto StringRef::c_str() const -> char const* {
31893         CATCH_ENFORCE(isNullTerminated(), "Called StringRef::c_str() on a non-null-terminated
instance");
31894         return m_start;
31895     }
31896     auto StringRef::data() const noexcept -> char const* {
31897         return m_start;
31898     }
31899
31900     auto StringRef::substr( size_type start, size_type size ) const noexcept -> StringRef {
31901         if (start < m_size) {
31902             return StringRef(m_start + start, (std::min)(m_size - start, size));
31903         } else {
31904             return StringRef();
31905         }
31906     }
31907     auto StringRef::operator == ( StringRef const& other ) const noexcept -> bool {
31908         return m_size == other.m_size
31909             && (std::memcmp( m_start, other.m_start, m_size ) == 0);
31910     }
31911
31912     auto operator << ( std::ostream& os, StringRef const& str ) -> std::ostream& {
31913         return os.write(str.data(), str.size());
31914     }
31915
31916     auto operator+=( std::string& lhs, StringRef const& rhs ) -> std::string& {
31917         lhs.append(rhs.data(), rhs.size());
31918         return lhs;
31919     }
31920
31921 } // namespace Catch
31922 // end catch_stringref.cpp
31923 // start catch_tag_alias.cpp
31924
31925 namespace Catch {
31926     TagAlias::TagAlias(std::string const& _tag, SourceLineInfo _lineInfo): tag(_tag),
lineInfo(_lineInfo) {}
31927 }
31928 // end catch_tag_alias.cpp
31929 // start catch_tag_alias_autoregistrar.cpp
31930
31931 namespace Catch {
31932
31933     RegistrarForTagAliases::RegistrarForTagAliases(char const* alias, char const* tag, SourceLineInfo
const& lineInfo) {
31934         CATCH_TRY {
31935             getMutableRegistryHub().registerTagAlias(alias, tag, lineInfo);
31936         } CATCH_CATCH_ALL {
31937             // Do not throw when constructing global objects, instead register the exception to be
processed later
31938             getMutableRegistryHub().registerStartupException();
31939         }
31940     }
31941
31942 }
31943 // end catch_tag_alias_autoregistrar.cpp
31944 // start catch_tag_alias_registry.cpp
31945
31946 #include <sstream>
31947
31948 namespace Catch {

```

```

31949
31950     TagAliasRegistry::~TagAliasRegistry() {}
31951
31952     TagAlias const* TagAliasRegistry::find( std::string const& alias ) const {
31953         auto it = m_registry.find( alias );
31954         if( it != m_registry.end() )
31955             return &(it->second);
31956         else
31957             return nullptr;
31958     }
31959
31960     std::string TagAliasRegistry::expandAliases( std::string const& unexpandedTestSpec ) const {
31961         std::string expandedTestSpec = unexpandedTestSpec;
31962         for( auto const& registryKvp : m_registry ) {
31963             std::size_t pos = expandedTestSpec.find( registryKvp.first );
31964             if( pos != std::string::npos ) {
31965                 expandedTestSpec = expandedTestSpec.substr( 0, pos ) +
31966                                     registryKvp.second.tag +
31967                                     expandedTestSpec.substr( pos + registryKvp.first.size() );
31968             }
31969         }
31970         return expandedTestSpec;
31971     }
31972
31973     void TagAliasRegistry::add( std::string const& alias, std::string const& tag, SourceLineInfo
const& lineInfo ) {
31974         CATCH_ENFORCE( startsWith(alias, "[@]") && endsWith(alias, ']'),
31975             "error: tag alias, '" < alias < "' is not of the form [@alias name].\n" <
lineInfo );
31976
31977         CATCH_ENFORCE( m_registry.insert(std::make_pair(alias, TagAlias(tag, lineInfo))).second,
31978             "error: tag alias, '" < alias < "' already registered.\n"
31979             < "\tFirst seen at: " < find(alias)->lineInfo < "\n"
31980             < "\tRedefined at: " < lineInfo );
31981     }
31982
31983     ITagAliasRegistry::~ITagAliasRegistry() {}
31984
31985     ITagAliasRegistry const& ITagAliasRegistry::get() {
31986         return getRegistryHub().getTagAliasRegistry();
31987     }
31988
31989 } // end namespace Catch
31990 // end catch_tag_alias_registry.cpp
31991 // start catch_test_case_info.cpp
31992
31993 #include <cctype>
31994 #include <exception>
31995 #include <algorithm>
31996 #include <sstream>
31997
31998 namespace Catch {
31999
40000     namespace {
40001         TestCaseInfo::SpecialProperties parseSpecialTag( std::string const& tag ) {
40002             if( startsWith( tag, '.' ) ||
40003                 tag == "!hide" )
40004                 return TestCaseInfo::IsHidden;
40005             else if( tag == "!throws" )
40006                 return TestCaseInfo::Throws;
40007             else if( tag == "!shouldfail" )
40008                 return TestCaseInfo::ShouldFail;
40009             else if( tag == "!mayfail" )
40010                 return TestCaseInfo::MayFail;
40011             else if( tag == "!nonportable" )
40012                 return TestCaseInfo::NonPortable;
40013             else if( tag == "!benchmark" )
40014                 return static_cast<TestCaseInfo::SpecialProperties>( TestCaseInfo::Benchmark |
TestCaseInfo::IsHidden );
40015             else
40016                 return TestCaseInfo::None;
40017         }
40018
40019         bool isReservedTag( std::string const& tag ) {
40020             return parseSpecialTag( tag ) == TestCaseInfo::None && tag.size() > 0 && !std::isalnum(
static_cast<unsigned char>(tag[0]) );
40021         }
40022
40023         void enforceNotReservedTag( std::string const& tag, SourceLineInfo const& _lineInfo ) {
40024             CATCH_ENFORCE( !isReservedTag(tag),
40025                 "Tag name: ['" < tag < "'] is not allowed.\n"
40026                 < "Tag names starting with non alphanumeric characters are reserved\n"
40027                 < _lineInfo );
40028         }
40029
40030         TestCase makeTestCase( ITestInvoker* _testCase,
40031             std::string const& _className,
40032             NameAndTags const& nameAndTags,

```

```

32032             SourceLineInfo const& _lineInfo )
32033     {
32034         bool isHidden = false;
32035
32036         // Parse out tags
32037         std::vector<std::string> tags;
32038         std::string desc, tag;
32039         bool inTag = false;
32040         for (char c : nameAndTags.tags) {
32041             if( !inTag ) {
32042                 if( c == '[' )
32043                     inTag = true;
32044                 else
32045                     desc += c;
32046             }
32047             else {
32048                 if( c == ']' ) {
32049                     TestCaseInfo::SpecialProperties prop = parseSpecialTag( tag );
32050                     if( ( prop & TestCaseInfo::IsHidden ) != 0 )
32051                         isHidden = true;
32052                     else if( prop == TestCaseInfo::None )
32053                         enforceNotReservedTag( tag, _lineInfo );
32054
32055                     // Merged hide tags like `[.approvals]` should be added as
32056                     // `[.][approvals]`. The `[.]` is added at later point, so
32057                     // we only strip the prefix
32058                     if (startsWith(tag, '.') && tag.size() > 1) {
32059                         tag.erase(0, 1);
32060                     }
32061                     tags.push_back( tag );
32062                     tag.clear();
32063                     inTag = false;
32064                 }
32065                 else
32066                     tag += c;
32067             }
32068         }
32069         if( isHidden ) {
32070             // Add all "hidden" tags to make them behave identically
32071             tags.insert( tags.end(), { ".", "!hide" } );
32072         }
32073
32074         TestCaseInfo info( static_cast<std::string>(nameAndTags.name), _className, desc, tags,
32075 _lineInfo );
32076         return TestCase( _testCase, std::move(info) );
32077     }
32078
32079     void setTags( TestCaseInfo& testCaseInfo, std::vector<std::string> tags ) {
32080         std::sort(begin(tags), end(tags));
32081         tags.erase(std::unique(begin(tags), end(tags)), end(tags));
32082         testCaseInfo.lcaseTags.clear();
32083
32084         for( auto const& tag : tags ) {
32085             std::string lcaseTag = toLower( tag );
32086             testCaseInfo.properties = static_cast<TestCaseInfo::SpecialProperties>(
32087                 testCaseInfo.properties | parseSpecialTag( lcaseTag ) );
32088             testCaseInfo.lcaseTags.push_back( lcaseTag );
32089         }
32090         testCaseInfo.tags = std::move(tags);
32091     }
32092
32093     TestCaseInfo::TestCaseInfo( std::string const& _name,
32094                                std::string const& _className,
32095                                std::string const& _description,
32096                                std::vector<std::string> const& _tags,
32097                                SourceLineInfo const& _lineInfo )
32098     :   name( _name ),
32099         className( _className ),
32100         description( _description ),
32101         lineInfo( _lineInfo ),
32102         properties( None )
32103     {
32104         setTags( *this, _tags );
32105     }
32106
32107     bool TestCaseInfo::isHidden() const {
32108         return ( properties & IsHidden ) != 0;
32109     }
32110
32111     bool TestCaseInfo::throws() const {
32112         return ( properties & Throws ) != 0;
32113     }
32114
32115     bool TestCaseInfo::okToFail() const {
32116         return ( properties & (ShouldFail | MayFail) ) != 0;
32117     }
32118
32119     bool TestCaseInfo::expectedToFail() const {
32120         return ( properties & (ShouldFail) ) != 0;
32121     }

```

```

32117
32118     std::string TestCaseInfo::tagsAsString() const {
32119         std::string ret;
32120         // '[' and ']' per tag
32121         std::size_t full_size = 2 * tags.size();
32122         for (const auto& tag : tags) {
32123             full_size += tag.size();
32124         }
32125         ret.reserve(full_size);
32126         for (const auto& tag : tags) {
32127             ret.push_back('[');
32128             ret.append(tag);
32129             ret.push_back(']');
32130         }
32131
32132         return ret;
32133     }
32134
32135     TestCase::TestCase( ITestInvoker* testCase, TestCaseInfo&& info ) : TestCaseInfo( std::move(info)
), test( testCase ) {}
32136
32137     TestCase TestCase::withName( std::string const& _newName ) const {
32138         TestCase other( *this );
32139         other.name = _newName;
32140         return other;
32141     }
32142
32143     void TestCase::invoke() const {
32144         test->invoke();
32145     }
32146
32147     bool TestCase::operator == ( TestCase const& other ) const {
32148         return test.get() == other.test.get() &&
32149             name == other.name &&
32150             className == other.className;
32151     }
32152
32153     bool TestCase::operator < ( TestCase const& other ) const {
32154         return name < other.name;
32155     }
32156
32157     TestCaseInfo const& TestCase::getTestCaseInfo() const
32158     {
32159         return *this;
32160     }
32161
32162 } // end namespace Catch
32163 // end catch_test_case_info.cpp
32164 // start catch_test_case_registry_impl.cpp
32165
32166 #include <algorithm>
32167 #include <sstream>
32168
32169 namespace Catch {
32170
32171     namespace {
32172         struct TestHasher {
32173             using hash_t = uint64_t;
32174
32175             explicit TestHasher( hash_t hashSuffix ) :
32176                 m_hashSuffix{ hashSuffix } {}
32177
32178             uint32_t operator()( TestCase const& t ) const {
32179                 // FNV-1a hash with multiplication fold.
32180                 const hash_t prime = 1099511628211u;
32181                 hash_t hash = 14695981039346656037u;
32182                 for ( const char c : t.name ) {
32183                     hash ^= c;
32184                     hash *= prime;
32185                 }
32186                 hash ^= m_hashSuffix;
32187                 hash *= prime;
32188                 const uint32_t low{ static_cast<uint32_t>( hash ) };
32189                 const uint32_t high{ static_cast<uint32_t>( hash >> 32 ) };
32190                 return low * high;
32191             }
32192
32193             private:
32194                 hash_t m_hashSuffix;
32195         };
32196     } // end unnamed namespace
32197
32198     std::vector<TestCase> sortTests( IConfig const& config, std::vector<TestCase> const&
unsortedTestCases ) {
32199         switch( config.runOrder() ) {
32200             case RunTests::InDeclarationOrder:
32201                 // already in declaration order

```

```

32202         break;
32203
32204     case RunTests::InLexicographicalOrder: {
32205         std::vector<TestCase> sorted = unsortedTestCases;
32206         std::sort( sorted.begin(), sorted.end() );
32207         return sorted;
32208     }
32209
32210     case RunTests::InRandomOrder: {
32211         seedRng( config );
32212         TestHasher h{ config.rngSeed() };
32213
32214         using hashedTest = std::pair<TestHasher::hash_t, TestCase const*>;
32215         std::vector<hashedTest> indexed_tests;
32216         indexed_tests.reserve( unsortedTestCases.size() );
32217
32218         for (auto const& testCase : unsortedTestCases) {
32219             indexed_tests.emplace_back(h(testCase), &testCase);
32220         }
32221
32222         std::sort(indexed_tests.begin(), indexed_tests.end(),
32223             [](hashedTest const& lhs, hashedTest const& rhs) {
32224                 if (lhs.first == rhs.first) {
32225                     return lhs.second->name < rhs.second->name;
32226                 }
32227                 return lhs.first < rhs.first;
32228             });
32229
32230         std::vector<TestCase> sorted;
32231         sorted.reserve( indexed_tests.size() );
32232
32233         for (auto const& hashed : indexed_tests) {
32234             sorted.emplace_back(*hashed.second);
32235         }
32236
32237         return sorted;
32238     }
32239 }
32240 return unsortedTestCases;
32241 }
32242
32243 bool isThrowSafe( TestCase const& testCase, IConfig const& config ) {
32244     return !testCasethrows() || config.allowThrows();
32245 }
32246
32247 bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config ) {
32248     return testSpec.matches( testCase ) && isThrowSafe( testCase, config );
32249 }
32250
32251 void enforceNoDuplicateTestCases( std::vector<TestCase> const& functions ) {
32252     std::set<TestCase> seenFunctions;
32253     for( auto const& function : functions ) {
32254         auto prev = seenFunctions.insert( function );
32255         CATCH_ENFORCE( prev.second,
32256             "error: TEST_CASE( \"" << function.name << "\"" ) already defined.\n"
32257             << "\tFirst seen at " << prev.first->getTestCaseInfo().lineInfo << "\n"
32258             << "\tRedefined at " << function.getTestCaseInfo().lineInfo );
32259     }
32260 }
32261
32262 std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
testSpec, IConfig const& config ) {
32263     std::vector<TestCase> filtered;
32264     filtered.reserve( testCases.size() );
32265     for (auto const& testCase : testCases) {
32266         if ( (!testSpec.hasFilters() && !testCase.isHidden()) ||
32267             (testSpec.hasFilters() && matchTest(testCase, testSpec, config)) ) {
32268             filtered.push_back(testCase);
32269         }
32270     }
32271     return filtered;
32272 }
32273
32274 std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config ) {
32275     return getRegistryHub().getTestCaseRegistry().getAllTestsSorted( config );
32276 }
32277
32278 void TestRegistry::registerTest( TestCase const& testCase ) {
32279     std::string name = testCase.getTestCaseInfo().name;
32280     if( name.empty() ) {
32281         ReusableStringStream rss;
32282         rss << "Anonymous test case " << ++m_unnamedCount;
32283         return registerTest( testCase.withName( rss.str() ) );
32284     }
32285     m_functions.push_back( testCase );
32286 }
32287
32288 std::vector<TestCase> const& TestRegistry::getAllTests() const {

```

```

32288         return m_functions;
32289     }
32290     std::vector<TestCase> const& TestRegistry::getAllTestsSorted( IConfig const& config ) const {
32291         if( m_sortedFunctions.empty() )
32292             enforceNoDuplicateTestCases( m_functions );
32293
32294         if( m_currentSortOrder != config.runOrder() || m_sortedFunctions.empty() ) {
32295             m_sortedFunctions = sortTests( config, m_functions );
32296             m_currentSortOrder = config.runOrder();
32297         }
32298         return m_sortedFunctions;
32299     }
32300
32301     TestInvokerAsFunction::TestInvokerAsFunction( void(*testAsFunction)() ) noexcept :
32302     m_testAsFunction( testAsFunction ) {}
32303
32304     void TestInvokerAsFunction::invoke() const {
32305         m_testAsFunction();
32306     }
32307
32308     std::string extractClassName( StringRef const& classOrQualifiedMethodName ) {
32309         std::string className(classOrQualifiedMethodName);
32310         if( startsWith( className, '&' ) )
32311         {
32312             std::size_t lastColons = className.rfind( "::" );
32313             std::size_t penultimateColons = className.rfind( "::", lastColons-1 );
32314             if( penultimateColons == std::string::npos )
32315                 penultimateColons = 1;
32316             className = className.substr( penultimateColons, lastColons-penultimateColons );
32317         }
32318         return className;
32319     }
32320
32321 } // end namespace Catch
32322 // end catch_test_case_registry_impl.cpp
32323 // start catch_test_case_tracker.cpp
32324
32325 #include <algorithm>
32326 #include <cassert>
32327 #include <stdexcept>
32328 #include <memory>
32329 #include <sstream>
32330
32331 #if defined(__clang__)
32332 #   pragma clang diagnostic push
32333 #   pragma clang diagnostic ignored "-Wexit-time-destructors"
32334 #endif
32335
32336 namespace Catch {
32337     namespace TestCaseTracking {
32338
32339         NameAndLocation::NameAndLocation( std::string const& _name, SourceLineInfo const& _location )
32340         :   name( _name ),
32341             location( _location )
32342         {}
32343
32344         ITracker::~ITracker() = default;
32345
32346         ITracker& TrackerContext::startRun() {
32347             m_rootTracker = std::make_shared<SectionTracker>( NameAndLocation( "{root}",
32348             CATCH_INTERNAL_LINEINFO ), *this, nullptr );
32349             m_currentTracker = nullptr;
32350             m_runState = Executing;
32351             return *m_rootTracker;
32352         }
32353
32354         void TrackerContext::endRun() {
32355             m_rootTracker.reset();
32356             m_currentTracker = nullptr;
32357             m_runState = NotStarted;
32358         }
32359
32360         void TrackerContext::startCycle() {
32361             m_currentTracker = m_rootTracker.get();
32362             m_runState = Executing;
32363         }
32364         void TrackerContext::completeCycle() {
32365             m_runState = CompletedCycle;
32366         }
32367
32368         bool TrackerContext::completedCycle() const {
32369             return m_runState == CompletedCycle;
32370         }
32371
32372         ITracker& TrackerContext::currentTracker() {
32373             return *m_currentTracker;
32374         }
32375         void TrackerContext::setCurrentTracker( ITracker* tracker ) {
32376             m_currentTracker = tracker;
32377         }
32378     }
32379 }

```



```

32374         m_currentTracker = tracker;
32375     }
32376
32377     TrackerBase::TrackerBase( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker*
parent ):
32378         ITracker(nameAndLocation),
32379         m_ctx( ctx ),
32380         m_parent( parent )
32381     {}
32382
32383     bool TrackerBase::isComplete() const {
32384         return m_runState == CompletedSuccessfully || m_runState == Failed;
32385     }
32386     bool TrackerBase::isSuccessfullyCompleted() const {
32387         return m_runState == CompletedSuccessfully;
32388     }
32389     bool TrackerBase::isOpen() const {
32390         return m_runState != NotStarted && !isComplete();
32391     }
32392     bool TrackerBase::hasChildren() const {
32393         return !m_children.empty();
32394     }
32395
32396     void TrackerBase::addChild( ITrackerPtr const& child ) {
32397         m_children.push_back( child );
32398     }
32399
32400     ITrackerPtr TrackerBase::findChild( NameAndLocation const& nameAndLocation ) {
32401         auto it = std::find_if( m_children.begin(), m_children.end(),
32402             [&nameAndLocation]( ITrackerPtr const& tracker ){
32403                 return
32404                     tracker->nameAndLocation().location == nameAndLocation.location &&
32405                     tracker->nameAndLocation().name == nameAndLocation.name;
32406             } );
32407         return( it != m_children.end() )
32408             ? *it
32409             : nullptr;
32410     }
32411     ITracker& TrackerBase::parent() {
32412         assert( m_parent ); // Should always be non-null except for root
32413         return *m_parent;
32414     }
32415
32416     void TrackerBase::openChild() {
32417         if( m_runState != ExecutingChildren ) {
32418             m_runState = ExecutingChildren;
32419             if( m_parent )
32420                 m_parent->openChild();
32421         }
32422     }
32423
32424     bool TrackerBase::isSectionTracker() const { return false; }
32425     bool TrackerBase::isGeneratorTracker() const { return false; }
32426
32427     void TrackerBase::open() {
32428         m_runState = Executing;
32429         moveToThis();
32430         if( m_parent )
32431             m_parent->openChild();
32432     }
32433
32434     void TrackerBase::close() {
32435
32436         // Close any still open children (e.g. generators)
32437         while( &m_ctx.currentTracker() != this )
32438             m_ctx.currentTracker().close();
32439
32440         switch( m_runState ) {
32441             case NeedsAnotherRun:
32442                 break;
32443
32444             case Executing:
32445                 m_runState = CompletedSuccessfully;
32446                 break;
32447             case ExecutingChildren:
32448                 if( std::all_of(m_children.begin(), m_children.end(), []( ITrackerPtr const& t){ return
32449 t->isComplete(); }) )
32450                     m_runState = CompletedSuccessfully;
32451                 break;
32452             case NotStarted:
32453             case CompletedSuccessfully:
32454             case Failed:
32455                 CATCH_INTERNAL_ERROR( "Illogical state: " « m_runState );
32456
32457             default:
32458                 CATCH_INTERNAL_ERROR( "Unknown state: " « m_runState );

```

```

32459         }
32460         moveToParent();
32461         m_ctx.completeCycle();
32462     }
32463     void TrackerBase::fail() {
32464         m_runState = Failed;
32465         if( m_parent )
32466             m_parent->markAsNeedingAnotherRun();
32467         moveToParent();
32468         m_ctx.completeCycle();
32469     }
32470     void TrackerBase::markAsNeedingAnotherRun() {
32471         m_runState = NeedsAnotherRun;
32472     }
32473
32474     void TrackerBase::moveToParent() {
32475         assert( m_parent );
32476         m_ctx.setCurrentTracker( m_parent );
32477     }
32478     void TrackerBase::moveToThis() {
32479         m_ctx.setCurrentTracker( this );
32480     }
32481
32482     SectionTracker::SectionTracker( NameAndLocation const& nameAndLocation, TrackerContext& ctx,
ITracker* parent )
32483     :   TrackerBase( nameAndLocation, ctx, parent ),
32484         m_trimmed_name(trim(nameAndLocation.name))
32485     {
32486         if( parent ) {
32487             while( !parent->isSectionTracker() )
32488                 parent = &parent->parent();
32489
32490             SectionTracker& parentSection = static_cast<SectionTracker&>( *parent );
32491             addNextFilters( parentSection.m_filters );
32492         }
32493     }
32494
32495     bool SectionTracker::isComplete() const {
32496         bool complete = true;
32497
32498         if (m_filters.empty())
32499             || m_filters[0] == ""
32500             || std::find(m_filters.begin(), m_filters.end(), m_trimmed_name) != m_filters.end()) {
32501             complete = TrackerBase::isComplete();
32502         }
32503         return complete;
32504     }
32505
32506     bool SectionTracker::isSectionTracker() const { return true; }
32507
32508     SectionTracker& SectionTracker::acquire( TrackerContext& ctx, NameAndLocation const&
nameAndLocation ) {
32509         std::shared_ptr<SectionTracker> section;
32510
32511         ITracker& currentTracker = ctx.currentTracker();
32512         if( ITrackerPtr childTracker = currentTracker.findChild( nameAndLocation ) ) {
32513             assert( childTracker );
32514             assert( childTracker->isSectionTracker() );
32515             section = std::static_pointer_cast<SectionTracker>( childTracker );
32516         }
32517         else {
32518             section = std::make_shared<SectionTracker>( nameAndLocation, ctx, &currentTracker );
32519             currentTracker.addChild( section );
32520         }
32521         if( !ctx.completedCycle() )
32522             section->tryOpen();
32523         return *section;
32524     }
32525
32526     void SectionTracker::tryOpen() {
32527         if( !isComplete() )
32528             open();
32529     }
32530
32531     void SectionTracker::addInitialFilters( std::vector<std::string> const& filters ) {
32532         if( !filters.empty() ) {
32533             m_filters.reserve( m_filters.size() + filters.size() + 2 );
32534             m_filters.emplace_back(""); // Root - should never be consulted
32535             m_filters.emplace_back(""); // Test Case - not a section filter
32536             m_filters.insert( m_filters.end(), filters.begin(), filters.end() );
32537         }
32538     }
32539     void SectionTracker::addNextFilters( std::vector<std::string> const& filters ) {
32540         if( filters.size() > 1 )
32541             m_filters.insert( m_filters.end(), filters.begin()+1, filters.end() );
32542     }
32543

```

```

32544     std::vector<std::string> const& SectionTracker::getFilters() const {
32545         return m_filters;
32546     }
32547
32548     std::string const& SectionTracker::trimmedName() const {
32549         return m_trimmed_name;
32550     }
32551
32552 } // namespace TestCaseTracking
32553
32554 using TestCaseTracking::ITracker;
32555 using TestCaseTracking::TrackerContext;
32556 using TestCaseTracking::SectionTracker;
32557
32558 } // namespace Catch
32559
32560 #if defined(__clang__)
32561 #    pragma clang diagnostic pop
32562 #endif
32563 // end catch_test_case_tracker.cpp
32564 // start catch_test_registry.cpp
32565
32566 namespace Catch {
32567
32568     auto makeTestInvoker( void(*testAsFunction)() ) noexcept -> ITestInvoker* {
32569         return new(std::nothrow) TestInvokerAsFunction( testAsFunction );
32570     }
32571
32572     NameAndTags::NameAndTags( StringRef const& name_ , StringRef const& tags_ ) noexcept : name( name_
), tags( tags_ ) {}
32573
32574     AutoReg::AutoReg( ITestInvoker* invoker, SourceLineInfo const& lineInfo, StringRef const&
classOrMethod, NameAndTags const& nameAndTags ) noexcept {
32575         CATCH_TRY {
32576             getMutableRegistryHub()
32577                 .registerTest(
32578                     makeTestCase(
32579                         invoker,
32580                         extractClassName( classOrMethod ),
32581                         nameAndTags,
32582                         lineInfo));
32583         } CATCH_CATCH_ALL {
32584             // Do not throw when constructing global objects, instead register the exception to be
processed later
32585             getMutableRegistryHub().registerStartupException();
32586         }
32587     }
32588
32589     AutoReg::~AutoReg() = default;
32590 }
32591 // end catch_test_registry.cpp
32592 // start catch_test_spec.cpp
32593
32594 #include <algorithm>
32595 #include <string>
32596 #include <vector>
32597 #include <memory>
32598
32599 namespace Catch {
32600
32601     TestSpec::Pattern::Pattern( std::string const& name )
32602     : m_name( name )
32603     {}
32604
32605     TestSpec::Pattern::~Pattern() = default;
32606
32607     std::string const& TestSpec::Pattern::name() const {
32608         return m_name;
32609     }
32610
32611     TestSpec::NamePattern::NamePattern( std::string const& name, std::string const& filterString )
32612     : Pattern( filterString )
32613     , m_wildcardPattern( toLower( name ), CaseSensitive::No )
32614     {}
32615
32616     bool TestSpec::NamePattern::matches( TestCaseInfo const& testCase ) const {
32617         return m_wildcardPattern.matches( testCase.name );
32618     }
32619
32620     TestSpec::TagPattern::TagPattern( std::string const& tag, std::string const& filterString )
32621     : Pattern( filterString )
32622     , m_tag( toLower( tag ) )
32623     {}
32624
32625     bool TestSpec::TagPattern::matches( TestCaseInfo const& testCase ) const {
32626         return std::find(begin(testCase.lcaseTags),
32627             end(testCase.lcaseTags),

```

```

32628             m_tag) != end(testCase.lcaseTags);
32629     }
32630
32631     TestSpec::ExcludedPattern::ExcludedPattern( PatternPtr const& underlyingPattern )
32632     : Pattern( underlyingPattern->name() )
32633     , m_underlyingPattern( underlyingPattern )
32634     {}
32635
32636     bool TestSpec::ExcludedPattern::matches( TestCaseInfo const& testCase ) const {
32637         return !m_underlyingPattern->matches( testCase );
32638     }
32639
32640     bool TestSpec::Filter::matches( TestCaseInfo const& testCase ) const {
32641         return std::all_of( m_patterns.begin(), m_patterns.end(), [&]( PatternPtr const& p ){ return
p->matches( testCase ); } );
32642     }
32643
32644     std::string TestSpec::Filter::name() const {
32645         std::string name;
32646         for( auto const& p : m_patterns )
32647             name += p->name();
32648         return name;
32649     }
32650
32651     bool TestSpec::hasFilters() const {
32652         return !m_filters.empty();
32653     }
32654
32655     bool TestSpec::matches( TestCaseInfo const& testCase ) const {
32656         return std::any_of( m_filters.begin(), m_filters.end(), [&]( Filter const& f ){ return
f.matches( testCase ); } );
32657     }
32658
32659     TestSpec::Matches TestSpec::matchesByFilter( std::vector<TestCase> const& testCases, IConfig
const& config ) const
32660     {
32661         Matches matches( m_filters.size() );
32662         std::transform( m_filters.begin(), m_filters.end(), matches.begin(), [&]( Filter const& filter
){
32663             std::vector<TestCase const*> currentMatches;
32664             for( auto const& test : testCases )
32665                 if( isThrowSafe( test, config ) && filter.matches( test ) )
32666                     currentMatches.emplace_back( &test );
32667             return FilterMatch{ filter.name(), currentMatches };
32668         } );
32669         return matches;
32670     }
32671
32672     const TestSpec::vectorStrings& TestSpec::getInvalidArgs() const{
32673         return (m_invalidArgs);
32674     }
32675
32676 }
32677 // end catch_test_spec.cpp
32678 // start catch_test_spec_parser.cpp
32679
32680 namespace Catch {
32681
32682     TestSpecParser::TestSpecParser( ITagAliasRegistry const& tagAliases ) : m_tagAliases( &tagAliases
) {}
32683
32684     TestSpecParser& TestSpecParser::parse( std::string const& arg ) {
32685         m_mode = None;
32686         m_exclusion = false;
32687         m_arg = m_tagAliases->expandAliases( arg );
32688         m_escapeChars.clear();
32689         m_substring.reserve(m_arg.size());
32690         m_patternName.reserve(m_arg.size());
32691         m_realPatternPos = 0;
32692
32693         for( m_pos = 0; m_pos < m_arg.size(); ++m_pos )
32694             //if visitChar fails
32695             if( !visitChar( m_arg[m_pos] ) ){
32696                 m_testSpec.m_invalidArgs.push_back(arg);
32697                 break;
32698             }
32699         endMode();
32700         return *this;
32701     }
32702     TestSpec TestSpecParser::testSpec() {
32703         addFilter();
32704         return m_testSpec;
32705     }
32706     bool TestSpecParser::visitChar( char c ) {
32707         if( (m_mode != EscapedName) && (c == '\\') ) {
32708             escape();
32709             addCharToPattern(c);

```

```

32710         return true;
32711     }else if((m_mode != EscapedName) && (c == ',')) {
32712         return separate();
32713     }
32714
32715     switch( m_mode ) {
32716     case None:
32717         if( processNoneChar( c ) )
32718             return true;
32719         break;
32720     case Name:
32721         processNameChar( c );
32722         break;
32723     case EscapedName:
32724         endMode();
32725         addCharToPattern(c);
32726         return true;
32727     default:
32728     case Tag:
32729     case QuotedName:
32730         if( processOtherChar( c ) )
32731             return true;
32732         break;
32733     }
32734
32735     m_substring += c;
32736     if( !isControlChar( c ) ) {
32737         m_patternName += c;
32738         m_realPatternPos++;
32739     }
32740     return true;
32741 }
32742 // Two of the processing methods return true to signal the caller to return
32743 // without adding the given character to the current pattern strings
32744 bool TestSpecParser::processNoneChar( char c ) {
32745     switch( c ) {
32746     case ' ':
32747         return true;
32748     case '~':
32749         m_exclusion = true;
32750         return false;
32751     case '[':
32752         startNewMode( Tag );
32753         return false;
32754     case '"':
32755         startNewMode( QuotedName );
32756         return false;
32757     default:
32758         startNewMode( Name );
32759         return false;
32760     }
32761 }
32762 void TestSpecParser::processNameChar( char c ) {
32763     if( c == '[' ) {
32764         if( m_substring == "exclude:" )
32765             m_exclusion = true;
32766         else
32767             endMode();
32768         startNewMode( Tag );
32769     }
32770 }
32771 bool TestSpecParser::processOtherChar( char c ) {
32772     if( !isControlChar( c ) )
32773         return false;
32774     m_substring += c;
32775     endMode();
32776     return true;
32777 }
32778 void TestSpecParser::startNewMode( Mode mode ) {
32779     m_mode = mode;
32780 }
32781 void TestSpecParser::endMode() {
32782     switch( m_mode ) {
32783     case Name:
32784     case QuotedName:
32785         return addNamePattern();
32786     case Tag:
32787         return addTagPattern();
32788     case EscapedName:
32789         revertBackToLastMode();
32790         return;
32791     case None:
32792     default:
32793         return startNewMode( None );
32794     }
32795 }
32796 void TestSpecParser::escape() {

```

```

32797         saveLastMode();
32798         m_mode = EscapedName;
32799         m_escapeChars.push_back(m_realPatternPos);
32800     }
32801     bool TestSpecParser::isControlChar( char c ) const {
32802         switch( m_mode ) {
32803             default:
32804                 return false;
32805             case None:
32806                 return c == '~';
32807             case Name:
32808                 return c == '[';
32809             case EscapedName:
32810                 return true;
32811             case QuotedName:
32812                 return c == '"';
32813             case Tag:
32814                 return c == '[' || c == '[';
32815         }
32816     }
32817
32818     void TestSpecParser::addFilter() {
32819         if( !m_currentFilter.m_patterns.empty() ) {
32820             m_testSpec.m_filters.push_back( m_currentFilter );
32821             m_currentFilter = TestSpec::Filter();
32822         }
32823     }
32824
32825     void TestSpecParser::saveLastMode() {
32826         lastMode = m_mode;
32827     }
32828
32829     void TestSpecParser::revertBackToLastMode() {
32830         m_mode = lastMode;
32831     }
32832
32833     bool TestSpecParser::separate() {
32834         if( (m_mode==QuotedName) || (m_mode==Tag) ){
32835             //invalid argument, signal failure to previous scope.
32836             m_mode = None;
32837             m_pos = m_arg.size();
32838             m_substring.clear();
32839             m_patternName.clear();
32840             m_realPatternPos = 0;
32841             return false;
32842         }
32843         endMode();
32844         addFilter();
32845         return true; //success
32846     }
32847
32848     std::string TestSpecParser::preprocessPattern() {
32849         std::string token = m_patternName;
32850         for (std::size_t i = 0; i < m_escapeChars.size(); ++i)
32851             token = token.substr(0, m_escapeChars[i] - i) + token.substr(m_escapeChars[i] - i + 1);
32852         m_escapeChars.clear();
32853         if (startsWith(token, "exclude:")) {
32854             m_exclusion = true;
32855             token = token.substr(8);
32856         }
32857
32858         m_patternName.clear();
32859         m_realPatternPos = 0;
32860
32861         return token;
32862     }
32863
32864     void TestSpecParser::addNamePattern() {
32865         auto token = preprocessPattern();
32866
32867         if (!token.empty()) {
32868             TestSpec::PatternPtr pattern = std::make_shared<TestSpec::NamePattern>(token,
32869 m_substring);
32870             if (m_exclusion)
32871                 pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
32872             m_currentFilter.m_patterns.push_back(pattern);
32873         }
32874         m_substring.clear();
32875         m_exclusion = false;
32876         m_mode = None;
32877     }
32878
32879     void TestSpecParser::addTagPattern() {
32880         auto token = preprocessPattern();
32881
32882         if (!token.empty()) {
32883             // If the tag pattern is the "hide and tag" shorthand (e.g. [.foo])

```

```

32883         // we have to create a separate hide tag and shorten the real one
32884         if (token.size() > 1 && token[0] == '.') {
32885             token.erase(token.begin());
32886             TestSpec::PatternPtr pattern = std::make_shared<TestSpec::TagPattern>(".",
m_substring);
32887             if (m_exclusion) {
32888                 pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
32889             }
32890             m_currentFilter.m_patterns.push_back(pattern);
32891         }
32892
32893         TestSpec::PatternPtr pattern = std::make_shared<TestSpec::TagPattern>(token, m_substring);
32894
32895         if (m_exclusion) {
32896             pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
32897         }
32898         m_currentFilter.m_patterns.push_back(pattern);
32899     }
32900     m_substring.clear();
32901     m_exclusion = false;
32902     m_mode = None;
32903 }
32904
32905 TestSpec parseTestSpec( std::string const& arg ) {
32906     return TestSpecParser( ITagAliasRegistry::get() ).parse( arg ).testSpec();
32907 }
32908
32909 } // namespace Catch
32910 // end catch_test_spec_parser.cpp
32911 // start catch_timer.cpp
32912
32913 #include <chrono>
32914
32915 static const uint64_t nanosecondsInSecond = 1000000000;
32916
32917 namespace Catch {
32918
32919     auto getCurrentNanosecondsSinceEpoch() -> uint64_t {
32920         return std::chrono::duration_cast<std::chrono::nanoseconds>(
std::chrono::high_resolution_clock::now().time_since_epoch() ).count();
32921     }
32922
32923     namespace {
32924         auto estimateClockResolution() -> uint64_t {
32925             uint64_t sum = 0;
32926             static const uint64_t iterations = 1000000;
32927
32928             auto startTime = getCurrentNanosecondsSinceEpoch();
32929
32930             for( std::size_t i = 0; i < iterations; ++i ) {
32931
32932                 uint64_t ticks;
32933                 uint64_t baseTicks = getCurrentNanosecondsSinceEpoch();
32934                 do {
32935                     ticks = getCurrentNanosecondsSinceEpoch();
32936                 } while( ticks == baseTicks );
32937
32938                 auto delta = ticks - baseTicks;
32939                 sum += delta;
32940
32941                 // If we have been calibrating for over 3 seconds -- the clock
32942                 // is terrible and we should move on.
32943                 // TBD: How to signal that the measured resolution is probably wrong?
32944                 if (ticks > startTime + 3 * nanosecondsInSecond) {
32945                     return sum / ( i + 1u );
32946                 }
32947             }
32948
32949             // We're just taking the mean, here. To do better we could take the std. dev and exclude
outliers
32950             // - and potentially do more iterations if there's a high variance.
32951             return sum/iterations;
32952         }
32953     }
32954     auto getEstimatedClockResolution() -> uint64_t {
32955         static auto s_resolution = estimateClockResolution();
32956         return s_resolution;
32957     }
32958
32959     void Timer::start() {
32960         m_nanoseconds = getCurrentNanosecondsSinceEpoch();
32961     }
32962     auto Timer::getElapsedNanoseconds() const -> uint64_t {
32963         return getCurrentNanosecondsSinceEpoch() - m_nanoseconds;
32964     }
32965     auto Timer::getElapsedMicroseconds() const -> uint64_t {
32966         return getElapsedNanoseconds()/1000;

```

```

32967     }
32968     auto Timer::getElapsedMilliseconds() const -> unsigned int {
32969         return static_cast<unsigned int>(getElapsedMicroseconds()/1000);
32970     }
32971     auto Timer::getElapsedSeconds() const -> double {
32972         return getElapsedMicroseconds()/1000000.0;
32973     }
32974
32975 } // namespace Catch
32976 // end catch_timer.cpp
32977 // start catch_tostring.cpp
32978
32979 #if defined(__clang__)
32980 #    pragma clang diagnostic push
32981 #    pragma clang diagnostic ignored "-Wexit-time-destructors"
32982 #    pragma clang diagnostic ignored "-Wglobal-constructors"
32983 #endif
32984
32985 // Enable specific decls locally
32986 #if !defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
32987 #define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
32988 #endif
32989
32990 #include <cmath>
32991 #include <iomanip>
32992
32993 namespace Catch {
32994
32995     namespace Detail {
32996
32997         const std::string printableString = "{?}";
32998
32999         namespace {
33000             const int hexThreshold = 255;
33001
33002             struct Endianness {
33003                 enum Arch { Big, Little };
33004
33005                 static Arch which() {
33006                     int one = 1;
33007                     // If the lowest byte we read is non-zero, we can assume
33008                     // that little endian format is used.
33009                     auto value = *reinterpret_cast<char*>(&one);
33010                     return value ? Little : Big;
33011                 }
33012             };
33013         }
33014
33015         std::string rawMemoryToString( const void *object, std::size_t size ) {
33016             // Reverse order for little endian architectures
33017             int i = 0, end = static_cast<int>( size ), inc = 1;
33018             if( Endianness::which() == Endianness::Little ) {
33019                 i = end-1;
33020                 end = inc = -1;
33021             }
33022
33023             unsigned char const *bytes = static_cast<unsigned char const *>(object);
33024             ReusableStringStream rss;
33025             rss << "0x" << std::setfill('0') << std::hex;
33026             for( ; i != end; i += inc )
33027                 rss << std::setw(2) << static_cast<unsigned>(bytes[i]);
33028             return rss.str();
33029         }
33030     }
33031
33032     template<typename T>
33033     std::string fpToString( T value, int precision ) {
33034         if (Catch::isnan(value)) {
33035             return "nan";
33036         }
33037
33038         ReusableStringStream rss;
33039         rss << std::setprecision( precision )
33040             << std::fixed
33041             << value;
33042         std::string d = rss.str();
33043         std::size_t i = d.find_last_not_of( '0' );
33044         if( i != std::string::npos && i != d.size()-1 ) {
33045             if( d[i] == '.' )
33046                 i++;
33047             d = d.substr( 0, i+1 );
33048         }
33049         return d;
33050     }
33051
33052 //
33053 // Out-of-line defs for full specialization of StringMaker

```



```

33055 //
33056
33057 std::string StringMaker<std::string>::convert(const std::string& str) {
33058     if (!getCurrentContext().getConfig()->showInvisibles()) {
33059         return "'" + str + "'";
33060     }
33061
33062     std::string s("\\");
33063     for (char c : str) {
33064         switch (c) {
33065             case '\\':
33066                 s.append("\\\\");
33067                 break;
33068             case '\n':
33069                 s.append("\\n");
33070                 break;
33071             case '\t':
33072                 s.append("\\t");
33073                 break;
33074             default:
33075                 s.push_back(c);
33076                 break;
33077         }
33078     }
33079     s.append("\\");
33080     return s;
33081 }
33082 #ifdef CATCH_CONFIG_CPP17_STRING_VIEW
33083 std::string StringMaker<std::string_view>::convert(std::string_view str) {
33084     return ::Catch::Detail::stringify(std::string{ str });
33085 }
33086 #endif
33087 std::string StringMaker<char const*>::convert(char const* str) {
33088     if (str) {
33089         return ::Catch::Detail::stringify(std::string{ str });
33090     } else {
33091         return "{null string}";
33092     }
33093 }
33094 std::string StringMaker<char*>::convert(char* str) {
33095     if (str) {
33096         return ::Catch::Detail::stringify(std::string{ str });
33097     } else {
33098         return "{null string}";
33099     }
33100 }
33101
33102 #ifdef CATCH_CONFIG_WCHAR
33103 std::string StringMaker<std::wstring>::convert(const std::wstring& wstr) {
33104     std::string s;
33105     s.reserve(wstr.size());
33106     for (auto c : wstr) {
33107         s += (c <= 0xff) ? static_cast<char>(c) : '?';
33108     }
33109     return ::Catch::Detail::stringify(s);
33110 }
33111
33112 #ifdef CATCH_CONFIG_CPP17_STRING_VIEW
33113 std::string StringMaker<std::wstring_view>::convert(std::wstring_view str) {
33114     return StringMaker<std::wstring>::convert(std::wstring(str));
33115 }
33116 #endif
33117
33118 std::string StringMaker<wchar_t const*>::convert(wchar_t const * str) {
33119     if (str) {
33120         return ::Catch::Detail::stringify(std::wstring{ str });
33121     } else {
33122         return "{null string}";
33123     }
33124 }
33125 std::string StringMaker<wchar_t *>::convert(wchar_t * str) {
33126     if (str) {
33127         return ::Catch::Detail::stringify(std::wstring{ str });
33128     } else {
33129         return "{null string}";
33130     }
33131 }
33132 #endif
33133
33134 #if defined(CATCH_CONFIG_CPP17_BYTE)
33135 #include <cstdint>
33136 std::string StringMaker<std::byte>::convert(std::byte value) {
33137     return ::Catch::Detail::stringify(std::to_integer<unsigned long long>(value));
33138 }
33139 #endif // defined(CATCH_CONFIG_CPP17_BYTE)
33140
33141 std::string StringMaker<int>::convert(int value) {
33142     return ::Catch::Detail::stringify(static_cast<long long>(value));

```

```

33143 }
33144 std::string StringMaker<long>::convert(long value) {
33145     return ::Catch::Detail::stringify(static_cast<long long>(value));
33146 }
33147 std::string StringMaker<long long>::convert(long long value) {
33148     ReusableStringStream rss;
33149     rss << value;
33150     if (value > Detail::hexThreshold) {
33151         rss << " (0x" << std::hex << value << ')';
33152     }
33153     return rss.str();
33154 }
33155
33156 std::string StringMaker<unsigned int>::convert(unsigned int value) {
33157     return ::Catch::Detail::stringify(static_cast<unsigned long long>(value));
33158 }
33159 std::string StringMaker<unsigned long>::convert(unsigned long value) {
33160     return ::Catch::Detail::stringify(static_cast<unsigned long long>(value));
33161 }
33162 std::string StringMaker<unsigned long long>::convert(unsigned long long value) {
33163     ReusableStringStream rss;
33164     rss << value;
33165     if (value > Detail::hexThreshold) {
33166         rss << " (0x" << std::hex << value << ')';
33167     }
33168     return rss.str();
33169 }
33170
33171 std::string StringMaker<bool>::convert(bool b) {
33172     return b ? "true" : "false";
33173 }
33174
33175 std::string StringMaker<signed char>::convert(signed char value) {
33176     if (value == '\r') {
33177         return "\\r";
33178     } else if (value == '\f') {
33179         return "\\f";
33180     } else if (value == '\n') {
33181         return "\\n";
33182     } else if (value == '\t') {
33183         return "\\t";
33184     } else if ('\0' <= value && value < ' ') {
33185         return ::Catch::Detail::stringify(static_cast<unsigned int>(value));
33186     } else {
33187         char chstr[] = " ' ";
33188         chstr[1] = value;
33189         return chstr;
33190     }
33191 }
33192 std::string StringMaker<char>::convert(char c) {
33193     return ::Catch::Detail::stringify(static_cast<signed char>(c));
33194 }
33195 std::string StringMaker<unsigned char>::convert(unsigned char c) {
33196     return ::Catch::Detail::stringify(static_cast<char>(c));
33197 }
33198
33199 std::string StringMaker<std::nullptr_t>::convert(std::nullptr_t) {
33200     return "nullptr";
33201 }
33202
33203 int StringMaker<float>::precision = 5;
33204
33205 std::string StringMaker<float>::convert(float value) {
33206     return fpToString(value, precision) + 'f';
33207 }
33208
33209 int StringMaker<double>::precision = 10;
33210
33211 std::string StringMaker<double>::convert(double value) {
33212     return fpToString(value, precision);
33213 }
33214
33215 std::string ratio_string<std::atto>::symbol() { return "a"; }
33216 std::string ratio_string<std::femto>::symbol() { return "f"; }
33217 std::string ratio_string<std::pico>::symbol() { return "p"; }
33218 std::string ratio_string<std::nano>::symbol() { return "n"; }
33219 std::string ratio_string<std::micro>::symbol() { return "u"; }
33220 std::string ratio_string<std::milli>::symbol() { return "m"; }
33221
33222 } // end namespace Catch
33223
33224 #if defined(__clang__)
33225 #    pragma clang diagnostic pop
33226 #endif
33227
33228 // end catch_tostring.cpp
33229 // start catch_totals.cpp

```

```

33230
33231 namespace Catch {
33232
33233     Counts Counts::operator - ( Counts const& other ) const {
33234         Counts diff;
33235         diff.passed = passed - other.passed;
33236         diff.failed = failed - other.failed;
33237         diff.failedButOk = failedButOk - other.failedButOk;
33238         return diff;
33239     }
33240
33241     Counts& Counts::operator += ( Counts const& other ) {
33242         passed += other.passed;
33243         failed += other.failed;
33244         failedButOk += other.failedButOk;
33245         return *this;
33246     }
33247
33248     std::size_t Counts::total() const {
33249         return passed + failed + failedButOk;
33250     }
33251     bool Counts::allPassed() const {
33252         return failed == 0 && failedButOk == 0;
33253     }
33254     bool Counts::allOk() const {
33255         return failed == 0;
33256     }
33257
33258     Totals Totals::operator - ( Totals const& other ) const {
33259         Totals diff;
33260         diff.assertions = assertions - other.assertions;
33261         diff.testCases = testCases - other.testCases;
33262         return diff;
33263     }
33264
33265     Totals& Totals::operator += ( Totals const& other ) {
33266         assertions += other.assertions;
33267         testCases += other.testCases;
33268         return *this;
33269     }
33270
33271     Totals Totals::delta( Totals const& prevTotals ) const {
33272         Totals diff = *this - prevTotals;
33273         if( diff.assertions.failed > 0 )
33274             ++diff.testCases.failed;
33275         else if( diff.assertions.failedButOk > 0 )
33276             ++diff.testCases.failedButOk;
33277         else
33278             ++diff.testCases.passed;
33279         return diff;
33280     }
33281 }
33282 // end catch_totals.cpp
33283 // start catch_uncaught_exceptions.cpp
33284 // start catch_config_uncaught_exceptions.hpp
33285
33286 // Copyright Catch2 Authors
33287 // Distributed under the Boost Software License, Version 1.0.
33288 // (See accompanying file LICENSE_1_0.txt or copy at
33289 // https://www.boost.org/LICENSE_1_0.txt)
33290
33291 // SPDX-License-Identifier: BSL-1.0
33292
33293 #ifndef CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
33294 #define CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
33295
33296 #if defined(_MSC_VER)
33297     #if _MSC_VER >= 1900 // Visual Studio 2015 or newer
33298         #define CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
33299     #endif
33300 #endif
33301 #endif
33302
33303 #include <exception>
33304
33305 #if defined(__cpp_lib_uncaught_exceptions) \
33306     && !defined(CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)
33307     #define CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
33308 #endif
33309 #endif // __cpp_lib_uncaught_exceptions
33310
33311 #if defined(CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS) \
33312     && !defined(CATCH_CONFIG_NO_CPP17_UNCAUGHT_EXCEPTIONS) \
33313     && !defined(CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)
33314     #define CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
33315 #endif
33316

```

```

33317 #endif
33318
33319 #endif // CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
33320 // end catch_config_uncaught_exceptions.hpp
33321 #include <exception>
33322
33323 namespace Catch {
33324     bool uncaught_exceptions() {
33325         #if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
33326             return false;
33327         #elif defined(CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)
33328             return std::uncaught_exceptions() > 0;
33329         #else
33330             return std::uncaught_exception();
33331         #endif
33332     }
33333 } // end namespace Catch
33334 // end catch_uncaught_exceptions.cpp
33335 // start catch_version.cpp
33336
33337 #include <ostream>
33338
33339 namespace Catch {
33340
33341     Version::Version
33342     (   unsigned int _majorVersion,
33343         unsigned int _minorVersion,
33344         unsigned int _patchNumber,
33345         char const* _branchName,
33346         unsigned int _buildNumber )
33347     :   majorVersion( _majorVersion ),
33348         minorVersion( _minorVersion ),
33349         patchNumber( _patchNumber ),
33350         branchName( _branchName ),
33351         buildNumber( _buildNumber )
33352     {}
33353
33354     std::ostream& operator << ( std::ostream& os, Version const& version ) {
33355         os << version.majorVersion << '.'
33356            << version.minorVersion << '.'
33357            << version.patchNumber;
33358         // branchName is never null -> 0th char is \0 if it is empty
33359         if (version.branchName[0]) {
33360             os << '-' << version.branchName
33361                << '.' << version.buildNumber;
33362         }
33363         return os;
33364     }
33365
33366     Version const& libraryVersion() {
33367         static Version version( 2, 13, 10, "", 0 );
33368         return version;
33369     }
33370
33371 }
33372 // end catch_version.cpp
33373 // start catch_wildcard_pattern.cpp
33374
33375 namespace Catch {
33376
33377     WildcardPattern::WildcardPattern( std::string const& pattern,
33378                                       CaseSensitive::Choice caseSensitivity )
33379     :   m_caseSensitivity( caseSensitivity ),
33380         m_pattern( normaliseString( pattern ) )
33381     {
33382         if( startsWith( m_pattern, '*' ) ) {
33383             m_pattern = m_pattern.substr( 1 );
33384             m_wildcard = WildcardAtStart;
33385         }
33386         if( endsWith( m_pattern, '*' ) ) {
33387             m_pattern = m_pattern.substr( 0, m_pattern.size()-1 );
33388             m_wildcard = static_cast<WildcardPosition>( m_wildcard | WildcardAtEnd );
33389         }
33390     }
33391
33392     bool WildcardPattern::matches( std::string const& str ) const {
33393         switch( m_wildcard ) {
33394             case NoWildcard:
33395                 return m_pattern == normaliseString( str );
33396             case WildcardAtStart:
33397                 return endsWith( normaliseString( str ), m_pattern );
33398             case WildcardAtEnd:
33399                 return startsWith( normaliseString( str ), m_pattern );
33400             case WildcardAtBothEnds:
33401                 return contains( normaliseString( str ), m_pattern );
33402             default:
33403                 CATCH_INTERNAL_ERROR( "Unknown enum" );

```

```

33404     }
33405 }
33406
33407 std::string WildcardPattern::normaliseString( std::string const& str ) const {
33408     return trim( m_caseSensitivity == CaseSensitive::No ? toLower( str ) : str );
33409 }
33410 }
33411 // end catch_wildcard_pattern.cpp
33412 // start catch_xmlwriter.cpp
33413
33414 #include <iomanip>
33415 #include <type_traits>
33416
33417 namespace Catch {
33418
33419 namespace {
33420
33421     size_t trailingBytes(unsigned char c) {
33422         if ((c & 0xE0) == 0xC0) {
33423             return 2;
33424         }
33425         if ((c & 0xF0) == 0xE0) {
33426             return 3;
33427         }
33428         if ((c & 0xF8) == 0xF0) {
33429             return 4;
33430         }
33431         CATCH_INTERNAL_ERROR("Invalid multibyte utf-8 start byte encountered");
33432     }
33433
33434     uint32_t headerValue(unsigned char c) {
33435         if ((c & 0xE0) == 0xC0) {
33436             return c & 0x1F;
33437         }
33438         if ((c & 0xF0) == 0xE0) {
33439             return c & 0x0F;
33440         }
33441         if ((c & 0xF8) == 0xF0) {
33442             return c & 0x07;
33443         }
33444         CATCH_INTERNAL_ERROR("Invalid multibyte utf-8 start byte encountered");
33445     }
33446
33447     void hexEscapeChar(std::ostream& os, unsigned char c) {
33448         std::ios_base::fmtflags f(os.flags());
33449         os << "\\x"
33450             << std::uppercase << std::hex << std::setfill('0') << std::setw(2)
33451             << static_cast<int>(c);
33452         os.flags(f);
33453     }
33454
33455     bool shouldNewline(XmlFormatting fmt) {
33456         return !(static_cast<std::underlying_type<XmlFormatting>::type>(fmt &
33457             XmlFormatting::Newline));
33458     }
33459
33460     bool shouldIndent(XmlFormatting fmt) {
33461         return !(static_cast<std::underlying_type<XmlFormatting>::type>(fmt &
33462             XmlFormatting::Indent));
33463     }
33464 } // anonymous namespace
33465
33466 XmlFormatting operator | (XmlFormatting lhs, XmlFormatting rhs) {
33467     return static_cast<XmlFormatting>(
33468         static_cast<std::underlying_type<XmlFormatting>::type>(lhs) |
33469         static_cast<std::underlying_type<XmlFormatting>::type>(rhs)
33470     );
33471 }
33472
33473 XmlFormatting operator & (XmlFormatting lhs, XmlFormatting rhs) {
33474     return static_cast<XmlFormatting>(
33475         static_cast<std::underlying_type<XmlFormatting>::type>(lhs) &
33476         static_cast<std::underlying_type<XmlFormatting>::type>(rhs)
33477     );
33478 }
33479
33480 XmlEncode::XmlEncode( std::string const& str, ForWhat forWhat )
33481 :   m_str( str ),
33482   m_forWhat( forWhat )
33483 {}
33484
33485 void XmlEncode::encodeTo( std::ostream& os ) const {
33486     // Apostrophe escaping not necessary if we always use " to write attributes
33487     // (see: http://www.w3.org/TR/xml/#syntax)
33488     for( std::size_t idx = 0; idx < m_str.size(); ++ idx ) {

```

```

33489         unsigned char c = m_str[idx];
33490         switch (c) {
33491             case '<': os << "<"; break;
33492             case '&': os << "&"; break;
33493
33494             case '>':
33495                 // See: http://www.w3.org/TR/xml/#syntax
33496                 if (idx > 2 && m_str[idx - 1] == ']' && m_str[idx - 2] == '[')
33497                     os << ">";
33498                 else
33499                     os << c;
33500                 break;
33501
33502             case '\"':
33503                 if (m_forWhat == ForAttributes)
33504                     os << """;
33505                 else
33506                     os << c;
33507                 break;
33508
33509             default:
33510                 // Check for control characters and invalid utf-8
33511
33512                 // Escape control characters in standard ascii
33513                 // see
33514                 http://stackoverflow.com/questions/404107/why-are-control-characters-illegal-in-xml-1-0
33515                 if (c < 0x09 || (c > 0x0D && c < 0x20) || c == 0x7F) {
33516                     hexEscapeChar(os, c);
33517                     break;
33518
33519                     // Plain ASCII: Write it to stream
33520                 if (c < 0x7F) {
33521                     os << c;
33522                     break;
33523                 }
33524
33525                 // UTF-8 territory
33526                 // Check if the encoding is valid and if it is not, hex escape bytes.
33527                 // Important: We do not check the exact decoded values for validity, only the encoding
33528                 format
33529                 // First check that this bytes is a valid lead byte:
33530                 // This means that it is not encoded as 1111 1XXX
33531                 // Or as 10XX XXXX
33532                 if (c < 0xC0 ||
33533                     c >= 0xF8) {
33534                     hexEscapeChar(os, c);
33535                     break;
33536
33537                     auto encBytes = trailingBytes(c);
33538                     // Are there enough bytes left to avoid accessing out-of-bounds memory?
33539                     if (idx + encBytes - 1 >= m_str.size()) {
33540                         hexEscapeChar(os, c);
33541                         break;
33542                     }
33543                     // The header is valid, check data
33544                     // The next encBytes bytes must together be a valid utf-8
33545                     // This means: bitpattern 10XX XXXX and the extracted value is sane (ish)
33546                     bool valid = true;
33547                     uint32_t value = headerValue(c);
33548                     for (std::size_t n = 1; n < encBytes; ++n) {
33549                         unsigned char nc = m_str[idx + n];
33550                         valid &= ((nc & 0xC0) == 0x80);
33551                         value = (value << 6) | (nc & 0x3F);
33552                     }
33553
33554                     if (
33555                         // Wrong bit pattern of following bytes
33556                         (!valid) ||
33557                         // Overlong encodings
33558                         (value < 0x80) ||
33559                         (0x80 <= value && value < 0x800 && encBytes > 2) ||
33560                         (0x800 < value && value < 0x10000 && encBytes > 3) ||
33561                         // Encoded value out of range
33562                         (value >= 0x110000)
33563                     ) {
33564                         hexEscapeChar(os, c);
33565                         break;
33566                     }
33567
33568                     // If we got here, this is in fact a valid(ish) utf-8 sequence
33569                     for (std::size_t n = 0; n < encBytes; ++n) {
33570                         os << m_str[idx + n];
33571                     }
33572                     idx += encBytes - 1;
33573                     break;

```

```

33574         }
33575     }
33576 }
33577
33578 std::ostream& operator<< ( std::ostream& os, XmlEncode const& xmlEncode ) {
33579     xmlEncode.encodeTo( os );
33580     return os;
33581 }
33582
33583 XmlWriter::ScopedElement::ScopedElement( XmlWriter* writer, XmlFormatting fmt )
33584 :   m_writer( writer ),
33585   m_fmt( fmt )
33586 {}
33587
33588 XmlWriter::ScopedElement::ScopedElement( ScopedElement&& other ) noexcept
33589 :   m_writer( other.m_writer ),
33590   m_fmt( other.m_fmt )
33591 {
33592     other.m_writer = nullptr;
33593     other.m_fmt = XmlFormatting::None;
33594 }
33595 XmlWriter::ScopedElement& XmlWriter::ScopedElement::operator=( ScopedElement&& other ) noexcept {
33596     if ( m_writer ) {
33597         m_writer->endElement();
33598     }
33599     m_writer = other.m_writer;
33600     other.m_writer = nullptr;
33601     m_fmt = other.m_fmt;
33602     other.m_fmt = XmlFormatting::None;
33603     return *this;
33604 }
33605
33606 XmlWriter::ScopedElement::~ScopedElement() {
33607     if ( m_writer ) {
33608         m_writer->endElement( m_fmt );
33609     }
33610 }
33611
33612 XmlWriter::ScopedElement& XmlWriter::ScopedElement::writeText( std::string const& text,
33613 XmlFormatting fmt ) {
33614     m_writer->writeText( text, fmt );
33615     return *this;
33616 }
33617
33618 XmlWriter::XmlWriter( std::ostream& os ) : m_os( os )
33619 {
33620     writeDeclaration();
33621 }
33622
33623 XmlWriter::~XmlWriter() {
33624     while ( !m_tags.empty() ) {
33625         endElement();
33626     }
33627     newlineIfNecessary();
33628 }
33629
33630 XmlWriter& XmlWriter::startElement( std::string const& name, XmlFormatting fmt ) {
33631     ensureTagClosed();
33632     newlineIfNecessary();
33633     if ( shouldIndent( fmt ) ) {
33634         m_os<< m_indent;
33635         m_indent += " ";
33636     }
33637     m_os<< '<'<< name;
33638     m_tags.push_back( name );
33639     m_tagIsOpen = true;
33640     applyFormatting( fmt );
33641     return *this;
33642 }
33643
33644 XmlWriter::ScopedElement XmlWriter::scopedElement( std::string const& name, XmlFormatting fmt ) {
33645     ScopedElement scoped( this, fmt );
33646     startElement( name, fmt );
33647     return scoped;
33648 }
33649
33650 XmlWriter& XmlWriter::endElement( XmlFormatting fmt ) {
33651     m_indent = m_indent.substr( 0, m_indent.size() - 2 );
33652     if ( m_tagIsOpen ) {
33653         m_os<< ">";
33654         m_tagIsOpen = false;
33655     } else {
33656         newlineIfNecessary();
33657         if ( shouldIndent( fmt ) ) {
33658             m_os<< m_indent;
33659         }

```

```

33660         m_os << "</" << m_tags.back() << ">";
33661     }
33662     m_os << std::flush;
33663     applyFormatting(fmt);
33664     m_tags.pop_back();
33665     return *this;
33666 }
33667
33668 XmlWriter& XmlWriter::writeAttribute( std::string const& name, std::string const& attribute ) {
33669     if( !name.empty() && !attribute.empty() )
33670         m_os << ' ' << name << "=" << XmlEncode( attribute, XmlEncode::ForAttributes ) << "'";
33671     return *this;
33672 }
33673
33674 XmlWriter& XmlWriter::writeAttribute( std::string const& name, bool attribute ) {
33675     m_os << ' ' << name << "=" << ( attribute ? "true" : "false" ) << "'";
33676     return *this;
33677 }
33678
33679 XmlWriter& XmlWriter::writeText( std::string const& text, XmlFormatting fmt ) {
33680     if( !text.empty() ){
33681         bool tagWasOpen = m_tagIsOpen;
33682         ensureTagClosed();
33683         if (tagWasOpen && shouldIndent(fmt)) {
33684             m_os << m_indent;
33685         }
33686         m_os << XmlEncode( text );
33687         applyFormatting(fmt);
33688     }
33689     return *this;
33690 }
33691
33692 XmlWriter& XmlWriter::writeComment( std::string const& text, XmlFormatting fmt ) {
33693     ensureTagClosed();
33694     if (shouldIndent(fmt)) {
33695         m_os << m_indent;
33696     }
33697     m_os << "<!--" << text << "-->";
33698     applyFormatting(fmt);
33699     return *this;
33700 }
33701
33702 void XmlWriter::writeStylesheetRef( std::string const& url ) {
33703     m_os << "<?xml-stylesheet type='text/xsl' href='" << url << "'?>\n";
33704 }
33705
33706 XmlWriter& XmlWriter::writeBlankLine() {
33707     ensureTagClosed();
33708     m_os << '\n';
33709     return *this;
33710 }
33711
33712 void XmlWriter::ensureTagClosed() {
33713     if( m_tagIsOpen ) {
33714         m_os << '>' << std::flush;
33715         newlineIfNecessary();
33716         m_tagIsOpen = false;
33717     }
33718 }
33719
33720 void XmlWriter::applyFormatting(XmlFormatting fmt) {
33721     m_needsNewline = shouldNewline(fmt);
33722 }
33723
33724 void XmlWriter::writeDeclaration() {
33725     m_os << "<?xml version='1.0' encoding='UTF-8'?>\n";
33726 }
33727
33728 void XmlWriter::newlineIfNecessary() {
33729     if( m_needsNewline ) {
33730         m_os << std::endl;
33731         m_needsNewline = false;
33732     }
33733 }
33734 }
33735 // end catch_xmlwriter.cpp
33736 // start catch_reporter_bases.cpp
33737
33738 #include <cstring>
33739 #include <cfloat>
33740 #include <cstdio>
33741 #include <cassert>
33742 #include <memory>
33743
33744 namespace Catch {
33745     void prepareExpandedExpression( AssertionResult& result ) {
33746         result.getExpandedExpression();

```



```

33747     }
33748
33749     // Because formatting using c++ streams is stateful, drop down to C is required
33750     // Alternatively we could use stringstream, but its performance is... not good.
33751     std::string getFormattedDuration( double duration ) {
33752         // Max exponent + 1 is required to represent the whole part
33753         // + 1 for decimal point
33754         // + 3 for the 3 decimal places
33755         // + 1 for null terminator
33756         const std::size_t maxDoubleSize = DBL_MAX_10_EXP + 1 + 1 + 3 + 1;
33757         char buffer[maxDoubleSize];
33758
33759         // Save previous errno, to prevent sprintf from overwriting it
33760         ErrnoGuard guard;
33761 #ifdef _MSC_VER
33762         sprintf_s(buffer, "%.3f", duration);
33763 #else
33764         std::sprintf(buffer, "%.3f", duration);
33765 #endif
33766         return std::string(buffer);
33767     }
33768
33769     bool shouldShowDuration( IConfig const& config, double duration ) {
33770         if ( config.showDurations() == ShowDurations::Always ) {
33771             return true;
33772         }
33773         if ( config.showDurations() == ShowDurations::Never ) {
33774             return false;
33775         }
33776         const double min = config.minDuration();
33777         return min >= 0 && duration >= min;
33778     }
33779
33780     std::string serializeFilters( std::vector<std::string> const& container ) {
33781         ReusableStringStream oss;
33782         bool first = true;
33783         for (auto&& filter : container)
33784         {
33785             if (!first)
33786                 oss << ' ';
33787             else
33788                 first = false;
33789
33790             oss << filter;
33791         }
33792         return oss.str();
33793     }
33794
33795     TestEventListenerBase::TestEventListenerBase(ReporterConfig const & _config)
33796         :StreamingReporterBase(_config) {}
33797
33798     std::set<Verbosity> TestEventListenerBase::getSupportedVerbsities() {
33799         return { Verbosity::Quiet, Verbosity::Normal, Verbosity::High };
33800     }
33801
33802     void TestEventListenerBase::assertionStarting(AssertionInfo const &) {}
33803
33804     bool TestEventListenerBase::assertionEnded(AssertionStats const &) {
33805         return false;
33806     }
33807
33808 } // end namespace Catch
33809 // end catch_reporter_bases.cpp
33810 // start catch_reporter_compact.cpp
33811
33812 namespace {
33813
33814 #ifdef CATCH_PLATFORM_MAC
33815     const char* failedString() { return "FAILED"; }
33816     const char* passedString() { return "PASSED"; }
33817 #else
33818     const char* failedString() { return "failed"; }
33819     const char* passedString() { return "passed"; }
33820 #endif
33821
33822     // Colour::LightGrey
33823     Catch::Colour::Code dimColour() { return Catch::Colour::FileName; }
33824
33825     std::string bothOrAll( std::size_t count ) {
33826         return count == 1 ? std::string() :
33827             count == 2 ? "both " : "all ";
33828     }
33829
33830 } // anon namespace
33831
33832 namespace Catch {
33833 namespace {

```

```

33834 // Colour, message variants:
33835 // - white: No tests ran.
33836 // - red: Failed [both/all] N test cases, failed [both/all] M assertions.
33837 // - white: Passed [both/all] N test cases (no assertions).
33838 // - red: Failed N tests cases, failed M assertions.
33839 // - green: Passed [both/all] N tests cases with M assertions.
33840 void printTotals(std::ostream& out, const Totals& totals) {
33841     if (totals.testCases.total() == 0) {
33842         out << "No tests ran.";
33843     } else if (totals.testCases.failed == totals.testCases.total()) {
33844         Colour colour(Colour::ResultError);
33845         const std::string qualify_assertions_failed =
33846             totals.assertions.failed == totals.assertions.total() ?
33847             bothOrAll(totals.assertions.failed) : std::string();
33848         out <<
33849             "Failed " << bothOrAll(totals.testCases.failed)
33850             << pluralise(totals.testCases.failed, "test case") << ", "
33851             "failed " << qualify_assertions_failed <<
33852             pluralise(totals.assertions.failed, "assertion") << ' ';
33853     } else if (totals.assertions.total() == 0) {
33854         out <<
33855             "Passed " << bothOrAll(totals.testCases.total())
33856             << pluralise(totals.testCases.total(), "test case")
33857             << " (no assertions).";
33858     } else if (totals.assertions.failed) {
33859         Colour colour(Colour::ResultError);
33860         out <<
33861             "Failed " << pluralise(totals.testCases.failed, "test case") << ", "
33862             "failed " << pluralise(totals.assertions.failed, "assertion") << ' ';
33863     } else {
33864         Colour colour(Colour::ResultSuccess);
33865         out <<
33866             "Passed " << bothOrAll(totals.testCases.passed)
33867             << pluralise(totals.testCases.passed, "test case") <<
33868             " with " << pluralise(totals.assertions.passed, "assertion") << ' ';
33869     }
33870 }
33871
33872 // Implementation of CompactReporter formatting
33873 class AssertionPrinter {
33874 public:
33875     AssertionPrinter& operator= (AssertionPrinter const&) = delete;
33876     AssertionPrinter(AssertionPrinter const&) = delete;
33877     AssertionPrinter(std::ostream& _stream, AssertionStats const& _stats, bool _printInfoMessages)
33878         : stream(_stream)
33879         , result(_stats.assertionResult)
33880         , messages(_stats.infoMessages)
33881         , itMessage(_stats.infoMessages.begin())
33882         , printInfoMessages(_printInfoMessages) {}
33883
33884     void print() {
33885         printSourceInfo();
33886
33887         itMessage = messages.begin();
33888
33889         switch (result.getResultType()) {
33890             case ResultWas::Ok:
33891                 printResultType(Colour::ResultSuccess, passedString());
33892                 printOriginalExpression();
33893                 printReconstructedExpression();
33894                 if (!result.hasExpression())
33895                     printRemainingMessages(Colour::None);
33896                 else
33897                     printRemainingMessages();
33898                 break;
33899             case ResultWas::ExpressionFailed:
33900                 if (result.isOk())
33901                     printResultType(Colour::ResultSuccess, failedString() + std::string(" - but was ok"));
33902                 else
33903                     printResultType(Colour::Error, failedString());
33904                 printOriginalExpression();
33905                 printReconstructedExpression();
33906                 printRemainingMessages();
33907                 break;
33908             case ResultWas::ThrewException:
33909                 printResultType(Colour::Error, failedString());
33910                 printIssue("unexpected exception with message:");
33911                 printMessage();
33912                 printExpressionWas();
33913                 printRemainingMessages();
33914                 break;
33915             case ResultWas::FatalErrorCondition:
33916                 printResultType(Colour::Error, failedString());
33917                 printIssue("fatal error condition with message:");
33918                 printMessage();
33919                 printExpressionWas();
33920                 printRemainingMessages();

```

```

33921         break;
33922     case ResultWas::DidntThrowException:
33923         printResultType(Colour::Error, failedString());
33924         printIssue("expected exception, got none");
33925         printExpressionWas();
33926         printRemainingMessages();
33927         break;
33928     case ResultWas::Info:
33929         printResultType(Colour::None, "info");
33930         printMessage();
33931         printRemainingMessages();
33932         break;
33933     case ResultWas::Warning:
33934         printResultType(Colour::None, "warning");
33935         printMessage();
33936         printRemainingMessages();
33937         break;
33938     case ResultWas::ExplicitFailure:
33939         printResultType(Colour::Error, failedString());
33940         printIssue("explicitly");
33941         printRemainingMessages(Colour::None);
33942         break;
33943         // These cases are here to prevent compiler warnings
33944     case ResultWas::Unknown:
33945     case ResultWas::FailureBit:
33946     case ResultWas::Exception:
33947         printResultType(Colour::Error, "** internal error **");
33948         break;
33949     }
33950 }
33951
33952 private:
33953     void printSourceInfo() const {
33954         Colour colourGuard(Colour::FileName);
33955         stream << result.getSourceInfo() << ':';
33956     }
33957
33958     void printResultType(Colour::Code colour, std::string const& passOrFail) const {
33959         if (!passOrFail.empty()) {
33960             {
33961                 Colour colourGuard(colour);
33962                 stream << ' ' << passOrFail;
33963             }
33964             stream << ':';
33965         }
33966     }
33967
33968     void printIssue(std::string const& issue) const {
33969         stream << ' ' << issue;
33970     }
33971
33972     void printExpressionWas() {
33973         if (result.hasExpression()) {
33974             stream << ' ';
33975             {
33976                 Colour colour(dimColour());
33977                 stream << " expression was:";
33978             }
33979             printOriginalExpression();
33980         }
33981     }
33982
33983     void printOriginalExpression() const {
33984         if (result.hasExpression()) {
33985             stream << ' ' << result.getExpression();
33986         }
33987     }
33988
33989     void printReconstructedExpression() const {
33990         if (result.hasExpandedExpression()) {
33991             {
33992                 Colour colour(dimColour());
33993                 stream << " for: ";
33994             }
33995             stream << result.getExpandedExpression();
33996         }
33997     }
33998
33999     void printMessage() {
34000         if (itMessage != messages.end()) {
34001             stream << " " << itMessage->message << '\n';
34002             ++itMessage;
34003         }
34004     }
34005
34006     void printRemainingMessages(Colour::Code colour = dimColour()) {
34007         if (itMessage == messages.end())

```

```

34008         return;
34009
34010     const auto itEnd = messages.cend();
34011     const auto N = static_cast<std::size_t>(std::distance(itMessage, itEnd));
34012
34013     {
34014         Colour colourGuard(colour);
34015         stream << " with " << pluralise(N, "message") << ':';
34016     }
34017
34018     while (itMessage != itEnd) {
34019         // If this assertion is a warning ignore any INFO messages
34020         if (printInfoMessages || itMessage->type != ResultWas::Info) {
34021             printMessage();
34022             if (itMessage != itEnd) {
34023                 Colour colourGuard(dimColour());
34024                 stream << " and";
34025             }
34026             continue;
34027         }
34028         ++itMessage;
34029     }
34030 }
34031
34032 private:
34033     std::ostream& stream;
34034     AssertionResult const& result;
34035     std::vector<MessageInfo> messages;
34036     std::vector<MessageInfo>::const_iterator itMessage;
34037     bool printInfoMessages;
34038 };
34039
34040 } // anon namespace
34041
34042     std::string CompactReporter::getDescription() {
34043         return "Reports test results on a single line, suitable for IDEs";
34044     }
34045
34046     void CompactReporter::noMatchingTestCases( std::string const& spec ) {
34047         stream << "No test cases matched '" << spec << "'\n" << std::endl;
34048     }
34049
34050     void CompactReporter::assertionStarting( AssertionInfo const& ) {}
34051
34052     bool CompactReporter::assertionEnded( AssertionStats const& _assertionStats ) {
34053         AssertionResult const& result = _assertionStats.assertionResult;
34054
34055         bool printInfoMessages = true;
34056
34057         // Drop out if result was successful and we're not printing those
34058         if ( !m_config->includeSuccessfulResults() && result.isOk() ) {
34059             if( result.getResultType() != ResultWas::Warning )
34060                 return false;
34061             printInfoMessages = false;
34062         }
34063
34064         AssertionPrinter printer( stream, _assertionStats, printInfoMessages );
34065         printer.print();
34066
34067         stream << std::endl;
34068         return true;
34069     }
34070
34071     void CompactReporter::sectionEnded(SectionStats const& _sectionStats) {
34072         double dur = _sectionStats.durationInSeconds;
34073         if ( shouldShowDuration( *m_config, dur ) ) {
34074             stream << getFormattedDuration( dur ) << " s: " << _sectionStats.sectionInfo.name <<
std::endl;
34075         }
34076     }
34077
34078     void CompactReporter::testRunEnded( TestRunStats const& _testRunStats ) {
34079         printTotals( stream, _testRunStats.totals );
34080         stream << '\n' << std::endl;
34081         StreamingReporterBase::testRunEnded( _testRunStats );
34082     }
34083
34084     CompactReporter::~CompactReporter() {}
34085
34086     CATCH_REGISTER_REPORTER( "compact", CompactReporter )
34087
34088 } // end namespace Catch
34089 // end catch_reporter_compact.cpp
34090 // start catch_reporter_console.cpp
34091
34092 #include <cfloat>
34093 #include <cstdint>

```

```

34094
34095 #if defined(_MSC_VER)
34096 #pragma warning(push)
34097 #pragma warning(disable:4061) // Not all labels are EXPLICITLY handled in switch
34098 // Note that 4062 (not all labels are handled and default is missing) is enabled
34099 #endif
34100
34101 #if defined(__clang__)
34102 # pragma clang diagnostic push
34103 // For simplicity, benchmarking-only helpers are always enabled
34104 # pragma clang diagnostic ignored "-Wunused-function"
34105 #endif
34106
34107 namespace Catch {
34108
34109 namespace {
34110
34111 // Formatter impl for ConsoleReporter
34112 class ConsoleAssertionPrinter {
34113 public:
34114     ConsoleAssertionPrinter& operator= (ConsoleAssertionPrinter const&) = delete;
34115     ConsoleAssertionPrinter(ConsoleAssertionPrinter const&) = delete;
34116     ConsoleAssertionPrinter(std::ostream& _stream, AssertionStats const& _stats, bool
_printInfoMessages)
34117         : stream(_stream),
34118           stats(_stats),
34119           result(_stats.assertionResult),
34120           colour(Colour::None),
34121           message(result.getMessage()),
34122           messages(_stats.infoMessages),
34123           printInfoMessages(_printInfoMessages) {
34124         switch (result.getResultType()) {
34125             case ResultWas::Ok:
34126                 colour = Colour::Success;
34127                 passOrFail = "PASSED";
34128                 //if( result.hasMessage() )
34129                 if (_stats.infoMessages.size() == 1)
34130                     messageLabel = "with message";
34131                 if (_stats.infoMessages.size() > 1)
34132                     messageLabel = "with messages";
34133                 break;
34134             case ResultWas::ExpressionFailed:
34135                 if (result.isOk()) {
34136                     colour = Colour::Success;
34137                     passOrFail = "FAILED - but was ok";
34138                 } else {
34139                     colour = Colour::Error;
34140                     passOrFail = "FAILED";
34141                 }
34142                 if (_stats.infoMessages.size() == 1)
34143                     messageLabel = "with message";
34144                 if (_stats.infoMessages.size() > 1)
34145                     messageLabel = "with messages";
34146                 break;
34147             case ResultWas::ThrewException:
34148                 colour = Colour::Error;
34149                 passOrFail = "FAILED";
34150                 messageLabel = "due to unexpected exception with ";
34151                 if (_stats.infoMessages.size() == 1)
34152                     messageLabel += "message";
34153                 if (_stats.infoMessages.size() > 1)
34154                     messageLabel += "messages";
34155                 break;
34156             case ResultWas::FatalErrorCondition:
34157                 colour = Colour::Error;
34158                 passOrFail = "FAILED";
34159                 messageLabel = "due to a fatal error condition";
34160                 break;
34161             case ResultWas::DidntThrowException:
34162                 colour = Colour::Error;
34163                 passOrFail = "FAILED";
34164                 messageLabel = "because no exception was thrown where one was expected";
34165                 break;
34166             case ResultWas::Info:
34167                 messageLabel = "info";
34168                 break;
34169             case ResultWas::Warning:
34170                 messageLabel = "warning";
34171                 break;
34172             case ResultWas::ExplicitFailure:
34173                 passOrFail = "FAILED";
34174                 colour = Colour::Error;
34175                 if (_stats.infoMessages.size() == 1)
34176                     messageLabel = "explicitly with message";
34177                 if (_stats.infoMessages.size() > 1)
34178                     messageLabel = "explicitly with messages";
34179                 break;

```

```

34180         // These cases are here to prevent compiler warnings
34181         case ResultWas::Unknown:
34182         case ResultWas::FailureBit:
34183         case ResultWas::Exception:
34184             passOrFail = "*** internal error **";
34185             colour = Colour::Error;
34186             break;
34187     }
34188 }
34189
34190 void print() const {
34191     printSourceInfo();
34192     if (stats.totals.assertions.total() > 0) {
34193         printResultType();
34194         printOriginalExpression();
34195         printReconstructedExpression();
34196     } else {
34197         stream << '\n';
34198     }
34199     printMessage();
34200 }
34201
34202 private:
34203     void printResultType() const {
34204         if (!passOrFail.empty()) {
34205             Colour colourGuard(colour);
34206             stream << passOrFail << "\n";
34207         }
34208     }
34209     void printOriginalExpression() const {
34210         if (result.hasExpression()) {
34211             Colour colourGuard(Colour::OriginalExpression);
34212             stream << " ";
34213             stream << result.getExpressionInMacro();
34214             stream << '\n';
34215         }
34216     }
34217     void printReconstructedExpression() const {
34218         if (result.hasExpandedExpression()) {
34219             stream << "with expansion:\n";
34220             Colour colourGuard(Colour::ReconstructedExpression);
34221             stream << Column(result.getExpandedExpression()).indent(2) << '\n';
34222         }
34223     }
34224     void printMessage() const {
34225         if (!messageLabel.empty())
34226             stream << messageLabel << ':' << '\n';
34227         for (auto const& msg : messages) {
34228             // If this assertion is a warning ignore any INFO messages
34229             if (printInfoMessages || msg.type != ResultWas::Info)
34230                 stream << Column(msg.message).indent(2) << '\n';
34231         }
34232     }
34233     void printSourceInfo() const {
34234         Colour colourGuard(Colour::FileName);
34235         stream << result.getSourceInfo() << ": ";
34236     }
34237
34238     std::ostream& stream;
34239     AssertionStats const& stats;
34240     AssertionResult const& result;
34241     Colour::Code colour;
34242     std::string passOrFail;
34243     std::string messageLabel;
34244     std::string message;
34245     std::vector<MessageInfo> messages;
34246     bool printInfoMessages;
34247 };
34248
34249 std::size_t makeRatio(std::size_t number, std::size_t total) {
34250     std::size_t ratio = total > 0 ? CATCH_CONFIG_CONSOLE_WIDTH * number / total : 0;
34251     return (ratio == 0 && number > 0) ? 1 : ratio;
34252 }
34253
34254 std::size_t& findMax(std::size_t& i, std::size_t& j, std::size_t& k) {
34255     if (i > j && i > k)
34256         return i;
34257     else if (j > k)
34258         return j;
34259     else
34260         return k;
34261 }
34262
34263 struct ColumnInfo {
34264     enum Justification { Left, Right };
34265     std::string name;
34266     int width;

```

```

34267     Justification justification;
34268 };
34269 struct ColumnBreak {};
34270 struct RowBreak {};
34271
34272 class Duration {
34273     enum class Unit {
34274         Auto,
34275         Nanoseconds,
34276         Microseconds,
34277         Milliseconds,
34278         Seconds,
34279         Minutes
34280     };
34281     static const uint64_t s_nanosecondsInAMicrosecond = 1000;
34282     static const uint64_t s_nanosecondsInAMillisecond = 1000 * s_nanosecondsInAMicrosecond;
34283     static const uint64_t s_nanosecondsInASecond = 1000 * s_nanosecondsInAMillisecond;
34284     static const uint64_t s_nanosecondsInAMinute = 60 * s_nanosecondsInASecond;
34285
34286     double m_inNanoseconds;
34287     Unit m_units;
34288
34289 public:
34290     explicit Duration(double inNanoseconds, Unit units = Unit::Auto)
34291         : m_inNanoseconds(inNanoseconds),
34292           m_units(units) {
34293         if (m_units == Unit::Auto) {
34294             if (m_inNanoseconds < s_nanosecondsInAMicrosecond)
34295                 m_units = Unit::Nanoseconds;
34296             else if (m_inNanoseconds < s_nanosecondsInAMillisecond)
34297                 m_units = Unit::Microseconds;
34298             else if (m_inNanoseconds < s_nanosecondsInASecond)
34299                 m_units = Unit::Milliseconds;
34300             else if (m_inNanoseconds < s_nanosecondsInAMinute)
34301                 m_units = Unit::Seconds;
34302             else
34303                 m_units = Unit::Minutes;
34304         }
34305     }
34306
34307     auto value() const -> double {
34308         switch (m_units) {
34309             case Unit::Microseconds:
34310                 return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMicrosecond);
34311             case Unit::Milliseconds:
34312                 return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMillisecond);
34313             case Unit::Seconds:
34314                 return m_inNanoseconds / static_cast<double>(s_nanosecondsInASecond);
34315             case Unit::Minutes:
34316                 return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMinute);
34317             default:
34318                 return m_inNanoseconds;
34319         }
34320     }
34321
34322     auto unitsAsString() const -> std::string {
34323         switch (m_units) {
34324             case Unit::Nanoseconds:
34325                 return "ns";
34326             case Unit::Microseconds:
34327                 return "us";
34328             case Unit::Milliseconds:
34329                 return "ms";
34330             case Unit::Seconds:
34331                 return "s";
34332             case Unit::Minutes:
34333                 return "m";
34334             default:
34335                 return "*** internal error **";
34336         }
34337     }
34338
34339     friend auto operator << (std::ostream& os, Duration const& duration) -> std::ostream& {
34340         return os << duration.value() << ' ' << duration.unitsAsString();
34341     }
34342 };
34343 } // end anon namespace
34344
34345 class TablePrinter {
34346     std::ostream& m_os;
34347     std::vector<ColumnInfo> m_columnInfos;
34348     std::ostringstream m_oss;
34349     int m_currentColumn = -1;
34350     bool m_isOpen = false;
34351
34352 public:
34353     TablePrinter( std::ostream& os, std::vector<ColumnInfo> columnInfos )

```

```

34354 : m_os( os ),
34355 m_columnInfos( std::move( columnInfos ) ) {}
34356
34357 auto columnInfos() const -> std::vector<ColumnInfo> const& {
34358     return m_columnInfos;
34359 }
34360
34361 void open() {
34362     if (!m_isOpen) {
34363         m_isOpen = true;
34364         *this << RowBreak();
34365
34366         Columns headerCols;
34367         Spacer spacer(2);
34368         for (auto const& info : m_columnInfos) {
34369             headerCols += Column(info.name).width(static_cast<std::size_t>(info.width - 2));
34370             headerCols += spacer;
34371         }
34372         m_os << headerCols << '\n';
34373
34374         m_os << Catch::getLineOfChars<'-'>() << '\n';
34375     }
34376 }
34377 void close() {
34378     if (m_isOpen) {
34379         *this << RowBreak();
34380         m_os << std::endl;
34381         m_isOpen = false;
34382     }
34383 }
34384
34385 template<typename T>
34386 friend TablePrinter& operator << (TablePrinter& tp, T const& value) {
34387     tp.m_oss << value;
34388     return tp;
34389 }
34390
34391 friend TablePrinter& operator << (TablePrinter& tp, ColumnBreak) {
34392     auto colStr = tp.m_oss.str();
34393     const auto strSize = colStr.size();
34394     tp.m_oss.str("");
34395     tp.open();
34396     if (tp.m_currentColumn == static_cast<int>(tp.m_columnInfos.size() - 1)) {
34397         tp.m_currentColumn = -1;
34398         tp.m_os << '\n';
34399     }
34400     tp.m_currentColumn++;
34401
34402     auto colInfo = tp.m_columnInfos[tp.m_currentColumn];
34403     auto padding = (strSize + 1 < static_cast<std::size_t>(colInfo.width))
34404         ? std::string(colInfo.width - (strSize + 1), ' ')
34405         : std::string();
34406     if (colInfo.justification == ColumnInfo::Left)
34407         tp.m_os << colStr << padding << ' ';
34408     else
34409         tp.m_os << padding << colStr << ' ';
34410     return tp;
34411 }
34412
34413 friend TablePrinter& operator << (TablePrinter& tp, RowBreak) {
34414     if (tp.m_currentColumn > 0) {
34415         tp.m_os << '\n';
34416         tp.m_currentColumn = -1;
34417     }
34418     return tp;
34419 }
34420 };
34421
34422 ConsoleReporter::ConsoleReporter(ReporterConfig const& config)
34423 : StreamingReporterBase(config),
34424 m_tablePrinter(new TablePrinter(config.stream(),
34425     [&config]() -> std::vector<ColumnInfo> {
34426         if (config.fullConfig()->benchmarkNoAnalysis())
34427         {
34428             return{
34429                 { "benchmark name", CATCH_CONFIG_CONSOLE_WIDTH - 43, ColumnInfo::Left },
34430                 { "    samples", 14, ColumnInfo::Right },
34431                 { "  iterations", 14, ColumnInfo::Right },
34432                 { "           mean", 14, ColumnInfo::Right }
34433             };
34434         }
34435         else
34436         {
34437             return{
34438                 { "benchmark name", CATCH_CONFIG_CONSOLE_WIDTH - 43, ColumnInfo::Left },
34439                 { "samples      mean          std dev", 14, ColumnInfo::Right },
34440                 { "iterations  low mean   low std dev", 14, ColumnInfo::Right },

```



```

34441         { "estimated    high mean  high std dev", 14, ColumnInfo::Right }
34442     };
34443     }
34444     }()) {}
34445     ConsoleReporter::~ConsoleReporter() = default;
34446
34447     std::string ConsoleReporter::getDescription() {
34448         return "Reports test results as plain lines of text";
34449     }
34450
34451     void ConsoleReporter::noMatchingTestCases(std::string const& spec) {
34452         stream << "No test cases matched '" << spec << "'\n" << std::endl;
34453     }
34454
34455     void ConsoleReporter::reportInvalidArguments(std::string const& arg) {
34456         stream << "Invalid Filter: " << arg << std::endl;
34457     }
34458
34459     void ConsoleReporter::assertionStarting(AssertionInfo const&) {}
34460
34461     bool ConsoleReporter::assertionEnded(AssertionStats const& _assertionStats) {
34462         AssertionResult const& result = _assertionStats.assertionResult;
34463
34464         bool includeResults = m_config->includeSuccessfulResults() || !result.isOk();
34465
34466         // Drop out if result was successful but we're not printing them.
34467         if (!includeResults && result.getResultType() != ResultWas::Warning)
34468             return false;
34469
34470         lazyPrint();
34471
34472         ConsoleAssertionPrinter printer(stream, _assertionStats, includeResults);
34473         printer.print();
34474         stream << std::endl;
34475         return true;
34476     }
34477
34478     void ConsoleReporter::sectionStarting(SectionInfo const& _sectionInfo) {
34479         m_tablePrinter->close();
34480         m_headerPrinted = false;
34481         StreamingReporterBase::sectionStarting(_sectionInfo);
34482     }
34483
34484     void ConsoleReporter::sectionEnded(SectionStats const& _sectionStats) {
34485         m_tablePrinter->close();
34486         if (_sectionStats.missingAssertions) {
34487             lazyPrint();
34488             Colour colour(Colour::ResultError);
34489             if (m_sectionStack.size() > 1)
34490                 stream << "\nNo assertions in section";
34491             else
34492                 stream << "\nNo assertions in test case";
34493             stream << " '" << _sectionStats.sectionInfo.name << "'\n" << std::endl;
34494         }
34495         double dur = _sectionStats.durationInSeconds;
34496         if (shouldShowDuration(*m_config, dur)) {
34497             stream << getFormattedDuration(dur) << " s: " << _sectionStats.sectionInfo.name << std::endl;
34498         }
34499         if (m_headerPrinted) {
34500             m_headerPrinted = false;
34501         }
34502         StreamingReporterBase::sectionEnded(_sectionStats);
34503     }
34504
34505     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
34506     void ConsoleReporter::benchmarkPreparing(std::string const& name) {
34507         lazyPrintWithoutClosingBenchmarkTable();
34508
34509         auto nameCol = Column(name).width(static_cast<std::size_t>(m_tablePrinter->columnInfos()[0].width
34510 - 2));
34511
34512         bool firstLine = true;
34513         for (auto line : nameCol) {
34514             if (!firstLine)
34515                 (*m_tablePrinter) << ColumnBreak() << ColumnBreak() << ColumnBreak();
34516             else
34517                 firstLine = false;
34518             (*m_tablePrinter) << line << ColumnBreak();
34519         }
34520
34521     void ConsoleReporter::benchmarkStarting(BenchmarkInfo const& info) {
34522         (*m_tablePrinter) << info.samples << ColumnBreak()
34523             << info.iterations << ColumnBreak();
34524         if (!m_config->benchmarkNoAnalysis())
34525             (*m_tablePrinter) << Duration(info.estimatedDuration) << ColumnBreak();
34526     }

```

```

34527 void ConsoleReporter::benchmarkEnded(BenchmarkStats<> const& stats) {
34528     if (m_config->benchmarkNoAnalysis())
34529     {
34530         (*m_tablePrinter) << Duration(stats.mean.point.count()) << ColumnBreak();
34531     }
34532     else
34533     {
34534         (*m_tablePrinter) << ColumnBreak()
34535             << Duration(stats.mean.point.count()) << ColumnBreak()
34536             << Duration(stats.mean.lower_bound.count()) << ColumnBreak()
34537             << Duration(stats.mean.upper_bound.count()) << ColumnBreak() << ColumnBreak()
34538             << Duration(stats.standardDeviation.point.count()) << ColumnBreak()
34539             << Duration(stats.standardDeviation.lower_bound.count()) << ColumnBreak()
34540             << Duration(stats.standardDeviation.upper_bound.count()) << ColumnBreak() << ColumnBreak() <<
ColumnBreak() << ColumnBreak() << ColumnBreak();
34541     }
34542 }
34543
34544 void ConsoleReporter::benchmarkFailed(std::string const& error) {
34545     Colour colour(Colour::Red);
34546     (*m_tablePrinter)
34547         << "Benchmark failed (" << error << ') '
34548         << ColumnBreak() << RowBreak();
34549 }
34550 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
34551
34552 void ConsoleReporter::testCaseEnded(TestCaseStats const& _testCaseStats) {
34553     m_tablePrinter->close();
34554     StreamingReporterBase::testCaseEnded(_testCaseStats);
34555     m_headerPrinted = false;
34556 }
34557 void ConsoleReporter::testGroupEnded(TestGroupStats const& _testGroupStats) {
34558     if (currentGroupInfo.used) {
34559         printSummaryDivider();
34560         stream << "Summary for group '" << _testGroupStats.groupInfo.name << "':\n";
34561         printTotals(_testGroupStats.totals);
34562         stream << '\n' << std::endl;
34563     }
34564     StreamingReporterBase::testGroupEnded(_testGroupStats);
34565 }
34566 void ConsoleReporter::testRunEnded(TestRunStats const& _testRunStats) {
34567     printTotalsDivider(_testRunStats.totals);
34568     printTotals(_testRunStats.totals);
34569     stream << std::endl;
34570     StreamingReporterBase::testRunEnded(_testRunStats);
34571 }
34572 void ConsoleReporter::testRunStarting(TestRunInfo const& _testInfo) {
34573     StreamingReporterBase::testRunStarting(_testInfo);
34574     printTestFilters();
34575 }
34576
34577 void ConsoleReporter::lazyPrint() {
34578     m_tablePrinter->close();
34579     lazyPrintWithoutClosingBenchmarkTable();
34580 }
34581
34582 void ConsoleReporter::lazyPrintWithoutClosingBenchmarkTable() {
34583     if (!currentTestRunInfo.used)
34584         lazyPrintRunInfo();
34585     if (!currentGroupInfo.used)
34586         lazyPrintGroupInfo();
34587     if (!m_headerPrinted) {
34588         printTestCaseAndSectionHeader();
34589         m_headerPrinted = true;
34590     }
34591 }
34592 void ConsoleReporter::lazyPrintRunInfo() {
34593     stream << '\n' << getLineOfChars<'~'>() << '\n';
34594     Colour colour(Colour::SecondaryText);
34595     stream << currentTestRunInfo->name
34596         << " is a Catch v" << libraryVersion() << " host application.\n"
34597         << "Run with -? for options\n\n";
34598     if (m_config->rngSeed() != 0)
34599         stream << "Randomness seeded to: " << m_config->rngSeed() << "\n\n";
34600     currentTestRunInfo.used = true;
34601 }
34602 void ConsoleReporter::lazyPrintGroupInfo() {
34603     if (!currentGroupInfo->name.empty() && currentGroupInfo->groupsCounts > 1) {
34604         printClosedHeader("Group: " + currentGroupInfo->name);
34605         currentGroupInfo.used = true;
34606     }
34607 }
34608
34609 }
34610
34611 }
34612

```

```

34613 void ConsoleReporter::printTestCaseAndSectionHeader() {
34614     assert(!m_sectionStack.empty());
34615     printOpenHeader(currentTestCaseInfo->name);
34616
34617     if (m_sectionStack.size() > 1) {
34618         Colour colourGuard(Colour::Headers);
34619
34620         auto
34621             it = m_sectionStack.begin() + 1, // Skip first section (test case)
34622             itEnd = m_sectionStack.end();
34623         for (; it != itEnd; ++it)
34624             printHeaderString(it->name, 2);
34625     }
34626
34627     SourceLineInfo lineInfo = m_sectionStack.back().lineInfo;
34628
34629     stream << getLineOfChars<'-'>() << '\n';
34630     Colour colourGuard(Colour::FileName);
34631     stream << lineInfo << '\n';
34632     stream << getLineOfChars<'.'>() << '\n' << std::endl;
34633 }
34634
34635 void ConsoleReporter::printClosedHeader(std::string const& _name) {
34636     printOpenHeader(_name);
34637     stream << getLineOfChars<'.'>() << '\n';
34638 }
34639 void ConsoleReporter::printOpenHeader(std::string const& _name) {
34640     stream << getLineOfChars<'-'>() << '\n';
34641     {
34642         Colour colourGuard(Colour::Headers);
34643         printHeaderString(_name);
34644     }
34645 }
34646
34647 // if string has a : in first line will set indent to follow it on
34648 // subsequent lines
34649 void ConsoleReporter::printHeaderString(std::string const& _string, std::size_t indent) {
34650     std::size_t i = _string.find(": ");
34651     if (i != std::string::npos)
34652         i += 2;
34653     else
34654         i = 0;
34655     stream << Column(_string).indent(indent + i).initialIndent(indent) << '\n';
34656 }
34657
34658 struct SummaryColumn {
34659     SummaryColumn( std::string _label, Colour::Code _colour )
34660     :   label( std::move( _label ) ),
34661         colour( _colour ) {}
34662     SummaryColumn addRow( std::size_t count ) {
34663         ReusableStringStream rss;
34664         rss << count;
34665         std::string row = rss.str();
34666         for (auto& oldRow : rows) {
34667             while (oldRow.size() < row.size())
34668                 oldRow = ' ' + oldRow;
34669             while (oldRow.size() > row.size())
34670                 row = ' ' + row;
34671         }
34672         rows.push_back(row);
34673         return *this;
34674     }
34675
34676     std::string label;
34677     Colour::Code colour;
34678     std::vector<std::string> rows;
34679 };
34680
34681 void ConsoleReporter::printTotals( Totals const& totals ) {
34682     if (totals.testCases.total() == 0) {
34683         stream << Colour(Colour::Warning) << "No tests ran\n";
34684     } else if (totals.assertions.total() > 0 && totals.testCases.allPassed()) {
34685         stream << Colour(Colour::ResultSuccess) << "All tests passed";
34686         stream << " ("
34687             << pluralise(totals.assertions.passed, "assertion") << " in "
34688             << pluralise(totals.testCases.passed, "test case") << ') '
34689             << '\n';
34690     } else {
34691         std::vector<SummaryColumn> columns;
34692         columns.push_back(SummaryColumn("", Colour::None)
34693             .addRow(totals.testCases.total())
34694             .addRow(totals.assertions.total()));
34695         columns.push_back(SummaryColumn("passed", Colour::Success)
34696             .addRow(totals.testCases.passed)

```

```

34700         .addRow(totals.assertions.passed));
34701     columns.push_back(SummaryColumn("failed", Colour::ResultError)
34702         .addRow(totals.testCases.failed)
34703         .addRow(totals.assertions.failed));
34704     columns.push_back(SummaryColumn("failed as expected", Colour::ResultExpectedFailure)
34705         .addRow(totals.testCases.failedButOk)
34706         .addRow(totals.assertions.failedButOk));
34707
34708     printSummaryRow("test cases", columns, 0);
34709     printSummaryRow("assertions", columns, 1);
34710 }
34711 }
34712 void ConsoleReporter::printSummaryRow(std::string const& label, std::vector<SummaryColumn> const&
34713     cols, std::size_t row) {
34714     for (auto col : cols) {
34715         std::string value = col.rows[row];
34716         if (col.label.empty()) {
34717             stream << label << ": ";
34718             if (value != "0")
34719                 stream << value;
34720             else
34721                 stream << Colour(Colour::Warning) << "- none -";
34722         } else if (value != "0") {
34723             stream << Colour(Colour::LightGrey) << " | ";
34724             stream << Colour(col.colour)
34725                 << value << ' ' << col.label;
34726         }
34727     }
34728     stream << '\n';
34729 }
34730 void ConsoleReporter::printTotalsDivider(Totals const& totals) {
34731     if (totals.testCases.total() > 0) {
34732         std::size_t failedRatio = makeRatio(totals.testCases.failed, totals.testCases.total());
34733         std::size_t failedButOkRatio = makeRatio(totals.testCases.failedButOk,
34734             totals.testCases.total());
34735         std::size_t passedRatio = makeRatio(totals.testCases.passed, totals.testCases.total());
34736         while (failedRatio + failedButOkRatio + passedRatio < CATCH_CONFIG_CONSOLE_WIDTH - 1)
34737             findMax(failedRatio, failedButOkRatio, passedRatio)++;
34738         while (failedRatio + failedButOkRatio + passedRatio > CATCH_CONFIG_CONSOLE_WIDTH - 1)
34739             findMax(failedRatio, failedButOkRatio, passedRatio)--;
34740
34741         stream << Colour(Colour::Error) << std::string(failedRatio, '=');
34742         stream << Colour(Colour::ResultExpectedFailure) << std::string(failedButOkRatio, '=');
34743         if (totals.testCases.allPassed())
34744             stream << Colour(Colour::ResultSuccess) << std::string(passedRatio, '=');
34745         else
34746             stream << Colour(Colour::Success) << std::string(passedRatio, '=');
34747     } else {
34748         stream << Colour(Colour::Warning) << std::string(CATCH_CONFIG_CONSOLE_WIDTH - 1, '=');
34749     }
34750     stream << '\n';
34751 }
34752 void ConsoleReporter::printSummaryDivider() {
34753     stream << getLineOfChars<'-'>() << '\n';
34754 }
34755 void ConsoleReporter::printTestFilters() {
34756     if (m_config->testSpec().hasFilters()) {
34757         Colour guard(Colour::BrightYellow);
34758         stream << "Filters: " << serializeFilters(m_config->getTestsOrTags()) << '\n';
34759     }
34760 }
34761
34762 CATCH_REGISTER_REPORTER("console", ConsoleReporter)
34763
34764 } // end namespace Catch
34765
34766 #if defined(_MSC_VER)
34767 #pragma warning(pop)
34768 #endif
34769
34770 #if defined(__clang__)
34771 #pragma clang diagnostic pop
34772 #endif
34773 // end catch_reporter_console.cpp
34774 // start catch_reporter_junit.cpp
34775
34776 #include <cassert>
34777 #include <sstream>
34778 #include <ctime>
34779 #include <algorithm>
34780 #include <iomanip>
34781
34782 namespace Catch {
34783
34784     namespace {

```

```

34785         std::string getCurrentTimestamp() {
34786             // Beware, this is not reentrant because of backward compatibility issues
34787             // Also, UTC only, again because of backward compatibility (%z is C++11)
34788             time_t rawtime;
34789             std::time(&rawtime);
34790             auto const timeStampSize = sizeof("2017-01-16T17:06:45Z");
34791
34792             #ifdef _MSC_VER
34793                 std::tm timeInfo = {};
34794                 gmtime_s(&timeInfo, &rawtime);
34795             #else
34796                 std::tm* timeInfo;
34797                 timeInfo = std::gmtime(&rawtime);
34798             #endif
34799
34800             char timeStamp[timeStampSize];
34801             const char * const fmt = "%Y-%m-%dT%H:%M:%SZ";
34802
34803             #ifdef _MSC_VER
34804                 std::strftime(timeStamp, timeStampSize, fmt, &timeInfo);
34805             #else
34806                 std::strftime(timeStamp, timeStampSize, fmt, timeInfo);
34807             #endif
34808             return std::string(timeStamp, timeStampSize-1);
34809         }
34810
34811         std::string fileNameTag(const std::vector<std::string> &tags) {
34812             auto it = std::find_if(begin(tags),
34813                                   end(tags),
34814                                   [](const std::string &tag) {return tag.front() == '#'; });
34815             if (it != tags.end())
34816                 return it->substr(1);
34817             return std::string();
34818         }
34819
34820         // Formats the duration in seconds to 3 decimal places.
34821         // This is done because some genius defined Maven Surefire schema
34822         // in a way that only accepts 3 decimal places, and tools like
34823         // Jenkins use that schema for validation JUnit reporter output.
34824         std::string formatDuration( double seconds ) {
34825             ReusableStringStream rss;
34826             rss << std::fixed << std::setprecision( 3 ) << seconds;
34827             return rss.str();
34828         }
34829
34830     } // anonymous namespace
34831
34832     JUnitReporter::JUnitReporter( ReporterConfig const& _config )
34833     :   CumulativeReporterBase( _config ),
34834         xml( _config.stream() )
34835     {
34836         m_reporterPrefs.shouldRedirectStdOut = true;
34837         m_reporterPrefs.shouldReportAllAssertions = true;
34838     }
34839
34840     JUnitReporter::~JUnitReporter() {}
34841
34842     std::string JUnitReporter::getDescription() {
34843         return "Reports test results in an XML format that looks like Ant's junitreport target";
34844     }
34845
34846     void JUnitReporter::noMatchingTestCases( std::string const& /*spec*/ ) {}
34847
34848     void JUnitReporter::testRunStarting( TestRunInfo const& runInfo ) {
34849         CumulativeReporterBase::testRunStarting( runInfo );
34850         xml.startElement( "testsuites" );
34851     }
34852
34853     void JUnitReporter::testGroupStarting( GroupInfo const& groupInfo ) {
34854         suiteTimer.start();
34855         stdOutForSuite.clear();
34856         stdErrForSuite.clear();
34857         unexpectedExceptions = 0;
34858         CumulativeReporterBase::testGroupStarting( groupInfo );
34859     }
34860
34861     void JUnitReporter::testCaseStarting( TestCaseInfo const& testCaseInfo ) {
34862         m_okToFail = testCaseInfo.okToFail();
34863     }
34864
34865     bool JUnitReporter::assertionEnded( AssertionStats const& assertionStats ) {
34866         if( assertionStats.assertionResult.getResultType() == ResultWas::ThrewException && !m_okToFail )
34867             unexpectedExceptions++;
34868         return CumulativeReporterBase::assertionEnded( assertionStats );
34869     }
34870

```

```

34871 void JunitReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
34872     stdOutForSuite += testCaseStats.stdOut;
34873     stdErrForSuite += testCaseStats.stdErr;
34874     CumulativeReporterBase::testCaseEnded( testCaseStats );
34875 }
34876
34877 void JunitReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
34878     double suiteTime = suiteTimer.getElapsedSeconds();
34879     CumulativeReporterBase::testGroupEnded( testGroupStats );
34880     writeGroup( *m_testGroups.back(), suiteTime );
34881 }
34882
34883 void JunitReporter::testRunEndedCumulative() {
34884     xml.endElement();
34885 }
34886
34887 void JunitReporter::writeGroup( TestGroupNode const& groupNode, double suiteTime ) {
34888     XmlWriter::ScopedElement e = xml.scopedElement( "testsuite" );
34889
34890     TestGroupStats const& stats = groupNode.value;
34891     xml.writeAttribute( "name", stats.groupInfo.name );
34892     xml.writeAttribute( "errors", unexpectedExceptions );
34893     xml.writeAttribute( "failures", stats.totals.assertions.failed-unexpectedExceptions );
34894     xml.writeAttribute( "tests", stats.totals.assertions.total() );
34895     xml.writeAttribute( "hostname", "tbd" ); // !TBD
34896     if( m_config->showDurations() == ShowDurations::Never )
34897         xml.writeAttribute( "time", "" );
34898     else
34899         xml.writeAttribute( "time", formatDuration( suiteTime ) );
34900     xml.writeAttribute( "timestamp", getCurrentTimestamp() );
34901
34902     // Write properties if there are any
34903     if (m_config->hasTestFilters() || m_config->rngSeed() != 0) {
34904         auto properties = xml.scopedElement("properties");
34905         if (m_config->hasTestFilters()) {
34906             xml.scopedElement("property")
34907                 .writeAttribute("name", "filters")
34908                 .writeAttribute("value", serializeFilters(m_config->getTestsOrTags()));
34909         }
34910         if (m_config->rngSeed() != 0) {
34911             xml.scopedElement("property")
34912                 .writeAttribute("name", "random-seed")
34913                 .writeAttribute("value", m_config->rngSeed());
34914         }
34915     }
34916
34917     // Write test cases
34918     for( auto const& child : groupNode.children )
34919         writeTestCase( *child );
34920
34921     xml.scopedElement( "system-out" ).writeText( trim( stdOutForSuite ), XmlFormatting::Newline );
34922     xml.scopedElement( "system-err" ).writeText( trim( stdErrForSuite ), XmlFormatting::Newline );
34923 }
34924
34925 void JunitReporter::writeTestCase( TestCaseNode const& testCaseNode ) {
34926     TestCaseStats const& stats = testCaseNode.value;
34927
34928     // All test cases have exactly one section - which represents the
34929     // test case itself. That section may have 0-n nested sections
34930     assert( testCaseNode.children.size() == 1 );
34931     SectionNode const& rootSection = *testCaseNode.children.front();
34932
34933     std::string className = stats.testInfo.className;
34934
34935     if( className.empty() ) {
34936         className = fileNameTag(stats.testInfo.tags);
34937         if ( className.empty() )
34938             className = "global";
34939     }
34940
34941     if ( !m_config->name().empty() )
34942         className = m_config->name() + "." + className;
34943
34944     writeSection( className, "", rootSection, stats.testInfo.okToFail() );
34945 }
34946
34947 void JunitReporter::writeSection( std::string const& className,
34948     std::string const& rootName,
34949     SectionNode const& sectionNode,
34950     bool testOkToFail ) {
34951     std::string name = trim( sectionNode.stats.sectionInfo.name );
34952     if( !rootName.empty() )
34953         name = rootName + '/' + name;
34954
34955     if( !sectionNode.assertions.empty() ||
34956         !sectionNode.stdOut.empty() ||
34957         !sectionNode.stdErr.empty() ) {

```

```

34958         XmlWriter::ScopedElement e = xml.scopedElement( "testcase" );
34959         if( className.empty() ) {
34960             xml.writeAttribute( "classname", name );
34961             xml.writeAttribute( "name", "root" );
34962         }
34963         else {
34964             xml.writeAttribute( "classname", className );
34965             xml.writeAttribute( "name", name );
34966         }
34967         xml.writeAttribute( "time", formatDuration( sectionNode.stats.durationInSeconds ) );
34968         // This is not ideal, but it should be enough to mimic gtest's
34969         // junit output.
34970         // Ideally the JUnit reporter would also handle `skipTest`
34971         // events and write those out appropriately.
34972         xml.writeAttribute( "status", "run" );
34973
34974         if (sectionNode.stats.assertions.failedButOk) {
34975             xml.scopedElement("skipped")
34976                 .writeAttribute("message", "TEST_CASE tagged with !mayfail");
34977         }
34978
34979         writeAssertions( sectionNode );
34980
34981         if( !sectionNode.stdOut.empty() )
34982             xml.scopedElement( "system-out" ).writeText( trim( sectionNode.stdOut ),
34983 XmlFormatting::Newline );
34984         if( !sectionNode.stdErr.empty() )
34985             xml.scopedElement( "system-err" ).writeText( trim( sectionNode.stdErr ),
34986 XmlFormatting::Newline );
34987
34988         for( auto const& childNode : sectionNode.childSections )
34989             if( className.empty() )
34990                 writeSection( name, "", *childNode, testOkToFail );
34991             else
34992                 writeSection( className, name, *childNode, testOkToFail );
34993
34994 void JunitReporter::writeAssertions( SectionNode const& sectionNode ) {
34995     for( auto const& assertion : sectionNode.assertions )
34996         writeAssertion( assertion );
34997 }
34998
34999 void JunitReporter::writeAssertion( AssertionStats const& stats ) {
35000     AssertionResult const& result = stats.assertionResult;
35001     if( !result.isOk() ) {
35002         std::string elementName;
35003         switch( result.getResultType() ) {
35004             case ResultWas::ThrewException:
35005             case ResultWas::FatalErrorCondition:
35006                 elementName = "error";
35007                 break;
35008             case ResultWas::ExplicitFailure:
35009             case ResultWas::ExpressionFailed:
35010             case ResultWas::DidntThrowException:
35011                 elementName = "failure";
35012                 break;
35013
35014             // We should never see these here:
35015             case ResultWas::Info:
35016             case ResultWas::Warning:
35017             case ResultWas::Ok:
35018             case ResultWas::Unknown:
35019             case ResultWas::FailureBit:
35020             case ResultWas::Exception:
35021                 elementName = "internalError";
35022                 break;
35023         }
35024
35025         XmlWriter::ScopedElement e = xml.scopedElement( elementName );
35026
35027         xml.writeAttribute( "message", result.getExpression() );
35028         xml.writeAttribute( "type", result.getTestMacroName() );
35029
35030         ReusableStringStream rss;
35031         if (stats.totals.assertions.total() > 0) {
35032             rss << "FAILED" << ":\n";
35033             if (result.hasExpression()) {
35034                 rss << " ";
35035                 rss << result.getExpressionInMacro();
35036                 rss << '\n';
35037             }
35038             if (result.hasExpandedExpression()) {
35039                 rss << "with expansion:\n";
35040                 rss << Column(result.getExpandedExpression()).indent(2) << '\n';
35041             }
35042         } else {
35043             rss << '\n';

```

```

35043     }
35044
35045     if( !result.getMessage().empty() )
35046         rss << result.getMessage() << '\n';
35047     for( auto const& msg : stats.infoMessages )
35048         if( msg.type == ResultWas::Info )
35049             rss << msg.message << '\n';
35050
35051     rss << "at " << result.getSourceInfo();
35052     xml.writeText( rss.str(), XmlFormatting::Newline );
35053 }
35054 }
35055
35056 CATCH_REGISTER_REPORTER( "junit", JunitReporter )
35057
35058 } // end namespace Catch
35059 // end catch_reporter_junit.cpp
35060 // start catch_reporter_listening.cpp
35061
35062 #include <cassert>
35063
35064 namespace Catch {
35065
35066     ListeningReporter::ListeningReporter() {
35067         // We will assume that listeners will always want all assertions
35068         m_preferences.shouldReportAllAssertions = true;
35069     }
35070
35071     void ListeningReporter::addListener( IStreamingReporterPtr&& listener ) {
35072         m_listeners.push_back( std::move( listener ) );
35073     }
35074
35075     void ListeningReporter::addReporter( IStreamingReporterPtr&& reporter ) {
35076         assert( !m_reporter && "Listening reporter can wrap only 1 real reporter" );
35077         m_reporter = std::move( reporter );
35078         m_preferences.shouldRedirectStdOut = m_reporter->getPreferences().shouldRedirectStdOut;
35079     }
35080
35081     ReporterPreferences ListeningReporter::getPreferences() const {
35082         return m_preferences;
35083     }
35084
35085     std::set<Verbosity> ListeningReporter::getSupportedVerbsities() {
35086         return std::set<Verbosity>{ };
35087     }
35088
35089     void ListeningReporter::noMatchingTestCases( std::string const& spec ) {
35090         for ( auto const& listener : m_listeners ) {
35091             listener->noMatchingTestCases( spec );
35092         }
35093         m_reporter->noMatchingTestCases( spec );
35094     }
35095
35096     void ListeningReporter::reportInvalidArguments( std::string const& arg ) {
35097         for ( auto const& listener : m_listeners ) {
35098             listener->reportInvalidArguments( arg );
35099         }
35100         m_reporter->reportInvalidArguments( arg );
35101     }
35102
35103 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
35104     void ListeningReporter::benchmarkPreparing( std::string const& name ) {
35105         for ( auto const& listener : m_listeners ) {
35106             listener->benchmarkPreparing( name );
35107         }
35108         m_reporter->benchmarkPreparing( name );
35109     }
35110     void ListeningReporter::benchmarkStarting( BenchmarkInfo const& benchmarkInfo ) {
35111         for ( auto const& listener : m_listeners ) {
35112             listener->benchmarkStarting( benchmarkInfo );
35113         }
35114         m_reporter->benchmarkStarting( benchmarkInfo );
35115     }
35116     void ListeningReporter::benchmarkEnded( BenchmarkStats<> const& benchmarkStats ) {
35117         for ( auto const& listener : m_listeners ) {
35118             listener->benchmarkEnded( benchmarkStats );
35119         }
35120         m_reporter->benchmarkEnded( benchmarkStats );
35121     }
35122
35123     void ListeningReporter::benchmarkFailed( std::string const& error ) {
35124         for ( auto const& listener : m_listeners ) {
35125             listener->benchmarkFailed( error );
35126         }
35127         m_reporter->benchmarkFailed( error );
35128     }
35129 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING

```



```

35130
35131 void ListeningReporter::testRunStarting( TestRunInfo const& testRunInfo ) {
35132     for ( auto const& listener : m_listeners ) {
35133         listener->testRunStarting( testRunInfo );
35134     }
35135     m_reporter->testRunStarting( testRunInfo );
35136 }
35137
35138 void ListeningReporter::testGroupStarting( GroupInfo const& groupInfo ) {
35139     for ( auto const& listener : m_listeners ) {
35140         listener->testGroupStarting( groupInfo );
35141     }
35142     m_reporter->testGroupStarting( groupInfo );
35143 }
35144
35145 void ListeningReporter::testCaseStarting( TestCaseInfo const& testInfo ) {
35146     for ( auto const& listener : m_listeners ) {
35147         listener->testCaseStarting( testInfo );
35148     }
35149     m_reporter->testCaseStarting( testInfo );
35150 }
35151
35152 void ListeningReporter::sectionStarting( SectionInfo const& sectionInfo ) {
35153     for ( auto const& listener : m_listeners ) {
35154         listener->sectionStarting( sectionInfo );
35155     }
35156     m_reporter->sectionStarting( sectionInfo );
35157 }
35158
35159 void ListeningReporter::assertionStarting( AssertionInfo const& assertionInfo ) {
35160     for ( auto const& listener : m_listeners ) {
35161         listener->assertionStarting( assertionInfo );
35162     }
35163     m_reporter->assertionStarting( assertionInfo );
35164 }
35165
35166 // The return value indicates if the messages buffer should be cleared:
35167 bool ListeningReporter::assertionEnded( AssertionStats const& assertionStats ) {
35168     for( auto const& listener : m_listeners ) {
35169         static_cast<void>( listener->assertionEnded( assertionStats ) );
35170     }
35171     return m_reporter->assertionEnded( assertionStats );
35172 }
35173
35174 void ListeningReporter::sectionEnded( SectionStats const& sectionStats ) {
35175     for ( auto const& listener : m_listeners ) {
35176         listener->sectionEnded( sectionStats );
35177     }
35178     m_reporter->sectionEnded( sectionStats );
35179 }
35180
35181 void ListeningReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
35182     for ( auto const& listener : m_listeners ) {
35183         listener->testCaseEnded( testCaseStats );
35184     }
35185     m_reporter->testCaseEnded( testCaseStats );
35186 }
35187
35188 void ListeningReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
35189     for ( auto const& listener : m_listeners ) {
35190         listener->testGroupEnded( testGroupStats );
35191     }
35192     m_reporter->testGroupEnded( testGroupStats );
35193 }
35194
35195 void ListeningReporter::testRunEnded( TestRunStats const& testRunStats ) {
35196     for ( auto const& listener : m_listeners ) {
35197         listener->testRunEnded( testRunStats );
35198     }
35199     m_reporter->testRunEnded( testRunStats );
35200 }
35201
35202 void ListeningReporter::skipTest( TestCaseInfo const& testInfo ) {
35203     for ( auto const& listener : m_listeners ) {
35204         listener->skipTest( testInfo );
35205     }
35206     m_reporter->skipTest( testInfo );
35207 }
35208
35209 bool ListeningReporter::isMulti() const {
35210     return true;
35211 }
35212
35213 } // end namespace Catch
35214 // end catch_reporter_listening.cpp
35215 // start catch_reporter_xml.cpp
35216

```

```

35217 #if defined(_MSC_VER)
35218 #pragma warning(push)
35219 #pragma warning(disable:4061) // Not all labels are EXPLICITLY handled in switch
35220                               // Note that 4062 (not all labels are handled
35221                               // and default is missing) is enabled
35222 #endif
35223
35224 namespace Catch {
35225     XmlReporter::XmlReporter( ReporterConfig const& _config )
35226     :   StreamingReporterBase( _config ),
35227         m_xml(_config.stream())
35228     {
35229         m_reporterPrefs.shouldRedirectStdOut = true;
35230         m_reporterPrefs.shouldReportAllAssertions = true;
35231     }
35232
35233     XmlReporter::~XmlReporter() = default;
35234
35235     std::string XmlReporter::getDescription() {
35236         return "Reports test results as an XML document";
35237     }
35238
35239     std::string XmlReporter::getStylesheetRef() const {
35240         return std::string();
35241     }
35242
35243     void XmlReporter::writeSourceInfo( SourceLineInfo const& sourceInfo ) {
35244         m_xml
35245             .writeAttribute( "filename", sourceInfo.file )
35246             .writeAttribute( "line", sourceInfo.line );
35247     }
35248
35249     void XmlReporter::noMatchingTestCases( std::string const& s ) {
35250         StreamingReporterBase::noMatchingTestCases( s );
35251     }
35252
35253     void XmlReporter::testRunStarting( TestRunInfo const& testInfo ) {
35254         StreamingReporterBase::testRunStarting( testInfo );
35255         std::string stylesheetRef = getStylesheetRef();
35256         if( !stylesheetRef.empty() )
35257             m_xml.writeStylesheetRef( stylesheetRef );
35258         m_xml.startElement( "Catch" );
35259         if( !m_config->name().empty() )
35260             m_xml.writeAttribute( "name", m_config->name() );
35261         if( m_config->testSpec().hasFilters() )
35262             m_xml.writeAttribute( "filters", serializeFilters( m_config->getTestsOrTags() ) );
35263         if( m_config->rngSeed() != 0 )
35264             m_xml.scopedElement( "Randomness" )
35265                 .writeAttribute( "seed", m_config->rngSeed() );
35266     }
35267
35268     void XmlReporter::testGroupStarting( GroupInfo const& groupInfo ) {
35269         StreamingReporterBase::testGroupStarting( groupInfo );
35270         m_xml.startElement( "Group" )
35271             .writeAttribute( "name", groupInfo.name );
35272     }
35273
35274     void XmlReporter::testCaseStarting( TestCaseInfo const& testInfo ) {
35275         StreamingReporterBase::testCaseStarting( testInfo );
35276         m_xml.startElement( "TestCase" )
35277             .writeAttribute( "name", trim( testInfo.name ) )
35278             .writeAttribute( "description", testInfo.description )
35279             .writeAttribute( "tags", testInfo.tagsAsString );
35280
35281         writeSourceInfo( testInfo.lineInfo );
35282
35283         if( m_config->showDurations() == ShowDurations::Always )
35284             m_testCaseTimer.start();
35285         m_xml.ensureTagClosed();
35286     }
35287
35288     void XmlReporter::sectionStarting( SectionInfo const& sectionInfo ) {
35289         StreamingReporterBase::sectionStarting( sectionInfo );
35290         if( m_sectionDepth++ > 0 ) {
35291             m_xml.startElement( "Section" )
35292                 .writeAttribute( "name", trim( sectionInfo.name ) );
35293             writeSourceInfo( sectionInfo.lineInfo );
35294             m_xml.ensureTagClosed();
35295         }
35296     }
35297
35298     void XmlReporter::assertionStarting( AssertionInfo const& ) { }
35299
35300     bool XmlReporter::assertionEnded( AssertionStats const& assertionStats ) {
35301         AssertionResult const& result = assertionStats.assertionResult;
35302
35303

```

```

35304         bool includeResults = m_config->includeSuccessfulResults() || !result.isOk();
35305
35306         if( includeResults || result.getResultType() == ResultWas::Warning ) {
35307             // Print any info messages in <Info> tags.
35308             for( auto const& msg : assertionStats.infoMessages ) {
35309                 if( msg.type == ResultWas::Info && includeResults ) {
35310                     m_xml.scopedElement( "Info" )
35311                         .writeText( msg.message );
35312                 } else if ( msg.type == ResultWas::Warning ) {
35313                     m_xml.scopedElement( "Warning" )
35314                         .writeText( msg.message );
35315                 }
35316             }
35317         }
35318
35319         // Drop out if result was successful but we're not printing them.
35320         if( !includeResults && result.getResultType() != ResultWas::Warning )
35321             return true;
35322
35323         // Print the expression if there is one.
35324         if( result.hasExpression() ) {
35325             m_xml.startElement( "Expression" )
35326                 .writeAttribute( "success", result.succeeded() )
35327                 .writeAttribute( "type", result.getTestMacroName() );
35328
35329             writeSourceInfo( result.getSourceInfo() );
35330
35331             m_xml.scopedElement( "Original" )
35332                 .writeText( result.getExpression() );
35333             m_xml.scopedElement( "Expanded" )
35334                 .writeText( result.getExpandedExpression() );
35335         }
35336
35337         // And... Print a result applicable to each result type.
35338         switch( result.getResultType() ) {
35339             case ResultWas::ThrowException:
35340                 m_xml.startElement( "Exception" );
35341                 writeSourceInfo( result.getSourceInfo() );
35342                 m_xml.writeText( result.getMessage() );
35343                 m_xml.endElement();
35344                 break;
35345             case ResultWas::FatalErrorCondition:
35346                 m_xml.startElement( "FatalErrorCondition" );
35347                 writeSourceInfo( result.getSourceInfo() );
35348                 m_xml.writeText( result.getMessage() );
35349                 m_xml.endElement();
35350                 break;
35351             case ResultWas::Info:
35352                 m_xml.scopedElement( "Info" )
35353                     .writeText( result.getMessage() );
35354                 break;
35355             case ResultWas::Warning:
35356                 // Warning will already have been written
35357                 break;
35358             case ResultWas::ExplicitFailure:
35359                 m_xml.startElement( "Failure" );
35360                 writeSourceInfo( result.getSourceInfo() );
35361                 m_xml.writeText( result.getMessage() );
35362                 m_xml.endElement();
35363                 break;
35364             default:
35365                 break;
35366         }
35367
35368         if( result.hasExpression() )
35369             m_xml.endElement();
35370
35371         return true;
35372     }
35373
35374     void XmlReporter::sectionEnded( SectionStats const& sectionStats ) {
35375         StreamingReporterBase::sectionEnded( sectionStats );
35376         if( --m_sectionDepth > 0 ) {
35377             XmlWriter::ScopedElement e = m_xml.scopedElement( "OverallResults" );
35378             e.writeAttribute( "successes", sectionStats.assertions.passed );
35379             e.writeAttribute( "failures", sectionStats.assertions.failed );
35380             e.writeAttribute( "expectedFailures", sectionStats.assertions.failedButOk );
35381
35382             if ( m_config->showDurations() == ShowDurations::Always )
35383                 e.writeAttribute( "durationInSeconds", sectionStats.durationInSeconds );
35384
35385             m_xml.endElement();
35386         }
35387     }
35388
35389     void XmlReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
35390         StreamingReporterBase::testCaseEnded( testCaseStats );

```

```

35391         XmlWriter::ScopedElement e = m_xml.scopedElement( "OverallResult" );
35392         e.writeAttribute( "success", testCaseStats.totals.assertions.allOk() );
35393
35394         if ( m_config->showDurations() == ShowDurations::Always )
35395             e.writeAttribute( "durationInSeconds", m_testCaseTimer.getElapsedSeconds() );
35396
35397         if( !testCaseStats.stdOut.empty() )
35398             m_xml.scopedElement( "StdOut" ).writeText( trim( testCaseStats.stdOut ),
35399 XmlFormatting::Newline );
35400         if( !testCaseStats.stdErr.empty() )
35401             m_xml.scopedElement( "StdErr" ).writeText( trim( testCaseStats.stdErr ),
35402 XmlFormatting::Newline );
35403
35404         m_xml.endElement();
35405     }
35406
35407     void XmlReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
35408         StreamingReporterBase::testGroupEnded( testGroupStats );
35409         // TODO: Check testGroupStats.aborting and act accordingly.
35410         m_xml.scopedElement( "OverallResults" )
35411             .writeAttribute( "successes", testGroupStats.totals.assertions.passed )
35412             .writeAttribute( "failures", testGroupStats.totals.assertions.failed )
35413             .writeAttribute( "expectedFailures", testGroupStats.totals.assertions.failedButOk );
35414         m_xml.scopedElement( "OverallResultsCases" )
35415             .writeAttribute( "successes", testGroupStats.totals.testCases.passed )
35416             .writeAttribute( "failures", testGroupStats.totals.testCases.failed )
35417             .writeAttribute( "expectedFailures", testGroupStats.totals.testCases.failedButOk );
35418         m_xml.endElement();
35419     }
35420
35421     void XmlReporter::testRunEnded( TestRunStats const& testRunStats ) {
35422         StreamingReporterBase::testRunEnded( testRunStats );
35423         m_xml.scopedElement( "OverallResults" )
35424             .writeAttribute( "successes", testRunStats.totals.assertions.passed )
35425             .writeAttribute( "failures", testRunStats.totals.assertions.failed )
35426             .writeAttribute( "expectedFailures", testRunStats.totals.assertions.failedButOk );
35427         m_xml.scopedElement( "OverallResultsCases" )
35428             .writeAttribute( "successes", testRunStats.totals.testCases.passed )
35429             .writeAttribute( "failures", testRunStats.totals.testCases.failed )
35430             .writeAttribute( "expectedFailures", testRunStats.totals.testCases.failedButOk );
35431         m_xml.endElement();
35432     }
35433
35434     #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
35435     void XmlReporter::benchmarkPreparing( std::string const& name ) {
35436         m_xml.startElement( "BenchmarkResults" )
35437             .writeAttribute( "name", name );
35438     }
35439
35440     void XmlReporter::benchmarkStarting( BenchmarkInfo const& info ) {
35441         m_xml.writeAttribute( "samples", info.samples )
35442             .writeAttribute( "resamples", info.resamples )
35443             .writeAttribute( "iterations", info.iterations )
35444             .writeAttribute( "clockResolution", info.clockResolution )
35445             .writeAttribute( "estimatedDuration", info.estimatedDuration )
35446             .writeComment( "All values in nano seconds" );
35447     }
35448
35449     void XmlReporter::benchmarkEnded( BenchmarkStats<> const& benchmarkStats ) {
35450         m_xml.startElement( "mean" )
35451             .writeAttribute( "value", benchmarkStats.mean.point.count() )
35452             .writeAttribute( "lowerBound", benchmarkStats.mean.lower_bound.count() )
35453             .writeAttribute( "upperBound", benchmarkStats.mean.upper_bound.count() )
35454             .writeAttribute( "ci", benchmarkStats.mean.confidence_interval );
35455         m_xml.endElement();
35456         m_xml.startElement( "standardDeviation" )
35457             .writeAttribute( "value", benchmarkStats.standardDeviation.point.count() )
35458             .writeAttribute( "lowerBound", benchmarkStats.standardDeviation.lower_bound.count() )
35459             .writeAttribute( "upperBound", benchmarkStats.standardDeviation.upper_bound.count() )
35460             .writeAttribute( "ci", benchmarkStats.standardDeviation.confidence_interval );
35461         m_xml.endElement();
35462         m_xml.startElement( "outliers" )
35463             .writeAttribute( "variance", benchmarkStats.outlierVariance )
35464             .writeAttribute( "lowMild", benchmarkStats.outliers.low_mild )
35465             .writeAttribute( "lowSevere", benchmarkStats.outliers.low_severe )
35466             .writeAttribute( "highMild", benchmarkStats.outliers.high_mild )
35467             .writeAttribute( "highSevere", benchmarkStats.outliers.high_severe );
35468         m_xml.endElement();
35469     }
35470
35471     void XmlReporter::benchmarkFailed( std::string const& error ) {
35472         m_xml.scopedElement( "failed" )
35473             .writeAttribute( "message", error );
35474         m_xml.endElement();
35475     }
35476
35477 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING

```

```

35476
35477     CATCH_REGISTER_REPORTER( "xml", XmlReporter )
35478
35479 } // end namespace Catch
35480
35481 #if defined(_MSC_VER)
35482 #pragma warning(pop)
35483 #endif
35484 // end catch_reporter_xml.cpp
35485
35486 namespace Catch {
35487     LeakDetector leakDetector;
35488 }
35489
35490 #ifdef __clang__
35491 #pragma clang diagnostic pop
35492 #endif
35493
35494 // end catch_impl.hpp
35495 #endif
35496
35497 #ifdef CATCH_CONFIG_MAIN
35498 // start catch_default_main.hpp
35499
35500 #ifndef __OBJC__
35501
35502 #ifndef CATCH_INTERNAL_CDECL
35503 #define _MSC_VER
35504 #define CATCH_INTERNAL_CDECL __cdecl
35505 #else
35506 #define CATCH_INTERNAL_CDECL
35507 #endif
35508 #endif
35509
35510 #if defined(CATCH_CONFIG_WCHAR) && defined(CATCH_PLATFORM_WINDOWS) && defined(_UNICODE) &&
    !defined(DO_NOT_USE_WMAIN)
35511 // Standard C/C++ Win32 Unicode wmain entry point
35512 extern "C" int CATCH_INTERNAL_CDECL wmain (int argc, wchar_t * argv[], wchar_t * []) {
35513     else
35514 // Standard C/C++ main entry point
35515 int CATCH_INTERNAL_CDECL main (int argc, char * argv[]) {
35516 #endif
35517
35518     return Catch::Session().run( argc, argv );
35519 }
35520
35521 #else // __OBJC__
35522
35523 // Objective-C entry point
35524 int main (int argc, char * const argv[]) {
35525     #if !CATCH_ARC_ENABLED
35526         NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
35527     #endif
35528
35529     Catch::registerTestMethods();
35530     int result = Catch::Session().run( argc, (char**)argv );
35531
35532     #if !CATCH_ARC_ENABLED
35533         [pool drain];
35534     #endif
35535
35536     return result;
35537 }
35538
35539 #endif // __OBJC__
35540
35541 // end catch_default_main.hpp
35542 #endif
35543
35544 #if !defined(CATCH_CONFIG_IMPL_ONLY)
35545
35546 #ifdef CLARA_CONFIG_MAIN_NOT_DEFINED
35547 #undef CLARA_CONFIG_MAIN
35548 #endif
35549
35550 #if !defined(CATCH_CONFIG_DISABLE)
35551 // If this config identifier is defined then all CATCH macros are prefixed with CATCH_
35552 #ifdef CATCH_CONFIG_PREFIX_ALL
35553
35554 #define CATCH_REQUIRE( ... ) INTERNAL_CATCH_TEST( "CATCH_REQUIRE", Catch::ResultDisposition::Normal,
    __VA_ARGS__ )
35555 #define CATCH_REQUIRE_FALSE( ... ) INTERNAL_CATCH_TEST( "CATCH_REQUIRE_FALSE",
    Catch::ResultDisposition::Normal | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
35556 #define CATCH_REQUIRE_THROWS( ... ) INTERNAL_CATCH_TEST( "CATCH_REQUIRE_THROWS",
    Catch::ResultDisposition::Normal, __VA_ARGS__ )
35557 #define CATCH_REQUIRE_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS(

```

```

    "CATCH_REQUIRE_THROWS_AS", exceptionType, Catch::ResultDisposition::Normal, expr )
35560 #define CATCH_REQUIRE_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES(
    "CATCH_REQUIRE_THROWS_WITH", Catch::ResultDisposition::Normal, matcher, expr )
35561 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
35562 #define CATCH_REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
    "CATCH_REQUIRE_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::Normal, matcher, expr )
35563 #endif// CATCH_CONFIG_DISABLE_MATCHERS
35564 #define CATCH_REQUIRE_NOTHROW( ... ) INTERNAL_CATCH_NO_THROW( "CATCH_REQUIRE_NOTHROW",
    Catch::ResultDisposition::Normal, __VA_ARGS__ )
35565
35566 #define CATCH_CHECK( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
35567 #define CATCH_CHECK_FALSE( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK_FALSE",
    Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
35568 #define CATCH_CHECKED_IF( ... ) INTERNAL_CATCH_IF( "CATCH_CHECKED_IF",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
35569 #define CATCH_CHECKED_ELSE( ... ) INTERNAL_CATCH_ELSE( "CATCH_CHECKED_ELSE",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
35570 #define CATCH_CHECK_NOFAIL( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK_NOFAIL",
    Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::SuppressFail, __VA_ARGS__ )
35571
35572 #define CATCH_CHECK_THROWS( ... ) INTERNAL_CATCH_THROWS( "CATCH_CHECK_THROWS",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
35573 #define CATCH_CHECK_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS(
    "CATCH_CHECK_THROWS_AS", exceptionType, Catch::ResultDisposition::ContinueOnFailure, expr )
35574 #define CATCH_CHECK_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES(
    "CATCH_CHECK_THROWS_WITH", Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
35575 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
35576 #define CATCH_CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
    "CATCH_CHECK_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::ContinueOnFailure, matcher,
    expr )
35577 #endif // CATCH_CONFIG_DISABLE_MATCHERS
35578 #define CATCH_CHECK_NOTHROW( ... ) INTERNAL_CATCH_NO_THROW( "CATCH_CHECK_NOTHROW",
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
35579
35580 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
35581 #define CATCH_CHECK_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CATCH_CHECK_THAT", matcher,
    Catch::ResultDisposition::ContinueOnFailure, arg )
35582
35583 #define CATCH_REQUIRE_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CATCH_REQUIRE_THAT", matcher,
    Catch::ResultDisposition::Normal, arg )
35584 #endif // CATCH_CONFIG_DISABLE_MATCHERS
35585
35586 #define CATCH_INFO( msg ) INTERNAL_CATCH_INFO( "CATCH_INFO", msg )
35587 #define CATCH_UNSCOPED_INFO( msg ) INTERNAL_CATCH_UNSCOPED_INFO( "CATCH_UNSCOPED_INFO", msg )
35588 #define CATCH_WARN( msg ) INTERNAL_CATCH_MSG( "CATCH_WARN", Catch::ResultWas::Warning,
    Catch::ResultDisposition::ContinueOnFailure, msg )
35589 #define CATCH_CAPTURE( ... ) INTERNAL_CATCH_CAPTURE( INTERNAL_CATCH_UNIQUE_NAME(capturer),
    "CATCH_CAPTURE", __VA_ARGS__ )
35590
35591 #define CATCH_TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE( __VA_ARGS__ )
35592 #define CATCH_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className,
    __VA_ARGS__ )
35593 #define CATCH_METHOD_AS_TEST_CASE( method, ... ) INTERNAL_CATCH_METHOD_AS_TEST_CASE( method,
    __VA_ARGS__ )
35594 #define CATCH_REGISTER_TEST_CASE( Function, ... ) INTERNAL_CATCH_REGISTER_TESTCASE( Function,
    __VA_ARGS__ )
35595 #define CATCH_SECTION( ... ) INTERNAL_CATCH_SECTION( __VA_ARGS__ )
35596 #define CATCH_DYNAMIC_SECTION( ... ) INTERNAL_CATCH_DYNAMIC_SECTION( __VA_ARGS__ )
35597 #define CATCH_FAIL( ... ) INTERNAL_CATCH_MSG( "CATCH_FAIL", Catch::ResultWas::ExplicitFailure,
    Catch::ResultDisposition::Normal, __VA_ARGS__ )
35598 #define CATCH_FAIL_CHECK( ... ) INTERNAL_CATCH_MSG( "CATCH_FAIL_CHECK",
    Catch::ResultWas::ExplicitFailure, Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
35599 #define CATCH_SUCCEED( ... ) INTERNAL_CATCH_MSG( "CATCH_SUCCEED", Catch::ResultWas::Ok,
    Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
35600
35601 #define CATCH_ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE()
35602
35603 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
35604 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
35605 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ )
35606 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD(
    className, __VA_ARGS__ )
35607 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
    INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
35608 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__
    )
35609 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(
    __VA_ARGS__ )
35610 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... )
    INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ )
35611 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... )
    INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
35612 #else
35613 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
    INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ ) )
35614 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(

```

```

INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ ) )
35615 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
35616 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
35617 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ ) )
35618 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( __VA_ARGS__ ) )
35619 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
35620 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
35621 #endif
35622
35623 #if !defined(CATCH_CONFIG_RUNTIME_STATIC_REQUIRE)
35624 #define CATCH_STATIC_REQUIRE( ... )          static_assert(    __VA_ARGS__ ,      #__VA_ARGS__ );
CATCH_SUCCEED( #__VA_ARGS__ )
35625 #define CATCH_STATIC_REQUIRE_FALSE( ... ) static_assert( !( __VA_ARGS__ ), "!( " #__VA_ARGS__ " )" );
CATCH_SUCCEED( #__VA_ARGS__ )
35626 #else
35627 #define CATCH_STATIC_REQUIRE( ... )          CATCH_REQUIRE( __VA_ARGS__ )
35628 #define CATCH_STATIC_REQUIRE_FALSE( ... ) CATCH_REQUIRE_FALSE( __VA_ARGS__ )
35629 #endif
35630
35631 // "BDD-style" convenience wrappers
35632 #define CATCH_TEST_CASE( ... ) CATCH_TEST_CASE( "Scenario: " __VA_ARGS__ )
35633 #define CATCH_SCENARIO_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, "Scenario:
" __VA_ARGS__ )
35634 #define CATCH_GIVEN( desc )      INTERNAL_CATCH_DYNAMIC_SECTION( "    Given: " « desc )
35635 #define CATCH_AND_GIVEN( desc )  INTERNAL_CATCH_DYNAMIC_SECTION( "And given: " « desc )
35636 #define CATCH_WHEN( desc )       INTERNAL_CATCH_DYNAMIC_SECTION( "    When: " « desc )
35637 #define CATCH_AND_WHEN( desc )   INTERNAL_CATCH_DYNAMIC_SECTION( " And when: " « desc )
35638 #define CATCH_THEN( desc )       INTERNAL_CATCH_DYNAMIC_SECTION( "    Then: " « desc )
35639 #define CATCH_AND_THEN( desc )   INTERNAL_CATCH_DYNAMIC_SECTION( " And: " « desc )
35640
35641 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
35642 #define CATCH_BENCHMARK(...) \
35643     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__))
35644 #define CATCH_BENCHMARK_ADVANCED( name ) \
35645     INTERNAL_CATCH_BENCHMARK_ADVANCED(INTERNAL_CATCH_UNIQUE_NAME(C_A_T_C_H_B_E_N_C_H_), name)
35646 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
35647
35648 // If CATCH_CONFIG_PREFIX_ALL is not defined then the CATCH_ prefix is not required
35649 #else
35650
35651 #define REQUIRE( ... ) INTERNAL_CATCH_TEST( "REQUIRE", Catch::ResultDisposition::Normal, __VA_ARGS__
)
35652 #define REQUIRE_FALSE( ... ) INTERNAL_CATCH_TEST( "REQUIRE_FALSE", Catch::ResultDisposition::Normal |
Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
35653
35654 #define REQUIRE_THROWS( ... ) INTERNAL_CATCH_THROWS( "REQUIRE_THROWS",
Catch::ResultDisposition::Normal, __VA_ARGS__ )
35655 #define REQUIRE_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS( "REQUIRE_THROWS_AS",
exceptionType, Catch::ResultDisposition::Normal, expr )
35656 #define REQUIRE_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "REQUIRE_THROWS_WITH",
Catch::ResultDisposition::Normal, matcher, expr )
35657 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
35658 #define REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
"REQUIRE_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::Normal, matcher, expr )
35659 #endif // CATCH_CONFIG_DISABLE_MATCHERS
35660 #define REQUIRE_NOTHROW( ... ) INTERNAL_CATCH_NO_THROW( "REQUIRE_NOTHROW",
Catch::ResultDisposition::Normal, __VA_ARGS__ )
35661
35662 #define CHECK( ... ) INTERNAL_CATCH_TEST( "CHECK", Catch::ResultDisposition::ContinueOnFailure,
__VA_ARGS__ )
35663 #define CHECK_FALSE( ... ) INTERNAL_CATCH_TEST( "CHECK_FALSE",
Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
35664 #define CHECKED_IF( ... ) INTERNAL_CATCH_IF( "CHECKED_IF",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
35665 #define CHECKED_ELSE( ... ) INTERNAL_CATCH_ELSE( "CHECKED_ELSE",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
35666 #define CHECK_NOFAIL( ... ) INTERNAL_CATCH_TEST( "CHECK_NOFAIL",
Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::SuppressFail, __VA_ARGS__ )
35667
35668 #define CHECK_THROWS( ... ) INTERNAL_CATCH_THROWS( "CHECK_THROWS",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
35669 #define CHECK_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS( "CHECK_THROWS_AS",
exceptionType, Catch::ResultDisposition::ContinueOnFailure, expr )
35670 #define CHECK_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "CHECK_THROWS_WITH",
Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
35671 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
35672 #define CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
"CHECK_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
35673 #endif // CATCH_CONFIG_DISABLE_MATCHERS
35674 #define CHECK_NOTHROW( ... ) INTERNAL_CATCH_NO_THROW( "CHECK_NOTHROW",

```



```

        Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
35675
35676 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
35677 #define CHECK_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CHECK_THAT", matcher,
        Catch::ResultDisposition::ContinueOnFailure, arg )
35678
35679 #define REQUIRE_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "REQUIRE_THAT", matcher,
        Catch::ResultDisposition::Normal, arg )
35680 #endif // CATCH_CONFIG_DISABLE_MATCHERS
35681
35682 #define INFO( msg ) INTERNAL_CATCH_INFO( "INFO", msg )
35683 #define UNSCOPED_INFO( msg ) INTERNAL_CATCH_UNSCOPED_INFO( "UNSCOPED_INFO", msg )
35684 #define WARN( msg ) INTERNAL_CATCH_MSG( "WARN", Catch::ResultWas::Warning,
        Catch::ResultDisposition::ContinueOnFailure, msg )
35685 #define CAPTURE( ... ) INTERNAL_CATCH_CAPTURE( INTERNAL_CATCH_UNIQUE_NAME(capturer),
        "CAPTURE",__VA_ARGS__ )
35686
35687 #define TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE( __VA_ARGS__ )
35688 #define TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, __VA_ARGS__ )
35689 #define METHOD_AS_TEST_CASE( method, ... ) INTERNAL_CATCH_METHOD_AS_TEST_CASE( method, __VA_ARGS__ )
35690 #define REGISTER_TEST_CASE( Function, ... ) INTERNAL_CATCH_REGISTER_TESTCASE( Function, __VA_ARGS__ )
35691 #define SECTION( ... ) INTERNAL_CATCH_SECTION( __VA_ARGS__ )
35692 #define DYNAMIC_SECTION( ... ) INTERNAL_CATCH_DYNAMIC_SECTION( __VA_ARGS__ )
35693 #define FAIL( ... ) INTERNAL_CATCH_MSG( "FAIL", Catch::ResultWas::ExplicitFailure,
        Catch::ResultDisposition::Normal, __VA_ARGS__ )
35694 #define FAIL_CHECK( ... ) INTERNAL_CATCH_MSG( "FAIL_CHECK", Catch::ResultWas::ExplicitFailure,
        Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
35695 #define SUCCEED( ... ) INTERNAL_CATCH_MSG( "SUCCEED", Catch::ResultWas::Ok,
        Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
35696 #define ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE()
35697
35698 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
35699 #define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
35700 #define TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ )
35701 #define TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD(
        className, __VA_ARGS__ )
35702 #define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG(
        className, __VA_ARGS__ )
35703 #define TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ )
35704 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(
        __VA_ARGS__ )
35705 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... )
        INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ )
35706 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... )
        INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
35707 #define TEMPLATE_LIST_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE(__VA_ARGS__)
35708 #define TEMPLATE_LIST_TEST_CASE_METHOD( className, ... )
        INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD( className, __VA_ARGS__ )
35709 #else
35710 #define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE(
        __VA_ARGS__ ) )
35711 #define TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ ) )
35712 #define TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
35713 #define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
35714 #define TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ ) )
35715 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( __VA_ARGS__ ) )
35716 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
35717 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
35718 #define TEMPLATE_LIST_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE( __VA_ARGS__ ) )
35719 #define TEMPLATE_LIST_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
35720 #endif
35721
35722 #if !defined(CATCH_CONFIG_RUNTIME_STATIC_REQUIRE)
35723 #define STATIC_REQUIRE( ... ) static_assert( __VA_ARGS__, #__VA_ARGS__ ); SUCCEED(
        #__VA_ARGS__ )
35724 #define STATIC_REQUIRE_FALSE( ... ) static_assert( !(__VA_ARGS__), "!( " #__VA_ARGS__ " )" ); SUCCEED(
        "!( " #__VA_ARGS__ " )" )
35725 #else
35726 #define STATIC_REQUIRE( ... ) REQUIRE( __VA_ARGS__ )
35727 #define STATIC_REQUIRE_FALSE( ... ) REQUIRE_FALSE( __VA_ARGS__ )
35728 #endif
35729
35730 #endif
35731
35732 #define CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION( signature )
35733
35734 // "BDD-style" convenience wrappers
35735 #define SCENARIO( ... ) TEST_CASE( "Scenario: " __VA_ARGS__ )

```



```

35736 #define SCENARIO_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, "Scenario: "
    __VA_ARGS__ )
35737
35738 #define GIVEN( desc )      INTERNAL_CATCH_DYNAMIC_SECTION( "    Given: " « desc )
35739 #define AND_GIVEN( desc )  INTERNAL_CATCH_DYNAMIC_SECTION( "And given: " « desc )
35740 #define WHEN( desc )       INTERNAL_CATCH_DYNAMIC_SECTION( "    When: " « desc )
35741 #define AND_WHEN( desc )   INTERNAL_CATCH_DYNAMIC_SECTION( " And when: " « desc )
35742 #define THEN( desc )       INTERNAL_CATCH_DYNAMIC_SECTION( "    Then: " « desc )
35743 #define AND_THEN( desc )   INTERNAL_CATCH_DYNAMIC_SECTION( " And: " « desc )
35744
35745 #if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
35746 #define BENCHMARK(...) \
35747     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__))
35748 #define BENCHMARK_ADVANCED(name) \
35749     INTERNAL_CATCH_BENCHMARK_ADVANCED(INTERNAL_CATCH_UNIQUE_NAME(C_A_T_C_H_B_E_N_C_H_), name)
35750 #endif // CATCH_CONFIG_ENABLE_BENCHMARKING
35751
35752 using Catch::Detail::Approx;
35753
35754 #else // CATCH_CONFIG_DISABLE
35755
35756 // If this config identifier is defined then all CATCH macros are prefixed with CATCH_
35757 #ifdef CATCH_CONFIG_PREFIX_ALL
35758
35759 #define CATCH_REQUIRE( ... )           (void)(0)
35760 #define CATCH_REQUIRE_FALSE( ... )     (void)(0)
35761
35762 #define CATCH_REQUIRE_THROWS( ... )    (void)(0)
35763 #define CATCH_REQUIRE_THROWS_AS( expr, exceptionType ) (void)(0)
35764 #define CATCH_REQUIRE_THROWS_WITH( expr, matcher )      (void)(0)
35765 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
35766 #define CATCH_REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
35767 #endif // CATCH_CONFIG_DISABLE_MATCHERS
35768 #define CATCH_REQUIRE_NO_THROW( ... ) (void)(0)
35769
35770 #define CATCH_CHECK( ... )             (void)(0)
35771 #define CATCH_CHECK_FALSE( ... )       (void)(0)
35772 #define CATCH_CHECKED_IF( ... )        if ( __VA_ARGS__ )
35773 #define CATCH_CHECKED_ELSE( ... )      if ( !(__VA_ARGS__) )
35774 #define CATCH_CHECK_NOFAIL( ... )      (void)(0)
35775
35776 #define CATCH_CHECK_THROWS( ... )      (void)(0)
35777 #define CATCH_CHECK_THROWS_AS( expr, exceptionType ) (void)(0)
35778 #define CATCH_CHECK_THROWS_WITH( expr, matcher )      (void)(0)
35779 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
35780 #define CATCH_CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
35781 #endif // CATCH_CONFIG_DISABLE_MATCHERS
35782 #define CATCH_CHECK_NO_THROW( ... )     (void)(0)
35783
35784 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
35785 #define CATCH_CHECK_THAT( arg, matcher ) (void)(0)
35786
35787 #define CATCH_REQUIRE_THAT( arg, matcher ) (void)(0)
35788 #endif // CATCH_CONFIG_DISABLE_MATCHERS
35789
35790 #define CATCH_INFO( msg )               (void)(0)
35791 #define CATCH_UNSCOPED_INFO( msg )      (void)(0)
35792 #define CATCH_WARN( msg )               (void)(0)
35793 #define CATCH_CAPTURE( msg )            (void)(0)
35794
35795 #define CATCH_TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
    C_A_T_C_H_T_E_S_T_ ))
35796 #define CATCH_TEST_CASE_METHOD( className, ... )
    INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ))
35797 #define CATCH_METHOD_AS_TEST_CASE( method, ... )
35798 #define CATCH_REGISTER_TEST_CASE( Function, ... ) (void)(0)
35799 #define CATCH_SECTION( ... )
35800 #define CATCH_DYNAMIC_SECTION( ... )
35801 #define CATCH_FAIL( ... ) (void)(0)
35802 #define CATCH_FAIL_CHECK( ... ) (void)(0)
35803 #define CATCH_SUCCEED( ... ) (void)(0)
35804
35805 #define CATCH_ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
    C_A_T_C_H_T_E_S_T_ ))
35806
35807 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
35808 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION( __VA_ARGS__ )
35809 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... )
    INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION( __VA_ARGS__ )
35810 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... )
    INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION( className, __VA_ARGS__ )
35811 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
    INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION( className, __VA_ARGS__ )
35812 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
35813 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
35814 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(

```

```

        className, __VA_ARGS__ )
35816 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
        className, __VA_ARGS__ )
35817 #else
35818 #define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__) )
35819 #define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__) )
35820 #define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__ ) )
35821 #define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ ) )
35822 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
35823 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
35824 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
        className, __VA_ARGS__ )
35825 #define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
        className, __VA_ARGS__ )
35826 #endif
35827
35828 // "BDD-style" convenience wrappers
35829 #define CATCH_SCENARIO( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
        C_A_T_C_H_T_E_S_T_ ))
35830 #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 )
35831 #define CATCH_GIVEN( desc )
35832 #define CATCH_AND_GIVEN( desc )
35833 #define CATCH_WHEN( desc )
35834 #define CATCH_AND_WHEN( desc )
35835 #define CATCH_THEN( desc )
35836 #define CATCH_AND_THEN( desc )
35837
35838 #define CATCH_STATIC_REQUIRE( ... ) (void)(0)
35839 #define CATCH_STATIC_REQUIRE_FALSE( ... ) (void)(0)
35840
35841 // If CATCH_CONFIG_PREFIX_ALL is not defined then the CATCH_ prefix is not required
35842 #else
35843
35844 #define REQUIRE( ... ) (void)(0)
35845 #define REQUIRE_FALSE( ... ) (void)(0)
35846
35847 #define REQUIRE_THROWS( ... ) (void)(0)
35848 #define REQUIRE_THROWS_AS( expr, exceptionType ) (void)(0)
35849 #define REQUIRE_THROWS_WITH( expr, matcher ) (void)(0)
35850 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
35851 #define REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
35852 #endif // CATCH_CONFIG_DISABLE_MATCHERS
35853 #define REQUIRE_NOTHROW( ... ) (void)(0)
35854
35855 #define CHECK( ... ) (void)(0)
35856 #define CHECK_FALSE( ... ) (void)(0)
35857 #define CHECKED_IF( ... ) if (__VA_ARGS__)
35858 #define CHECKED_ELSE( ... ) if (!(__VA_ARGS__))
35859 #define CHECK_NOFAIL( ... ) (void)(0)
35860
35861 #define CHECK_THROWS( ... ) (void)(0)
35862 #define CHECK_THROWS_AS( expr, exceptionType ) (void)(0)
35863 #define CHECK_THROWS_WITH( expr, matcher ) (void)(0)
35864 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
35865 #define CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
35866 #endif // CATCH_CONFIG_DISABLE_MATCHERS
35867 #define CHECK_NOTHROW( ... ) (void)(0)
35868
35869 #if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
35870 #define CHECK_THAT( arg, matcher ) (void)(0)
35871
35872 #define REQUIRE_THAT( arg, matcher ) (void)(0)
35873 #endif // CATCH_CONFIG_DISABLE_MATCHERS
35874
35875 #define INFO( msg ) (void)(0)
35876 #define UNSCOPED_INFO( msg ) (void)(0)
35877 #define WARN( msg ) (void)(0)
35878 #define CAPTURE( ... ) (void)(0)
35879
35880 #define TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
        C_A_T_C_H_T_E_S_T_ ))
35881 #define TEST_CASE_METHOD( className, ... )
        INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ))
35882 #define METHOD_AS_TEST_CASE( method, ... )
35883 #define REGISTER_TEST_CASE( Function, ... ) (void)(0)
35884 #define SECTION( ... )
35885 #define DYNAMIC_SECTION( ... )
35886 #define FAIL( ... ) (void)(0)
35887 #define FAIL_CHECK( ... ) (void)(0)
35888 #define SUCCEED( ... ) (void)(0)
35889 #define ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(

```

```

    C_A_T_C_H_T_E_S_T_ ))
35890
35891 #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
35892 #define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__)
35893 #define TEMPLATE_TEST_CASE_SIG( ... )
35894     INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__)
35895 #define TEMPLATE_TEST_CASE_METHOD( className, ... )
35896     INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__)
35897 #define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
35898     INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ )
35899 #define TEMPLATE_PRODUCT_TEST_CASE( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
35900 #define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
35901 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
35902     __VA_ARGS__ )
35903 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
35904     __VA_ARGS__ )
35905 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
35906     __VA_ARGS__ )
35907 #define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
35908     __VA_ARGS__ )
35909 #endif
35910
35911 #define STATIC_REQUIRE( ... ) (void)(0)
35912 #define STATIC_REQUIRE_FALSE( ... ) (void)(0)
35913
35914 #endif
35915
35916 #define CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION_NO_REG(
35917     INTERNAL_CATCH_UNIQUE_NAME( catch_internal_ExceptionTranslator ), signature )
35918
35919 // "BDD-style" convenience wrappers
35920 #define SCENARIO( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
35921     C_A_T_C_H_T_E_S_T_ ))
35922 #define SCENARIO_METHOD( className, ... )
35923     INTERNAL_CATCH_TESTCASE_METHOD_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ),
35924     className )
35925
35926 #define GIVEN( desc )
35927 #define AND_GIVEN( desc )
35928 #define WHEN( desc )
35929 #define AND_WHEN( desc )
35930 #define THEN( desc )
35931 #define AND_THEN( desc )
35932
35933 using Catch::Detail::Approx;
35934
35935 #endif
35936
35937 #endif // ! CATCH_CONFIG_IMPL_ONLY
35938
35939 // start catch_reenable_warnings.h
35940
35941 #ifdef __clang__
35942 #   ifdef __ICC // icpc defines the __clang__ macro
35943 #       pragma warning(pop)
35944 #   else
35945 #       pragma clang diagnostic pop
35946 #   endif
35947 #elif defined __GNUC__
35948 #   pragma GCC diagnostic pop
35949 #endif
35950
35951 // end catch_reenable_warnings.h
35952 // end catch.hpp
35953 #endif // TWOBLUECUBES_SINGLE_INCLUDE_CATCH_HPP_INCLUDED

```

7.3 /Users/samanthapope/msdscriptRepo/msdScript/cmake-build-debug/CMakeFiles/3.27.8/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- `#define __has_include(x) 0`
- `#define COMPILER_ID ""`
- `#define STRINGIFY_HELPER(X) #X`
- `#define STRINGIFY(X) STRINGIFY_HELPER(X)`
- `#define PLATFORM_ID`
- `#define ARCHITECTURE_ID`
- `#define DEC(n)`
- `#define HEX(n)`
- `#define C_VERSION`

Functions

- `int main (int argc, char *argv[])`

Variables

- `char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"`
- `char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"`
- `char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"`
- `const char * info_language_standard_default`
- `const char * info_language_extensions_default`

7.3.1 Macro Definition Documentation

7.3.1.1 __has_include

```
#define __has_include(  
    x ) 0
```

Definition at line 17 of file [CMakeCCompilerId.c](#).

7.3.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 716 of file [CMakeCCompilerId.c](#).

7.3.1.3 C_VERSION

```
#define C_VERSION
```

Definition at line 805 of file [CMakeCCompilerId.c](#).

7.3.1.4 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 427 of file [CMakeCCompilerId.c](#).

7.3.1.5 DEC

```
#define DEC(  
    n )
```

Value:

```
('0' + (((n) / 10000000) % 10)), \  
( '0' + (((n) / 1000000) % 10)), \  
( '0' + (((n) / 100000) % 10)), \  
( '0' + (((n) / 10000) % 10)), \  
( '0' + (((n) / 1000) % 10)), \  
( '0' + (((n) / 100) % 10)), \  
( '0' + (((n) / 10) % 10)), \  
( '0' + ((n) % 10))
```

Definition at line 720 of file [CMakeCCompilerId.c](#).

7.3.1.6 HEX

```
#define HEX(  
    n )
```

Value:

```
('0' + ((n) >> 28 & 0xF)), \  
( '0' + ((n) >> 24 & 0xF)), \  
( '0' + ((n) >> 20 & 0xF)), \  
( '0' + ((n) >> 16 & 0xF)), \  
( '0' + ((n) >> 12 & 0xF)), \  
( '0' + ((n) >> 8 & 0xF)), \  
( '0' + ((n) >> 4 & 0xF)), \  
( '0' + ((n) & 0xF))
```

Definition at line 731 of file [CMakeCCompilerId.c](#).

7.3.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 558 of file [CMakeCCompilerId.c](#).

7.3.1.8 STRINGIFY

```
#define STRINGIFY(  
    X ) STRINGIFY\_HELPER(X)
```

Definition at line 448 of file [CMakeCCompilerId.c](#).

7.3.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(  
    X ) #X
```

Definition at line 447 of file [CMakeCCompilerId.c](#).

7.3.2 Function Documentation

7.3.2.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

Definition at line 839 of file [CMakeCCompilerId.c](#).

7.3.3 Variable Documentation

7.3.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 797 of file [CMakeCCompilerId.c](#).

7.3.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 434 of file [CMakeCCompilerId.c](#).

7.3.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
```

```
    "OFF"  
    "]"
```

Definition at line 821 of file [CMakeCCompilerId.c](#).

7.3.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
=
  "INFO" ":" "standard_default[" C_VERSION "]"
```

Definition at line 818 of file [CMakeCCompilerId.c](#).

7.3.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 796 of file [CMakeCCompilerId.c](#).

7.4 CMakeCCompilerId.c

[Go to the documentation of this file.](#)

```
00001 #ifdef __cplusplus
00002 # error "A C++ compiler has been selected for C."
00003 #endif
00004
00005 #if defined(__18CXX)
00006 # define ID_VOID_MAIN
00007 #endif
00008 #if defined(__CLASSIC_C__)
00009 /* cv-qualifiers did not exist in K&R C */
00010 # define const
00011 # define volatile
00012 #endif
00013
00014 #if !defined(__has_include)
00015 /* If the compiler does not have __has_include, pretend the answer is
00016  always no. */
00017 # define __has_include(x) 0
00018 #endif
00019
00020
00021 /* Version number components: V=Version, R=Revision, P=Patch
00022  Version date components: YYYY=Year, MM=Month, DD=Day */
00023
00024 #if defined(__INTEL_COMPILER) || defined(__ICC)
00025 # define COMPILER_ID "Intel"
00026 # if defined(_MSC_VER)
00027 # define SIMULATE_ID "MSVC"
00028 # endif
00029 # if defined(__GNUC__)
00030 # define SIMULATE_ID "GNU"
00031 # endif
00032 /* __INTEL_COMPILER = VRP prior to 2021, and then VVVV for 2021 and later,
00033  except that a few beta releases use the old format with V=2021. */
00034 # if __INTEL_COMPILER < 2021 || __INTEL_COMPILER == 202110 || __INTEL_COMPILER == 202111
00035 # define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER/100)
00036 # define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER/10 % 10)
00037 # if defined(__INTEL_COMPILER_UPDATE)
00038 # define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER_UPDATE)
00039 # else
00040 # define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER % 10)
00041 # endif
00042 # else
00043 # define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER)
00044 # define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER_UPDATE)
00045 /* The third version component from --version is an update index,
00046  but no macro is provided for it. */
00047 # define COMPILER_VERSION_PATCH DEC(0)
00048 # endif
00049 # if defined(__INTEL_COMPILER_BUILD_DATE)
00050 /* __INTEL_COMPILER_BUILD_DATE = YYYYMMDD */
00051 # define COMPILER_VERSION_TWEAK DEC(__INTEL_COMPILER_BUILD_DATE)
00052 # endif
00053 # if defined(_MSC_VER)
```

```

00054      /* _MSC_VER = VVRR */
00055 #   define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00056 #   define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00057 #   endif
00058 #   if defined(__GNUC__)
00059 #       define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00060 #   elif defined(__GNUG__)
00061 #       define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00062 #   endif
00063 #   if defined(__GNUC_MINOR__)
00064 #       define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00065 #   endif
00066 #   if defined(__GNUC_PATCHLEVEL__)
00067 #       define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00068 #   endif
00069
00070 #elif (defined(__clang__) && defined(__INTEL_CLANG_COMPILER)) || defined(__INTEL_LLVM_COMPILER)
00071 #   define COMPILER_ID "IntelLLVM"
00072 #if defined(_MSC_VER)
00073 #   define SIMULATE_ID "MSVC"
00074 #endif
00075 #if defined(__GNUC__)
00076 #   define SIMULATE_ID "GNU"
00077 #endif
00078 /* __INTEL_LLVM_COMPILER = VVVVRP prior to 2021.2.0, VVVVRRPP for 2021.2.0 and
00079  * later. Look for 6 digit vs. 8 digit version number to decide encoding.
00080  * VVVV is no smaller than the current year when a version is released.
00081  */
00082 #if __INTEL_LLVM_COMPILER < 1000000L
00083 #   define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/100)
00084 #   define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/10 % 10)
00085 #   define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER % 10)
00086 #else
00087 #   define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/10000)
00088 #   define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/100 % 100)
00089 #   define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER % 100)
00090 #endif
00091 #if defined(_MSC_VER)
00092     /* _MSC_VER = VVRR */
00093 #   define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00094 #   define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00095 #   endif
00096 #   if defined(__GNUC__)
00097 #       define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00098 #   elif defined(__GNUG__)
00099 #       define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00100 #   endif
00101 #   if defined(__GNUC_MINOR__)
00102 #       define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00103 #   endif
00104 #   if defined(__GNUC_PATCHLEVEL__)
00105 #       define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00106 #   endif
00107
00108 #elif defined(__PATHCC__)
00109 #   define COMPILER_ID "PathScale"
00110 #   define COMPILER_VERSION_MAJOR DEC(__PATHCC__)
00111 #   define COMPILER_VERSION_MINOR DEC(__PATHCC_MINOR__)
00112 #   if defined(__PATHCC_PATCHLEVEL__)
00113 #       define COMPILER_VERSION_PATCH DEC(__PATHCC_PATCHLEVEL__)
00114 #   endif
00115
00116 #elif defined(__BORLANDC__) && defined(__CODEGEARC_VERSION__)
00117 #   define COMPILER_ID "Embarcadero"
00118 #   define COMPILER_VERSION_MAJOR HEX(__CODEGEARC_VERSION__>24 & 0x00FF)
00119 #   define COMPILER_VERSION_MINOR HEX(__CODEGEARC_VERSION__>16 & 0x00FF)
00120 #   define COMPILER_VERSION_PATCH DEC(__CODEGEARC_VERSION__ & 0xFFFF)
00121
00122 #elif defined(__BORLANDC__)
00123 #   define COMPILER_ID "Borland"
00124     /* __BORLANDC__ = 0xVRR */
00125 #   define COMPILER_VERSION_MAJOR HEX(__BORLANDC__>8)
00126 #   define COMPILER_VERSION_MINOR HEX(__BORLANDC__ & 0xFF)
00127
00128 #elif defined(__WATCOMC__) && __WATCOMC__ < 1200
00129 #   define COMPILER_ID "Watcom"
00130     /* __WATCOMC__ = VVRR */
00131 #   define COMPILER_VERSION_MAJOR DEC(__WATCOMC__ / 100)
00132 #   define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00133 #   if (__WATCOMC__ % 10) > 0
00134 #       define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00135 #   endif
00136
00137 #elif defined(__WATCOMC__)
00138 #   define COMPILER_ID "OpenWatcom"
00139     /* __WATCOMC__ = VVRP + 1100 */
00140 #   define COMPILER_VERSION_MAJOR DEC((__WATCOMC__ - 1100) / 100)

```



```

00141 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00142 # if (__WATCOMC__ % 10) > 0
00143 #   define COMPILER_VERSION_PATCH DEC((__WATCOMC__ % 10)
00144 # endif
00145
00146 #elif defined(__SUNPRO_C)
00147 # define COMPILER_ID "SunPro"
00148 # if __SUNPRO_C >= 0x5100
00149   /* __SUNPRO_C = 0xVRRP */
00150 #   define COMPILER_VERSION_MAJOR HEX(__SUNPRO_C>>12)
00151 #   define COMPILER_VERSION_MINOR HEX(__SUNPRO_C>>4 & 0xFF)
00152 #   define COMPILER_VERSION_PATCH HEX(__SUNPRO_C    & 0xF)
00153 # else
00154   /* __SUNPRO_CC = 0xVRP */
00155 #   define COMPILER_VERSION_MAJOR HEX(__SUNPRO_C>>8)
00156 #   define COMPILER_VERSION_MINOR HEX(__SUNPRO_C>>4 & 0xF)
00157 #   define COMPILER_VERSION_PATCH HEX(__SUNPRO_C    & 0xF)
00158 # endif
00159
00160 #elif defined(__HP_cc)
00161 # define COMPILER_ID "HP"
00162   /* __HP_cc = VVRPP */
00163 #   define COMPILER_VERSION_MAJOR DEC(__HP_cc/10000)
00164 #   define COMPILER_VERSION_MINOR DEC(__HP_cc/100 % 100)
00165 #   define COMPILER_VERSION_PATCH DEC(__HP_cc    % 100)
00166
00167 #elif defined(__DECC)
00168 # define COMPILER_ID "Compaq"
00169   /* __DECC_VER = VVVRTPPPP */
00170 #   define COMPILER_VERSION_MAJOR DEC(__DECC_VER/10000000)
00171 #   define COMPILER_VERSION_MINOR DEC(__DECC_VER/100000 % 100)
00172 #   define COMPILER_VERSION_PATCH DEC(__DECC_VER    % 10000)
00173
00174 #elif defined(__IBMC__) && defined(__COMPILER_VER__)
00175 # define COMPILER_ID "zOS"
00176   /* __IBMC__ = VRP */
00177 #   define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00178 #   define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00179 #   define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00180
00181 #elif defined(__open_xl__) && defined(__clang__)
00182 # define COMPILER_ID "IBMclang"
00183 #   define COMPILER_VERSION_MAJOR DEC(__open_xl_version__)
00184 #   define COMPILER_VERSION_MINOR DEC(__open_xl_release__)
00185 #   define COMPILER_VERSION_PATCH DEC(__open_xl_modification__)
00186 #   define COMPILER_VERSION_TWEAK DEC(__open_xl_ptf_fix_level__)
00187
00188
00189 #elif defined(__ibmxl__) && defined(__clang__)
00190 # define COMPILER_ID "XLclang"
00191 #   define COMPILER_VERSION_MAJOR DEC(__ibmxl_version__)
00192 #   define COMPILER_VERSION_MINOR DEC(__ibmxl_release__)
00193 #   define COMPILER_VERSION_PATCH DEC(__ibmxl_modification__)
00194 #   define COMPILER_VERSION_TWEAK DEC(__ibmxl_ptf_fix_level__)
00195
00196
00197 #elif defined(__IBMC__) && !defined(__COMPILER_VER__) && __IBMC__ >= 800
00198 # define COMPILER_ID "XL"
00199   /* __IBMC__ = VRP */
00200 #   define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00201 #   define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00202 #   define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00203
00204 #elif defined(__IBMC__) && !defined(__COMPILER_VER__) && __IBMC__ < 800
00205 # define COMPILER_ID "VisualAge"
00206   /* __IBMC__ = VRP */
00207 #   define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00208 #   define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00209 #   define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00210
00211 #elif defined(__NVCOMPILER)
00212 # define COMPILER_ID "NVHPC"
00213 #   define COMPILER_VERSION_MAJOR DEC(__NVCOMPILER_MAJOR__)
00214 #   define COMPILER_VERSION_MINOR DEC(__NVCOMPILER_MINOR__)
00215 #   if defined(__NVCOMPILER_PATCHLEVEL__)
00216 #     define COMPILER_VERSION_PATCH DEC(__NVCOMPILER_PATCHLEVEL__)
00217 #   endif
00218
00219 #elif defined(__PGI)
00220 # define COMPILER_ID "PGI"
00221 #   define COMPILER_VERSION_MAJOR DEC(__PGIC__)
00222 #   define COMPILER_VERSION_MINOR DEC(__PGIC_MINOR__)
00223 #   if defined(__PGIC_PATCHLEVEL__)
00224 #     define COMPILER_VERSION_PATCH DEC(__PGIC_PATCHLEVEL__)
00225 #   endif
00226
00227 #elif defined(__CRAYC)

```

```

00228 # define COMPILER_ID "Cray"
00229 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
00230 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00231
00232 #elif defined(__TI_COMPILER_VERSION__)
00233 # define COMPILER_ID "TI"
00234 /* __TI_COMPILER_VERSION__ = VVRRRRPPP */
00235 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00236 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000 % 1000)
00237 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__ % 1000)
00238
00239 #elif defined(__CLANG_FUJITSU)
00240 # define COMPILER_ID "FujitsuClang"
00241 # define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00242 # define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00243 # define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00244 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00245
00246
00247 #elif defined(__FUJITSU)
00248 # define COMPILER_ID "Fujitsu"
00249 # if defined(__FCC_version__)
00250 #   define COMPILER_VERSION __FCC_version__
00251 # elif defined(__FCC_major__)
00252 #   define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00253 #   define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00254 #   define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00255 # endif
00256 # if defined(__fcc_version)
00257 #   define COMPILER_VERSION_INTERNAL DEC(__fcc_version)
00258 # elif defined(__FCC_VERSION)
00259 #   define COMPILER_VERSION_INTERNAL DEC(__FCC_VERSION)
00260 # endif
00261
00262
00263 #elif defined(__ghs__)
00264 # define COMPILER_ID "GHS"
00265 /* __GHS_VERSION_NUMBER = VVVVRP */
00266 # ifdef __GHS_VERSION_NUMBER
00267 #   define COMPILER_VERSION_MAJOR DEC(__GHS_VERSION_NUMBER / 100)
00268 #   define COMPILER_VERSION_MINOR DEC(__GHS_VERSION_NUMBER / 10 % 10)
00269 #   define COMPILER_VERSION_PATCH DEC(__GHS_VERSION_NUMBER % 10)
00270 # endif
00271
00272 #elif defined(__TASKING__)
00273 # define COMPILER_ID "Tasking"
00274 #   define COMPILER_VERSION_MAJOR DEC(__VERSION__/1000)
00275 #   define COMPILER_VERSION_MINOR DEC(__VERSION__ % 100)
00276 #   define COMPILER_VERSION_INTERNAL DEC(__VERSION__)
00277
00278 #elif defined(__TINYC__)
00279 # define COMPILER_ID "TinyCC"
00280
00281 #elif defined(__BCC__)
00282 # define COMPILER_ID "Bruce"
00283
00284 #elif defined(__SCO_VERSION__)
00285 # define COMPILER_ID "SCO"
00286
00287 #elif defined(__ARMCC_VERSION) && !defined(__clang__)
00288 # define COMPILER_ID "ARMCC"
00289 #if __ARMCC_VERSION >= 1000000
00290 /* __ARMCC_VERSION = VRRPPPP */
00291 #   define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/1000000)
00292 #   define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 100)
00293 #   define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION % 10000)
00294 #else
00295 /* __ARMCC_VERSION = VRPPPP */
00296 #   define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/100000)
00297 #   define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 10)
00298 #   define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION % 10000)
00299 #endif
00300 #endif
00301
00302 #elif defined(__clang__) && defined(__apple_build_version__)
00303 # define COMPILER_ID "AppleClang"
00304 # if defined(_MSC_VER)
00305 #   define SIMULATE_ID "MSVC"
00306 # endif
00307 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00308 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00309 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00310 # if defined(_MSC_VER)
00311 /* _MSC_VER = VVRR */
00312 #   define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00313 #   define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00314 # endif

```

```

00315 # define COMPILER_VERSION_TWEAK DEC(__apple_build_version__)
00316
00317 #elif defined(__clang__) && defined(__ARMCOMPILER_VERSION)
00318 # define COMPILER_ID "ARMClang"
00319 # define COMPILER_VERSION_MAJOR DEC(__ARMCOMPILER_VERSION/1000000)
00320 # define COMPILER_VERSION_MINOR DEC(__ARMCOMPILER_VERSION/10000 % 100)
00321 # define COMPILER_VERSION_PATCH DEC(__ARMCOMPILER_VERSION/100 % 100)
00322 # define COMPILER_VERSION_INTERNAL DEC(__ARMCOMPILER_VERSION)
00323
00324 #elif defined(__clang__)
00325 # define COMPILER_ID "Clang"
00326 # if defined(_MSC_VER)
00327 #   define SIMULATE_ID "MSVC"
00328 # endif
00329 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00330 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00331 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00332 # if defined(_MSC_VER)
00333   /* _MSC_VER = VVRR */
00334 #   define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00335 #   define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00336 # endif
00337
00338 #elif defined(__LCC__) && (defined(__GNUC__) || defined(__GNUG__) || defined(__MCST__))
00339 # define COMPILER_ID "LCC"
00340 # define COMPILER_VERSION_MAJOR DEC(__LCC__ / 100)
00341 # define COMPILER_VERSION_MINOR DEC(__LCC__ % 100)
00342 # if defined(__LCC_MINOR__)
00343 #   define COMPILER_VERSION_PATCH DEC(__LCC_MINOR__)
00344 # endif
00345 # if defined(__GNUC__) && defined(__GNUC_MINOR__)
00346 #   define SIMULATE_ID "GNU"
00347 #   define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00348 #   define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00349 #   if defined(__GNUC_PATCHLEVEL__)
00350 #     define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00351 #   endif
00352 # endif
00353
00354 #elif defined(__GNUC__)
00355 # define COMPILER_ID "GNU"
00356 # define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00357 # if defined(__GNUC_MINOR__)
00358 #   define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00359 # endif
00360 # if defined(__GNUC_PATCHLEVEL__)
00361 #   define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00362 # endif
00363
00364 #elif defined(_MSC_VER)
00365 # define COMPILER_ID "MSVC"
00366 /* _MSC_VER = VVRR */
00367 # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00368 # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00369 # if defined(_MSC_FULL_VER)
00370 #   if _MSC_VER >= 1400
00371     /* _MSC_FULL_VER = VVRRPPPP */
00372 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00373 #   else
00374     /* _MSC_FULL_VER = VVRRPPPP */
00375 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00376 #   endif
00377 # endif
00378 # if defined(_MSC_BUILD)
00379 #   define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
00380 # endif
00381
00382 #elif defined(_ADI_COMPILER)
00383 # define COMPILER_ID "ADSP"
00384 #if defined(__VERSIONNUM__)
00385   /* __VERSIONNUM__ = 0xVVRRPPTT */
00386 #   define COMPILER_VERSION_MAJOR DEC(__VERSIONNUM__ >> 24 & 0xFF)
00387 #   define COMPILER_VERSION_MINOR DEC(__VERSIONNUM__ >> 16 & 0xFF)
00388 #   define COMPILER_VERSION_PATCH DEC(__VERSIONNUM__ >> 8 & 0xFF)
00389 #   define COMPILER_VERSION_TWEAK DEC(__VERSIONNUM__ & 0xFF)
00390 #endif
00391
00392 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00393 # define COMPILER_ID "IAR"
00394 # if defined(__VER__) && defined(__ICCARM__)
00395 #   define COMPILER_VERSION_MAJOR DEC((__VER__) / 1000000)
00396 #   define COMPILER_VERSION_MINOR DEC(((__VER__) / 1000) % 1000)
00397 #   define COMPILER_VERSION_PATCH DEC((__VER__) % 1000)
00398 #   define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00399 # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRISCV__) || defined(__ICCV850__) ||
defined(__ICC8051__) || defined(__ICCSTM8__))

```

```

00400 # define COMPILER_VERSION_MAJOR DEC((__VER__) / 100)
00401 # define COMPILER_VERSION_MINOR DEC((__VER__) - (((__VER__) / 100)*100))
00402 # define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00403 # define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00404 # endif
00405
00406 #elif defined(__SDCC_VERSION_MAJOR) || defined(SDCC)
00407 # define COMPILER_ID "SDCC"
00408 # if defined(__SDCC_VERSION_MAJOR)
00409 #   define COMPILER_VERSION_MAJOR DEC(__SDCC_VERSION_MAJOR)
00410 #   define COMPILER_VERSION_MINOR DEC(__SDCC_VERSION_MINOR)
00411 #   define COMPILER_VERSION_PATCH DEC(__SDCC_VERSION_PATCH)
00412 # else
00413 /* SDCC = VRP */
00414 #   define COMPILER_VERSION_MAJOR DEC(SDCC/100)
00415 #   define COMPILER_VERSION_MINOR DEC(SDCC/10 % 10)
00416 #   define COMPILER_VERSION_PATCH DEC(SDCC % 10)
00417 # endif
00418
00419
00420 /* These compilers are either not known or too old to define an
00421 identification macro. Try to identify the platform and guess that
00422 it is the native compiler. */
00423 #elif defined(__hpux) || defined(__hpua)
00424 # define COMPILER_ID "HP"
00425
00426 #else /* unknown compiler */
00427 # define COMPILER_ID ""
00428 #endif
00429
00430 /* Construct the string literal in pieces to prevent the source from
00431 getting matched. Store it in a pointer rather than an array
00432 because some compilers will just produce instructions to fill the
00433 array rather than assigning a pointer to a static array. */
00434 char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]";
00435 #ifndef SIMULATE_ID
00436 char const* info_simulate = "INFO" ":" "simulate[" SIMULATE_ID "]";
00437 #endif
00438
00439 #ifdef __QNXNTO__
00440 char const* qnxnto = "INFO" ":" "qnxnto[]";
00441 #endif
00442
00443 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00444 char const* info_cray = "INFO" ":" "compiler_wrapper[CrayPrgEnv]";
00445 #endif
00446
00447 #define STRINGIFY_HELPER(X) #X
00448 #define STRINGIFY(X) STRINGIFY_HELPER(X)
00449
00450 /* Identify known platforms by name. */
00451 #if defined(__linux) || defined(__linux__) || defined(linux)
00452 # define PLATFORM_ID "Linux"
00453
00454 #elif defined(__MSYS__)
00455 # define PLATFORM_ID "MSYS"
00456
00457 #elif defined(__CYGWIN__)
00458 # define PLATFORM_ID "Cygwin"
00459
00460 #elif defined(__MINGW32__)
00461 # define PLATFORM_ID "MinGW"
00462
00463 #elif defined(__APPLE__)
00464 # define PLATFORM_ID "Darwin"
00465
00466 #elif defined(__WIN32) || defined(__WIN32__) || defined(WIN32)
00467 # define PLATFORM_ID "Windows"
00468
00469 #elif defined(__FreeBSD__) || defined(__FreeBSD)
00470 # define PLATFORM_ID "FreeBSD"
00471
00472 #elif defined(__NetBSD__) || defined(__NetBSD)
00473 # define PLATFORM_ID "NetBSD"
00474
00475 #elif defined(__OpenBSD__) || defined(__OPENBSD)
00476 # define PLATFORM_ID "OpenBSD"
00477
00478 #elif defined(__sun) || defined(sun)
00479 # define PLATFORM_ID "SunOS"
00480
00481 #elif defined(_AIX) || defined(__AIX) || defined(__AIX__) || defined(__aix) || defined(__aix__)
00482 # define PLATFORM_ID "AIX"
00483
00484 #elif defined(__hpux) || defined(__hpux__)
00485 # define PLATFORM_ID "HP-UX"
00486

```

```
00487 #elif defined(__HAIKU__)
00488 # define PLATFORM_ID "Haiku"
00489
00490 #elif defined(__BeOS) || defined(__BEOS__) || defined(_BEOS)
00491 # define PLATFORM_ID "BeOS"
00492
00493 #elif defined(__QNX__) || defined(__QNXNTO__)
00494 # define PLATFORM_ID "QNX"
00495
00496 #elif defined(__tru64) || defined(_tru64) || defined(__TRU64__)
00497 # define PLATFORM_ID "Tru64"
00498
00499 #elif defined(__riscos) || defined(__riscos__)
00500 # define PLATFORM_ID "RISCos"
00501
00502 #elif defined(__sinix) || defined(__sinix__) || defined(__SINIX__)
00503 # define PLATFORM_ID "SINIX"
00504
00505 #elif defined(__UNIX_SV__)
00506 # define PLATFORM_ID "UNIX_SV"
00507
00508 #elif defined(__bsdos__)
00509 # define PLATFORM_ID "BSDOS"
00510
00511 #elif defined(_MPRAS) || defined(MPRAS)
00512 # define PLATFORM_ID "MP-RAS"
00513
00514 #elif defined(__osf) || defined(__osf__)
00515 # define PLATFORM_ID "OSF1"
00516
00517 #elif defined(_SCO_SV) || defined(SCO_SV) || defined(sco_sv)
00518 # define PLATFORM_ID "SCO_SV"
00519
00520 #elif defined(__ultrix) || defined(__ultrix__) || defined(_ULTRIX)
00521 # define PLATFORM_ID "ULTRIX"
00522
00523 #elif defined(__XENIX__) || defined(_XENIX) || defined(XENIX)
00524 # define PLATFORM_ID "Xenix"
00525
00526 #elif defined(__WATCOMC__)
00527 # if defined(__LINUX__)
00528 # define PLATFORM_ID "Linux"
00529
00530 # elif defined(__DOS__)
00531 # define PLATFORM_ID "DOS"
00532
00533 # elif defined(__OS2__)
00534 # define PLATFORM_ID "OS2"
00535
00536 # elif defined(__WINDOWS__)
00537 # define PLATFORM_ID "Windows3x"
00538
00539 # elif defined(__VXWORKS__)
00540 # define PLATFORM_ID "VxWorks"
00541
00542 # else /* unknown platform */
00543 # define PLATFORM_ID
00544 # endif
00545
00546 #elif defined(__INTEGRITY)
00547 # if defined(INT_178B)
00548 # define PLATFORM_ID "Integrity178"
00549
00550 # else /* regular Integrity */
00551 # define PLATFORM_ID "Integrity"
00552 # endif
00553
00554 # elif defined(_ADI_COMPILER)
00555 # define PLATFORM_ID "ADSP"
00556
00557 #else /* unknown platform */
00558 # define PLATFORM_ID
00559
00560 #endif
00561
00562 /* For windows compilers MSVC and Intel we can determine
00563 the architecture of the compiler being used. This is because
00564 the compilers do not have flags that can change the architecture,
00565 but rather depend on which compiler is being used
00566 */
00567 #if defined(_WIN32) && defined(_MSC_VER)
00568 # if defined(_M_IA64)
00569 # define ARCHITECTURE_ID "IA64"
00570
00571 # elif defined(_M_ARM64EC)
00572 # define ARCHITECTURE_ID "ARM64EC"
00573
```

```
00574 # elif defined(_M_X64) || defined(_M_AMD64)
00575 #   define ARCHITECTURE_ID "x64"
00576
00577 # elif defined(_M_IX86)
00578 #   define ARCHITECTURE_ID "X86"
00579
00580 # elif defined(_M_ARM64)
00581 #   define ARCHITECTURE_ID "ARM64"
00582
00583 # elif defined(_M_ARM)
00584 #   if _M_ARM == 4
00585 #     define ARCHITECTURE_ID "ARMV4I"
00586 #   elif _M_ARM == 5
00587 #     define ARCHITECTURE_ID "ARMV5I"
00588 #   else
00589 #     define ARCHITECTURE_ID "ARMV" STRINGIFY(_M_ARM)
00590 #   endif
00591
00592 # elif defined(_M_MIPS)
00593 #   define ARCHITECTURE_ID "MIPS"
00594
00595 # elif defined(_M_SH)
00596 #   define ARCHITECTURE_ID "SHx"
00597
00598 # else /* unknown architecture */
00599 #   define ARCHITECTURE_ID ""
00600 # endif
00601
00602 #elif defined(__WATCOMC__)
00603 # if defined(_M_I86)
00604 #   define ARCHITECTURE_ID "I86"
00605
00606 # elif defined(_M_IX86)
00607 #   define ARCHITECTURE_ID "X86"
00608
00609 # else /* unknown architecture */
00610 #   define ARCHITECTURE_ID ""
00611 # endif
00612
00613 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00614 # if defined(__ICCARM__)
00615 #   define ARCHITECTURE_ID "ARM"
00616
00617 # elif defined(__ICCRX__)
00618 #   define ARCHITECTURE_ID "RX"
00619
00620 # elif defined(__ICCRH850__)
00621 #   define ARCHITECTURE_ID "RH850"
00622
00623 # elif defined(__ICCRL78__)
00624 #   define ARCHITECTURE_ID "RL78"
00625
00626 # elif defined(__ICCRISCV__)
00627 #   define ARCHITECTURE_ID "RISCV"
00628
00629 # elif defined(__ICCAVR__)
00630 #   define ARCHITECTURE_ID "AVR"
00631
00632 # elif defined(__ICC430__)
00633 #   define ARCHITECTURE_ID "MSP430"
00634
00635 # elif defined(__ICCV850__)
00636 #   define ARCHITECTURE_ID "V850"
00637
00638 # elif defined(__ICC8051__)
00639 #   define ARCHITECTURE_ID "8051"
00640
00641 # elif defined(__ICCSTM8__)
00642 #   define ARCHITECTURE_ID "STM8"
00643
00644 # else /* unknown architecture */
00645 #   define ARCHITECTURE_ID ""
00646 # endif
00647
00648 #elif defined(__ghs__)
00649 # if defined(__PPC64__)
00650 #   define ARCHITECTURE_ID "PPC64"
00651
00652 # elif defined(__ppc__)
00653 #   define ARCHITECTURE_ID "PPC"
00654
00655 # elif defined(__ARM__)
00656 #   define ARCHITECTURE_ID "ARM"
00657
00658 # elif defined(__x86_64__)
00659 #   define ARCHITECTURE_ID "x64"
00660
```

```

00661 # elif defined(__i386__)
00662 #   define ARCHITECTURE_ID "X86"
00663
00664 # else /* unknown architecture */
00665 #   define ARCHITECTURE_ID ""
00666 # endif
00667
00668 #elif defined(__TI_COMPILER_VERSION__)
00669 # if defined(__TI_ARM__)
00670 #   define ARCHITECTURE_ID "ARM"
00671
00672 # elif defined(__MSP430__)
00673 #   define ARCHITECTURE_ID "MSP430"
00674
00675 # elif defined(__TMS320C28XX__)
00676 #   define ARCHITECTURE_ID "TMS320C28x"
00677
00678 # elif defined(__TMS320C6X__) || defined(__TMS320C6X)
00679 #   define ARCHITECTURE_ID "TMS320C6x"
00680
00681 # else /* unknown architecture */
00682 #   define ARCHITECTURE_ID ""
00683 # endif
00684
00685 # elif defined(__ADSPSHARC__)
00686 #   define ARCHITECTURE_ID "SHARC"
00687
00688 # elif defined(__ADSPBLACKFIN__)
00689 #   define ARCHITECTURE_ID "Blackfin"
00690
00691 #elif defined(__TASKING__)
00692
00693 # if defined(__CTC__) || defined(__CPTC__)
00694 #   define ARCHITECTURE_ID "TriCore"
00695
00696 # elif defined(__CMCS__)
00697 #   define ARCHITECTURE_ID "MCS"
00698
00699 # elif defined(__CARM__)
00700 #   define ARCHITECTURE_ID "ARM"
00701
00702 # elif defined(__CARC__)
00703 #   define ARCHITECTURE_ID "ARC"
00704
00705 # elif defined(__C51__)
00706 #   define ARCHITECTURE_ID "8051"
00707
00708 # elif defined(__CPCP__)
00709 #   define ARCHITECTURE_ID "PCP"
00710
00711 # else
00712 #   define ARCHITECTURE_ID ""
00713 # endif
00714
00715 #else
00716 #   define ARCHITECTURE_ID
00717 #endif
00718
00719 /* Convert integer to decimal digit literals. */
00720 #define DEC(n) \
00721   ('0' + ((n) / 10000000) % 10), \
00722   ('0' + ((n) / 1000000) % 10), \
00723   ('0' + ((n) / 100000) % 10), \
00724   ('0' + ((n) / 10000) % 10), \
00725   ('0' + ((n) / 1000) % 10), \
00726   ('0' + ((n) / 100) % 10), \
00727   ('0' + ((n) / 10) % 10), \
00728   ('0' + ((n) % 10))
00729
00730 /* Convert integer to hex digit literals. */
00731 #define HEX(n) \
00732   ('0' + ((n) >> 28 & 0xF)), \
00733   ('0' + ((n) >> 24 & 0xF)), \
00734   ('0' + ((n) >> 20 & 0xF)), \
00735   ('0' + ((n) >> 16 & 0xF)), \
00736   ('0' + ((n) >> 12 & 0xF)), \
00737   ('0' + ((n) >> 8 & 0xF)), \
00738   ('0' + ((n) >> 4 & 0xF)), \
00739   ('0' + ((n) & 0xF))
00740
00741 /* Construct a string literal encoding the version number. */
00742 #ifndef COMPILER_VERSION
00743 char const* info_version = "INFO" ":" "compiler_version[" COMPILER_VERSION "];"
00744
00745 /* Construct a string literal encoding the version number components. */
00746 #elif defined(COMPILER_VERSION_MAJOR)
00747 char const info_version[] = {

```

```

00748 'I', 'N', 'F', 'O', ':',
00749 'c', 'o', 'm', 'p', 'i', 'l', 'e', 'r', '-', 'v', 'e', 'r', 's', 'i', 'o', 'n', '[',
00750 COMPILER_VERSION_MAJOR,
00751 # ifdef COMPILER_VERSION_MINOR
00752 '.', COMPILER_VERSION_MINOR,
00753 # ifdef COMPILER_VERSION_PATCH
00754 '.', COMPILER_VERSION_PATCH,
00755 # ifdef COMPILER_VERSION_TWEAK
00756 '.', COMPILER_VERSION_TWEAK,
00757 # endif
00758 # endif
00759 # endif
00760 ']', '\0';
00761 #endif
00762
00763 /* Construct a string literal encoding the internal version number. */
00764 #ifdef COMPILER_VERSION_INTERNAL
00765 char const info_version_internal[] = {
00766 'I', 'N', 'F', 'O', ':',
00767 'c', 'o', 'm', 'p', 'i', 'l', 'e', 'r', '-', 'v', 'e', 'r', 's', 'i', 'o', 'n', '-',
00768 'i', 'n', 't', 'e', 'r', 'n', 'a', 'l', '[',
00769 COMPILER_VERSION_INTERNAL, ']', '\0';
00770 #elif defined(COMPILER_VERSION_INTERNAL_STR)
00771 char const* info_version_internal = "INFO" ":" "compiler_version_internal["
COMPILER_VERSION_INTERNAL_STR "];"
00772 #endif
00773
00774 /* Construct a string literal encoding the version number components. */
00775 #ifdef SIMULATE_VERSION_MAJOR
00776 char const info_simulate_version[] = {
00777 'I', 'N', 'F', 'O', ':',
00778 's', 'i', 'm', 'u', 'l', 'a', 't', 'e', '-', 'v', 'e', 'r', 's', 'i', 'o', 'n', '[',
00779 SIMULATE_VERSION_MAJOR,
00780 # ifdef SIMULATE_VERSION_MINOR
00781 '.', SIMULATE_VERSION_MINOR,
00782 # ifdef SIMULATE_VERSION_PATCH
00783 '.', SIMULATE_VERSION_PATCH,
00784 # ifdef SIMULATE_VERSION_TWEAK
00785 '.', SIMULATE_VERSION_TWEAK,
00786 # endif
00787 # endif
00788 # endif
00789 ']', '\0';
00790 #endif
00791
00792 /* Construct the string literal in pieces to prevent the source from
00793 getting matched. Store it in a pointer rather than an array
00794 because some compilers will just produce instructions to fill the
00795 array rather than assigning a pointer to a static array. */
00796 char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "];"
00797 char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "];"
00798
00799
00800
00801 #if !defined(__STDC__) && !defined(__clang__)
00802 # if defined(_MSC_VER) || defined(__ibmxl__) || defined(__IBMC__)
00803 # define C_VERSION "90"
00804 # else
00805 # define C_VERSION
00806 # endif
00807 #elif __STDC_VERSION__ > 201710L
00808 # define C_VERSION "23"
00809 #elif __STDC_VERSION__ >= 201710L
00810 # define C_VERSION "17"
00811 #elif __STDC_VERSION__ >= 201000L
00812 # define C_VERSION "11"
00813 #elif __STDC_VERSION__ >= 199901L
00814 # define C_VERSION "99"
00815 #else
00816 # define C_VERSION "90"
00817 #endif
00818 const char* info_language_standard_default =
00819 "INFO" ":" "standard_default[" C_VERSION "];"
00820
00821 const char* info_language_extensions_default = "INFO" ":" "extensions_default["
00822 #if (defined(__clang__) || defined(__GNUC__) || defined(__xlc__) ||
00823      defined(__TI_COMPILER_VERSION__)) &&
00824 !defined(__STRICT_ANSI__)
00825 "ON"
00826 #else
00827 "OFF"
00828 #endif
00829 "];"
00830
00831 /*-----*/
00832
00833 #ifdef ID_VOID_MAIN

```



```

00834 void main() {}
00835 #else
00836 # if defined(__CLASSIC_C__)
00837 int main(argc, argv) int argc; char *argv[];
00838 # else
00839 int main(int argc, char* argv[])
00840 # endif
00841 {
00842     int require = 0;
00843     require += info_compiler[argc];
00844     require += info_platform[argc];
00845     require += info_arch[argc];
00846 #ifdef COMPILER_VERSION_MAJOR
00847     require += info_version[argc];
00848 #endif
00849 #ifdef COMPILER_VERSION_INTERNAL
00850     require += info_version_internal[argc];
00851 #endif
00852 #ifdef SIMULATE_ID
00853     require += info_simulate[argc];
00854 #endif
00855 #ifdef SIMULATE_VERSION_MAJOR
00856     require += info_simulate_version[argc];
00857 #endif
00858 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00859     require += info_cray[argc];
00860 #endif
00861     require += info_language_standard_default[argc];
00862     require += info_language_extensions_default[argc];
00863     (void)argv;
00864     return require;
00865 }
00866 #endif

```

7.5 /Users/samanthapope/msdscriptRepo/msdScript/cmake-build-debug/CMakeFiles/3.27.8/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define `__has_include(x)` 0
- #define `COMPILER_ID` ""
- #define `STRINGIFY_HELPER(X)` #X
- #define `STRINGIFY(X)` `STRINGIFY_HELPER(X)`
- #define `PLATFORM_ID`
- #define `ARCHITECTURE_ID`
- #define `DEC(n)`
- #define `HEX(n)`
- #define `CXX_STD` `__cplusplus`

Functions

- int `main` (int argc, char *argv[])

Variables

- char const * `info_compiler` = "INFO" ":" "compiler[" `COMPILER_ID` "]"
- char const * `info_platform` = "INFO" ":" "platform[" `PLATFORM_ID` "]"
- char const * `info_arch` = "INFO" ":" "arch[" `ARCHITECTURE_ID` "]"
- const char * `info_language_standard_default`
- const char * `info_language_extensions_default`

7.5.1 Macro Definition Documentation

7.5.1.1 `__has_include`

```
#define __has_include(
    x ) 0
```

Definition at line 11 of file [CMakeCXXCompilerId.cpp](#).

7.5.1.2 `ARCHITECTURE_ID`

```
#define ARCHITECTURE_ID
```

Definition at line 701 of file [CMakeCXXCompilerId.cpp](#).

7.5.1.3 `COMPILER_ID`

```
#define COMPILER_ID ""
```

Definition at line 412 of file [CMakeCXXCompilerId.cpp](#).

7.5.1.4 `CXX_STD`

```
#define CXX_STD __cplusplus
```

Definition at line 799 of file [CMakeCXXCompilerId.cpp](#).

7.5.1.5 `DEC`

```
#define DEC(
    n )
```

Value:

```
('0' + ((n) / 10000000) % 10), \
('0' + ((n) / 1000000) % 10), \
('0' + ((n) / 100000) % 10), \
('0' + ((n) / 10000) % 10), \
('0' + ((n) / 1000) % 10), \
('0' + ((n) / 100) % 10), \
('0' + ((n) / 10) % 10), \
('0' + ((n) % 10))
```

Definition at line 705 of file [CMakeCXXCompilerId.cpp](#).

7.5.1.6 `HEX`

```
#define HEX(
    n )
```

Value:

```
('0' + ((n) >> 28 & 0xF)), \
('0' + ((n) >> 24 & 0xF)), \
('0' + ((n) >> 20 & 0xF)), \
('0' + ((n) >> 16 & 0xF)), \
('0' + ((n) >> 12 & 0xF)), \
('0' + ((n) >> 8 & 0xF)), \
('0' + ((n) >> 4 & 0xF)), \
('0' + ((n) & 0xF))
```

Definition at line 716 of file [CMakeCXXCompilerId.cpp](#).

7.5.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 543 of file [CMakeCXXCompilerId.cpp](#).

7.5.1.8 STRINGIFY

```
#define STRINGIFY(  
    X ) STRINGIFY_HELPER(X)
```

Definition at line 433 of file [CMakeCXXCompilerId.cpp](#).

7.5.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(  
    X ) #X
```

Definition at line 432 of file [CMakeCXXCompilerId.cpp](#).

7.5.2 Function Documentation

7.5.2.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

Definition at line 830 of file [CMakeCXXCompilerId.cpp](#).

7.5.3 Variable Documentation

7.5.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 782 of file [CMakeCXXCompilerId.cpp](#).

7.5.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 419 of file [CMakeCXXCompilerId.cpp](#).

7.5.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
```

```
    "OFF"
"]"
```

Definition at line 818 of file [CMakeCXXCompilerId.cpp](#).

7.5.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
= "INFO" ":" "standard_default["
```

```
    "98"
"]"
```

Definition at line 802 of file [CMakeCXXCompilerId.cpp](#).

7.5.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 781 of file [CMakeCXXCompilerId.cpp](#).

7.6 CMakeCXXCompilerId.cpp

[Go to the documentation of this file.](#)

```
00001 /* This source file must have a .cpp extension so that all C++ compilers
00002      recognize the extension without flags. Borland does not know .cxx for
00003      example. */
00004 #ifndef __cplusplus
00005 # error "A C compiler has been selected for C++."
00006 #endif
00007
00008 #if !defined(__has_include)
00009 /* If the compiler does not have __has_include, pretend the answer is
00010      always no. */
00011 # define __has_include(x) 0
00012 #endif
00013
00014
00015 /* Version number components: V=Version, R=Revision, P=Patch
00016      Version date components: YYYY=Year, MM=Month, DD=Day */
```

```

00017
00018 #if defined(__COMO__)
00019 # define COMPILER_ID "Comeau"
00020 /* __COMO_VERSION__ = VRR */
00021 # define COMPILER_VERSION_MAJOR DEC(__COMO_VERSION__ / 100)
00022 # define COMPILER_VERSION_MINOR DEC(__COMO_VERSION__ % 100)
00023
00024 #elif defined(__INTEL_COMPILER) || defined(__ICC)
00025 # define COMPILER_ID "Intel"
00026 # if defined(_MSC_VER)
00027 #   define SIMULATE_ID "MSVC"
00028 # endif
00029 # if defined(__GNUC__)
00030 #   define SIMULATE_ID "GNU"
00031 # endif
00032 /* __INTEL_COMPILER = VRP prior to 2021, and then VVVV for 2021 and later,
00033    except that a few beta releases use the old format with V=2021. */
00034 # if __INTEL_COMPILER < 2021 || __INTEL_COMPILER == 202110 || __INTEL_COMPILER == 202111
00035 #   define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER/100)
00036 #   define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER/10 % 10)
00037 #   if defined(__INTEL_COMPILER_UPDATE)
00038 #     define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER_UPDATE)
00039 #   else
00040 #     define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER % 10)
00041 #   endif
00042 # else
00043 #   define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER)
00044 #   define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER_UPDATE)
00045 /* The third version component from --version is an update index,
00046    but no macro is provided for it. */
00047 #   define COMPILER_VERSION_PATCH DEC(0)
00048 # endif
00049 # if defined(__INTEL_COMPILER_BUILD_DATE)
00050 /* __INTEL_COMPILER_BUILD_DATE = YYYYMMDD */
00051 #   define COMPILER_VERSION_TWEAK DEC(__INTEL_COMPILER_BUILD_DATE)
00052 # endif
00053 # if defined(_MSC_VER)
00054 /* _MSC_VER = VVRR */
00055 #   define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00056 #   define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00057 # endif
00058 # if defined(__GNUC__)
00059 #   define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00060 # elif defined(__GNUG__)
00061 #   define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00062 # endif
00063 # if defined(__GNUC_MINOR__)
00064 #   define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00065 # endif
00066 # if defined(__GNUC_PATCHLEVEL__)
00067 #   define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00068 # endif
00069
00070 #elif (defined(__clang__) && defined(__INTEL_CLANG_COMPILER)) || defined(__INTEL_LLVM_COMPILER)
00071 # define COMPILER_ID "IntelLLVM"
00072 #if defined(_MSC_VER)
00073 # define SIMULATE_ID "MSVC"
00074 #endif
00075 #if defined(__GNUC__)
00076 # define SIMULATE_ID "GNU"
00077 #endif
00078 /* __INTEL_LLVM_COMPILER = VVVVRP prior to 2021.2.0, VVVVRRPP for 2021.2.0 and
00079    * later. Look for 6 digit vs. 8 digit version number to decide encoding.
00080    * VVVV is no smaller than the current year when a version is released.
00081    */
00082 #if __INTEL_LLVM_COMPILER < 1000000L
00083 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/100)
00084 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/10 % 10)
00085 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER % 10)
00086 #else
00087 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/10000)
00088 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/100 % 100)
00089 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER % 100)
00090 #endif
00091 #if defined(_MSC_VER)
00092 /* _MSC_VER = VVRR */
00093 # define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00094 # define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00095 #endif
00096 #if defined(__GNUC__)
00097 # define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00098 #elif defined(__GNUG__)
00099 # define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00100 #endif
00101 #if defined(__GNUC_MINOR__)
00102 # define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00103 #endif

```

```

00104 #if defined(__GNUC_PATCHLEVEL__)
00105 # define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00106 #endif
00107
00108 #elif defined(__PATHCC__)
00109 # define COMPILER_ID "PathScale"
00110 # define COMPILER_VERSION_MAJOR DEC(__PATHCC__)
00111 # define COMPILER_VERSION_MINOR DEC(__PATHCC_MINOR__)
00112 # if defined(__PATHCC_PATCHLEVEL__)
00113 #  define COMPILER_VERSION_PATCH DEC(__PATHCC_PATCHLEVEL__)
00114 # endif
00115
00116 #elif defined(__BORLANDC__) && defined(__CODEGEARC_VERSION__)
00117 # define COMPILER_ID "Embarcadero"
00118 # define COMPILER_VERSION_MAJOR HEX(__CODEGEARC_VERSION__>24 & 0x00FF)
00119 # define COMPILER_VERSION_MINOR HEX(__CODEGEARC_VERSION__>16 & 0x00FF)
00120 # define COMPILER_VERSION_PATCH DEC(__CODEGEARC_VERSION__ & 0xFFFF)
00121
00122 #elif defined(__BORLANDC__)
00123 # define COMPILER_ID "Borland"
00124 /* __BORLANDC__ = 0xVRR */
00125 # define COMPILER_VERSION_MAJOR HEX(__BORLANDC__>8)
00126 # define COMPILER_VERSION_MINOR HEX(__BORLANDC__ & 0xFF)
00127
00128 #elif defined(__WATCOMC__) && __WATCOMC__ < 1200
00129 # define COMPILER_ID "Watcom"
00130 /* __WATCOMC__ = VVRR */
00131 # define COMPILER_VERSION_MAJOR DEC(__WATCOMC__ / 100)
00132 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00133 # if (__WATCOMC__ % 10) > 0
00134 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00135 # endif
00136
00137 #elif defined(__WATCOMC__)
00138 # define COMPILER_ID "OpenWatcom"
00139 /* __WATCOMC__ = VVRP + 1100 */
00140 # define COMPILER_VERSION_MAJOR DEC((__WATCOMC__ - 1100) / 100)
00141 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00142 # if (__WATCOMC__ % 10) > 0
00143 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00144 # endif
00145
00146 #elif defined(__SUNPRO_CC)
00147 # define COMPILER_ID "SunPro"
00148 # if __SUNPRO_CC >= 0x5100
00149 /* __SUNPRO_CC = 0xVRRP */
00150 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC>12)
00151 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC>4 & 0xFF)
00152 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC & 0xF)
00153 # else
00154 /* __SUNPRO_CC = 0xVRP */
00155 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC>8)
00156 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC>4 & 0xF)
00157 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC & 0xF)
00158 # endif
00159
00160 #elif defined(__HP_aCC)
00161 # define COMPILER_ID "HP"
00162 /* __HP_aCC = VVRRPP */
00163 # define COMPILER_VERSION_MAJOR DEC(__HP_aCC/10000)
00164 # define COMPILER_VERSION_MINOR DEC(__HP_aCC/100 % 100)
00165 # define COMPILER_VERSION_PATCH DEC(__HP_aCC % 100)
00166
00167 #elif defined(__DECCXX)
00168 # define COMPILER_ID "Compaq"
00169 /* __DECCXX_VER = VVRRTPPPP */
00170 # define COMPILER_VERSION_MAJOR DEC(__DECCXX_VER/10000000)
00171 # define COMPILER_VERSION_MINOR DEC(__DECCXX_VER/100000 % 100)
00172 # define COMPILER_VERSION_PATCH DEC(__DECCXX_VER % 10000)
00173
00174 #elif defined(__IBMCPP__) && defined(__COMPILER_VER__)
00175 # define COMPILER_ID "zOS"
00176 /* __IBMCPP__ = VRP */
00177 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00178 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00179 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__ % 10)
00180
00181 #elif defined(__open_xl__) && defined(__clang__)
00182 # define COMPILER_ID "IBMClang"
00183 # define COMPILER_VERSION_MAJOR DEC(__open_xl_version__)
00184 # define COMPILER_VERSION_MINOR DEC(__open_xl_release__)
00185 # define COMPILER_VERSION_PATCH DEC(__open_xl_modification__)
00186 # define COMPILER_VERSION_TWEAK DEC(__open_xl_ptf_fix_level__)
00187
00188
00189 #elif defined(__ibmxl__) && defined(__clang__)
00190 # define COMPILER_ID "XLClang"

```

```

00191 # define COMPILER_VERSION_MAJOR DEC(__ibmxl_version__)
00192 # define COMPILER_VERSION_MINOR DEC(__ibmxl_release__)
00193 # define COMPILER_VERSION_PATCH DEC(__ibmxl_modification__)
00194 # define COMPILER_VERSION_TWEAK DEC(__ibmxl_ptf_fix_level__)
00195
00196
00197 #elif defined(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ >= 800
00198 # define COMPILER_ID "XL"
00199 /* __IBMCPP__ = VRP */
00200 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00201 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00202 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__ % 10)
00203
00204 #elif defined(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ < 800
00205 # define COMPILER_ID "VisualAge"
00206 /* __IBMCPP__ = VRP */
00207 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00208 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00209 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__ % 10)
00210
00211 #elif defined(__NVCOMPILER)
00212 # define COMPILER_ID "NVHPC"
00213 # define COMPILER_VERSION_MAJOR DEC(__NVCOMPILER_MAJOR__)
00214 # define COMPILER_VERSION_MINOR DEC(__NVCOMPILER_MINOR__)
00215 # if defined(__NVCOMPILER_PATCHLEVEL__)
00216 #   define COMPILER_VERSION_PATCH DEC(__NVCOMPILER_PATCHLEVEL__)
00217 # endif
00218
00219 #elif defined(__PGI)
00220 # define COMPILER_ID "PGI"
00221 # define COMPILER_VERSION_MAJOR DEC(__PGIC__)
00222 # define COMPILER_VERSION_MINOR DEC(__PGIC_MINOR__)
00223 # if defined(__PGIC_PATCHLEVEL__)
00224 #   define COMPILER_VERSION_PATCH DEC(__PGIC_PATCHLEVEL__)
00225 # endif
00226
00227 #elif defined(_CRAYC)
00228 # define COMPILER_ID "Cray"
00229 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
00230 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00231
00232 #elif defined(__TI_COMPILER_VERSION__)
00233 # define COMPILER_ID "TI"
00234 /* __TI_COMPILER_VERSION__ = VVVRRRPPP */
00235 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00236 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000 % 1000)
00237 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__ % 1000)
00238
00239 #elif defined(__CLANG_FUJITSU)
00240 # define COMPILER_ID "FujitsuClang"
00241 # define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00242 # define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00243 # define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00244 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00245
00246
00247 #elif defined(__FUJITSU)
00248 # define COMPILER_ID "Fujitsu"
00249 # if defined(__FCC_version__)
00250 #   define COMPILER_VERSION __FCC_version__
00251 # elif defined(__FCC_major__)
00252 #   define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00253 #   define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00254 #   define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00255 # endif
00256 # if defined(__fcc_version)
00257 #   define COMPILER_VERSION_INTERNAL DEC(__fcc_version)
00258 # elif defined(__FCC_VERSION)
00259 #   define COMPILER_VERSION_INTERNAL DEC(__FCC_VERSION)
00260 # endif
00261
00262
00263 #elif defined(__ghs__)
00264 # define COMPILER_ID "GHS"
00265 /* __GHS_VERSION_NUMBER = VVVVRP */
00266 # ifdef __GHS_VERSION_NUMBER
00267 #   define COMPILER_VERSION_MAJOR DEC(__GHS_VERSION_NUMBER / 100)
00268 #   define COMPILER_VERSION_MINOR DEC(__GHS_VERSION_NUMBER / 10 % 10)
00269 #   define COMPILER_VERSION_PATCH DEC(__GHS_VERSION_NUMBER % 10)
00270 # endif
00271
00272 #elif defined(__TASKING__)
00273 # define COMPILER_ID "Tasking"
00274 #   define COMPILER_VERSION_MAJOR DEC(__VERSION__/1000)
00275 #   define COMPILER_VERSION_MINOR DEC(__VERSION__ % 100)
00276 #   define COMPILER_VERSION_INTERNAL DEC(__VERSION__)
00277

```

```

00278 #elif defined(__SCO_VERSION__)
00279 # define COMPILER_ID "SCO"
00280
00281 #elif defined(__ARMCC_VERSION) && !defined(__clang__)
00282 # define COMPILER_ID "ARMCC"
00283 #if __ARMCC_VERSION >= 1000000
00284 /* __ARMCC_VERSION = VRPPPP */
00285 # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/1000000)
00286 # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 100)
00287 # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION % 10000)
00288 #else
00289 /* __ARMCC_VERSION = VRPPPP */
00290 # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/100000)
00291 # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 10)
00292 # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION % 10000)
00293 #endif
00294
00295
00296 #elif defined(__clang__) && defined(__apple_build_version__)
00297 # define COMPILER_ID "AppleClang"
00298 # if defined(_MSC_VER)
00299 #   define SIMULATE_ID "MSVC"
00300 # endif
00301 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00302 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00303 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00304 # if defined(_MSC_VER)
00305 /* _MSC_VER = VVRR */
00306 #   define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00307 #   define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00308 # endif
00309 # define COMPILER_VERSION_TWEAK DEC(__apple_build_version__)
00310
00311 #elif defined(__clang__) && defined(__ARMCOMPILER_VERSION)
00312 # define COMPILER_ID "ARMClang"
00313 # define COMPILER_VERSION_MAJOR DEC(__ARMCOMPILER_VERSION/1000000)
00314 # define COMPILER_VERSION_MINOR DEC(__ARMCOMPILER_VERSION/10000 % 100)
00315 # define COMPILER_VERSION_PATCH DEC(__ARMCOMPILER_VERSION/100 % 100)
00316 # define COMPILER_VERSION_INTERNAL DEC(__ARMCOMPILER_VERSION)
00317
00318 #elif defined(__clang__)
00319 # define COMPILER_ID "Clang"
00320 # if defined(_MSC_VER)
00321 #   define SIMULATE_ID "MSVC"
00322 # endif
00323 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00324 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00325 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00326 # if defined(_MSC_VER)
00327 /* _MSC_VER = VVRR */
00328 #   define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00329 #   define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00330 # endif
00331 # endif
00332
00333 #elif defined(__LCC__) && (defined(__GNUC__) || defined(__GNU__ || defined(__MCST__)))
00334 # define COMPILER_ID "LCC"
00335 # define COMPILER_VERSION_MAJOR DEC(__LCC__ / 100)
00336 # define COMPILER_VERSION_MINOR DEC(__LCC__ % 100)
00337 # if defined(__LCC_MINOR__)
00338 #   define COMPILER_VERSION_PATCH DEC(__LCC_MINOR__)
00339 # endif
00340 # if defined(__GNUC__) && defined(__GNUC_MINOR__)
00341 #   define SIMULATE_ID "GNU"
00342 #   define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00343 #   define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00344 #   if defined(__GNUC_PATCHLEVEL__)
00345 #     define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00346 #   endif
00347 # endif
00348
00349 #elif defined(__GNUC__) || defined(__GNU__)
00350 # define COMPILER_ID "GNU"
00351 # if defined(__GNUC__)
00352 #   define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00353 # else
00354 #   define COMPILER_VERSION_MAJOR DEC(__GNU__)
00355 # endif
00356 # if defined(__GNUC_MINOR__)
00357 #   define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00358 # endif
00359 # if defined(__GNUC_PATCHLEVEL__)
00360 #   define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00361 # endif
00362
00363 #elif defined(_MSC_VER)
00364 # define COMPILER_ID "MSVC"
00365 /* _MSC_VER = VVRR */

```



```

00365 # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00366 # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00367 # if defined(_MSC_FULL_VER)
00368 #   if _MSC_VER >= 1400
00369     /* _MSC_FULL_VER = VVRRPPPPP */
00370 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00371 #   else
00372     /* _MSC_FULL_VER = VVRRPPPPP */
00373 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00374 #   endif
00375 # endif
00376 # if defined(_MSC_BUILD)
00377 #   define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
00378 # endif
00379
00380 #elif defined(_ADI_COMPILER)
00381 # define COMPILER_ID "ADSP"
00382 #if defined(__VERSIONNUM__)
00383     /* __VERSIONNUM__ = 0xVVRRPPTT */
00384 #   define COMPILER_VERSION_MAJOR DEC(__VERSIONNUM__ >> 24 & 0xFF)
00385 #   define COMPILER_VERSION_MINOR DEC(__VERSIONNUM__ >> 16 & 0xFF)
00386 #   define COMPILER_VERSION_PATCH DEC(__VERSIONNUM__ >> 8 & 0xFF)
00387 #   define COMPILER_VERSION_TWEAK DEC(__VERSIONNUM__ & 0xFF)
00388 #endif
00389
00390 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00391 # define COMPILER_ID "IAR"
00392 # if defined(__VER__) && defined(__ICCARM__)
00393 #   define COMPILER_VERSION_MAJOR DEC(__VER__) / 1000000)
00394 #   define COMPILER_VERSION_MINOR DEC((__VER__) / 1000) % 1000)
00395 #   define COMPILER_VERSION_PATCH DEC(__VER__) % 1000)
00396 #   define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00397 # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRISC_V__) || defined(__ICCV850__) ||
defined(__ICC8051__) || defined(__ICSTM8__))
00398 #   define COMPILER_VERSION_MAJOR DEC(__VER__) / 100)
00399 #   define COMPILER_VERSION_MINOR DEC(__VER__) - (((__VER__) / 100)*100))
00400 #   define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00401 #   define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00402 # endif
00403
00404
00405 /* These compilers are either not known or too old to define an
00406    identification macro. Try to identify the platform and guess that
00407    it is the native compiler. */
00408 #elif defined(__hpux) || defined(__hpua)
00409 # define COMPILER_ID "HP"
00410
00411 #else /* unknown compiler */
00412 # define COMPILER_ID ""
00413 #endif
00414
00415 /* Construct the string literal in pieces to prevent the source from
00416    getting matched. Store it in a pointer rather than an array
00417    because some compilers will just produce instructions to fill the
00418    array rather than assigning a pointer to a static array. */
00419 char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]";
00420 #ifdef SIMULATE_ID
00421 char const* info_simulate = "INFO" ":" "simulate[" SIMULATE_ID "]";
00422 #endif
00423
00424 #ifdef __QNXNTO__
00425 char const* qnxnto = "INFO" ":" "qnxnto[]";
00426 #endif
00427
00428 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00429 char const* info_cray = "INFO" ":" "compiler_wrapper[CrayPrgEnv]";
00430 #endif
00431
00432 #define STRINGIFY_HELPER(X) #X
00433 #define STRINGIFY(X) STRINGIFY_HELPER(X)
00434
00435 /* Identify known platforms by name. */
00436 #if defined(__linux) || defined(__linux__) || defined(linux)
00437 #   define PLATFORM_ID "Linux"
00438
00439 #elif defined(__MSYS__)
00440 #   define PLATFORM_ID "MSYS"
00441
00442 #elif defined(__CYGWIN__)
00443 #   define PLATFORM_ID "Cygwin"
00444
00445 #elif defined(__MINGW32__)
00446 #   define PLATFORM_ID "MinGW"
00447
00448 #elif defined(__APPLE__)
00449 #   define PLATFORM_ID "Darwin"

```

```
00450
00451 #elif defined(_WIN32) || defined(__WIN32__) || defined(WIN32)
00452 # define PLATFORM_ID "Windows"
00453
00454 #elif defined(__FreeBSD__) || defined(_FreeBSD)
00455 # define PLATFORM_ID "FreeBSD"
00456
00457 #elif defined(__NetBSD__) || defined(_NetBSD)
00458 # define PLATFORM_ID "NetBSD"
00459
00460 #elif defined(__OpenBSD__) || defined(_OPENBSD)
00461 # define PLATFORM_ID "OpenBSD"
00462
00463 #elif defined(__sun) || defined(sun)
00464 # define PLATFORM_ID "SunOS"
00465
00466 #elif defined(_AIX) || defined(__AIX) || defined(__AIX__) || defined(__aix) || defined(__aix__)
00467 # define PLATFORM_ID "AIX"
00468
00469 #elif defined(__hpux) || defined(_hpux__)
00470 # define PLATFORM_ID "HP-UX"
00471
00472 #elif defined(__HAIKU__)
00473 # define PLATFORM_ID "Haiku"
00474
00475 #elif defined(__BeOS) || defined(_BEOS__) || defined(_BEOS)
00476 # define PLATFORM_ID "BeOS"
00477
00478 #elif defined(__QNX__) || defined(_QNXNTO__)
00479 # define PLATFORM_ID "QNX"
00480
00481 #elif defined(__tru64) || defined(_tru64) || defined(__TRU64__)
00482 # define PLATFORM_ID "Tru64"
00483
00484 #elif defined(__riscos) || defined(_riscos__)
00485 # define PLATFORM_ID "RISCos"
00486
00487 #elif defined(__sinix) || defined(_sinix__) || defined(__SINIX__)
00488 # define PLATFORM_ID "SINIX"
00489
00490 #elif defined(__UNIX_SV__)
00491 # define PLATFORM_ID "UNIX_SV"
00492
00493 #elif defined(__bsdos__)
00494 # define PLATFORM_ID "BSDOS"
00495
00496 #elif defined(_MPRAS) || defined(MPRAS)
00497 # define PLATFORM_ID "MP-RAS"
00498
00499 #elif defined(__osf) || defined(_osf__)
00500 # define PLATFORM_ID "OSF1"
00501
00502 #elif defined(_SCO_SV) || defined(SCO_SV) || defined(sco_sv)
00503 # define PLATFORM_ID "SCO_SV"
00504
00505 #elif defined(__ultrix) || defined(_ultrix__) || defined(ULTRIX)
00506 # define PLATFORM_ID "ULTRIX"
00507
00508 #elif defined(__XENIX__) || defined(_XENIX) || defined(XENIX)
00509 # define PLATFORM_ID "Xenix"
00510
00511 #elif defined(__WATCOMC__)
00512 # if defined(__LINUX__)
00513 #   define PLATFORM_ID "Linux"
00514
00515 # elif defined(__DOS__)
00516 #   define PLATFORM_ID "DOS"
00517
00518 # elif defined(__OS2__)
00519 #   define PLATFORM_ID "OS2"
00520
00521 # elif defined(__WINDOWS__)
00522 #   define PLATFORM_ID "Windows3x"
00523
00524 # elif defined(__VXWORKS__)
00525 #   define PLATFORM_ID "VxWorks"
00526
00527 # else /* unknown platform */
00528 #   define PLATFORM_ID
00529 # endif
00530
00531 #elif defined(__INTEGRITY)
00532 # if defined(INT_178B)
00533 #   define PLATFORM_ID "Integrity178"
00534
00535 # else /* regular Integrity */
00536 #   define PLATFORM_ID "Integrity"
```

```

00537 # endif
00538
00539 # elif defined(_ADI_COMPILER)
00540 #   define PLATFORM_ID "ADSP"
00541
00542 #else /* unknown platform */
00543 # define PLATFORM_ID
00544
00545 #endif
00546
00547 /* For windows compilers MSVC and Intel we can determine
00548    the architecture of the compiler being used. This is because
00549    the compilers do not have flags that can change the architecture,
00550    but rather depend on which compiler is being used
00551 */
00552 #if defined(_WIN32) && defined(_MSC_VER)
00553 # if defined(_M_IA64)
00554 #   define ARCHITECTURE_ID "IA64"
00555
00556 # elif defined(_M_ARM64EC)
00557 #   define ARCHITECTURE_ID "ARM64EC"
00558
00559 # elif defined(_M_X64) || defined(_M_AMD64)
00560 #   define ARCHITECTURE_ID "x64"
00561
00562 # elif defined(_M_IX86)
00563 #   define ARCHITECTURE_ID "X86"
00564
00565 # elif defined(_M_ARM64)
00566 #   define ARCHITECTURE_ID "ARM64"
00567
00568 # elif defined(_M_ARM)
00569 #   if _M_ARM == 4
00570 #     define ARCHITECTURE_ID "ARMV4I"
00571 #   elif _M_ARM == 5
00572 #     define ARCHITECTURE_ID "ARMV5I"
00573 #   else
00574 #     define ARCHITECTURE_ID "ARMV" STRINGIFY(_M_ARM)
00575 #   endif
00576
00577 # elif defined(_M_MIPS)
00578 #   define ARCHITECTURE_ID "MIPS"
00579
00580 # elif defined(_M_SH)
00581 #   define ARCHITECTURE_ID "SHx"
00582
00583 # else /* unknown architecture */
00584 #   define ARCHITECTURE_ID ""
00585 # endif
00586
00587 #elif defined(__WATCOMC__)
00588 # if defined(_M_I86)
00589 #   define ARCHITECTURE_ID "I86"
00590
00591 # elif defined(_M_IX86)
00592 #   define ARCHITECTURE_ID "X86"
00593
00594 # else /* unknown architecture */
00595 #   define ARCHITECTURE_ID ""
00596 # endif
00597
00598 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00599 # if defined(__ICCARM__)
00600 #   define ARCHITECTURE_ID "ARM"
00601
00602 # elif defined(__ICCRX__)
00603 #   define ARCHITECTURE_ID "RX"
00604
00605 # elif defined(__ICCRH850__)
00606 #   define ARCHITECTURE_ID "RH850"
00607
00608 # elif defined(__ICCRL78__)
00609 #   define ARCHITECTURE_ID "RL78"
00610
00611 # elif defined(__ICCRISCV__)
00612 #   define ARCHITECTURE_ID "RISCV"
00613
00614 # elif defined(__ICCAVR__)
00615 #   define ARCHITECTURE_ID "AVR"
00616
00617 # elif defined(__ICC430__)
00618 #   define ARCHITECTURE_ID "MSP430"
00619
00620 # elif defined(__ICCV850__)
00621 #   define ARCHITECTURE_ID "V850"
00622
00623 # elif defined(__ICC8051__)

```

```

00624 # define ARCHITECTURE_ID "8051"
00625
00626 # elif defined(__ICSTM8__)
00627 # define ARCHITECTURE_ID "STM8"
00628
00629 # else /* unknown architecture */
00630 # define ARCHITECTURE_ID ""
00631 # endif
00632
00633 #elif defined(__ghs__)
00634 # if defined(__PPC64__)
00635 # define ARCHITECTURE_ID "PPC64"
00636
00637 # elif defined(__ppc__)
00638 # define ARCHITECTURE_ID "PPC"
00639
00640 # elif defined(__ARM__)
00641 # define ARCHITECTURE_ID "ARM"
00642
00643 # elif defined(__x86_64__)
00644 # define ARCHITECTURE_ID "x64"
00645
00646 # elif defined(__i386__)
00647 # define ARCHITECTURE_ID "X86"
00648
00649 # else /* unknown architecture */
00650 # define ARCHITECTURE_ID ""
00651 # endif
00652
00653 #elif defined(__TI_COMPILER_VERSION__)
00654 # if defined(__TI_ARM__)
00655 # define ARCHITECTURE_ID "ARM"
00656
00657 # elif defined(__MSP430__)
00658 # define ARCHITECTURE_ID "MSP430"
00659
00660 # elif defined(__TMS320C28XX__)
00661 # define ARCHITECTURE_ID "TMS320C28x"
00662
00663 # elif defined(__TMS320C6X__) || defined(__TMS320C6X)
00664 # define ARCHITECTURE_ID "TMS320C6x"
00665
00666 # else /* unknown architecture */
00667 # define ARCHITECTURE_ID ""
00668 # endif
00669
00670 # elif defined(__ADSPSHARC__)
00671 # define ARCHITECTURE_ID "SHARC"
00672
00673 # elif defined(__ADSPBLACKFIN__)
00674 # define ARCHITECTURE_ID "Blackfin"
00675
00676 #elif defined(__TASKING__)
00677
00678 # if defined(__CTC__) || defined(__CPTC__)
00679 # define ARCHITECTURE_ID "TriCore"
00680
00681 # elif defined(__CMCS__)
00682 # define ARCHITECTURE_ID "MCS"
00683
00684 # elif defined(__CARM__)
00685 # define ARCHITECTURE_ID "ARM"
00686
00687 # elif defined(__CARC__)
00688 # define ARCHITECTURE_ID "ARC"
00689
00690 # elif defined(__C51__)
00691 # define ARCHITECTURE_ID "8051"
00692
00693 # elif defined(__CPCP__)
00694 # define ARCHITECTURE_ID "PCP"
00695
00696 # else
00697 # define ARCHITECTURE_ID ""
00698 # endif
00699
00700 #else
00701 # define ARCHITECTURE_ID
00702 #endif
00703
00704 /* Convert integer to decimal digit literals. */
00705 #define DEC(n) \
00706 ('0' + ((n) / 10000000) % 10), \
00707 ('0' + ((n) / 1000000) % 10), \
00708 ('0' + ((n) / 100000) % 10), \
00709 ('0' + ((n) / 10000) % 10), \
00710 ('0' + ((n) / 1000) % 10), \

```

```

00711 ('0' + ((n) / 100)%10)), \
00712 ('0' + ((n) / 10)%10)), \
00713 ('0' + ((n) % 10))
00714
00715 /* Convert integer to hex digit literals. */
00716 #define HEX(n) \
00717 ('0' + ((n)>28 & 0xF)), \
00718 ('0' + ((n)>24 & 0xF)), \
00719 ('0' + ((n)>20 & 0xF)), \
00720 ('0' + ((n)>16 & 0xF)), \
00721 ('0' + ((n)>12 & 0xF)), \
00722 ('0' + ((n)>8 & 0xF)), \
00723 ('0' + ((n)>4 & 0xF)), \
00724 ('0' + ((n) & 0xF))
00725
00726 /* Construct a string literal encoding the version number. */
00727 #ifndef COMPILER_VERSION
00728 char const* info_version = "INFO" ":" "compiler_version[" COMPILER_VERSION "];"
00729
00730 /* Construct a string literal encoding the version number components. */
00731 #elif defined(COMPILER_VERSION_MAJOR)
00732 char const info_version[] = {
00733 'I', 'N', 'F', 'O', ':',
00734 'c', 'o', 'm', 'p', 'i', 'l', 'e', 'r', '_', 'v', 'e', 'r', 's', 'i', 'o', 'n', '[',
00735 COMPILER_VERSION_MAJOR,
00736 # ifdef COMPILER_VERSION_MINOR
00737 '.', COMPILER_VERSION_MINOR,
00738 # ifdef COMPILER_VERSION_PATCH
00739 '.', COMPILER_VERSION_PATCH,
00740 # ifdef COMPILER_VERSION_TWEAK
00741 '.', COMPILER_VERSION_TWEAK,
00742 # endif
00743 # endif
00744 # endif
00745 ']', '\0'};
00746 #endif
00747
00748 /* Construct a string literal encoding the internal version number. */
00749 #ifndef COMPILER_VERSION_INTERNAL
00750 char const info_version_internal[] = {
00751 'I', 'N', 'F', 'O', ':',
00752 'c', 'o', 'm', 'p', 'i', 'l', 'e', 'r', '_', 'v', 'e', 'r', 's', 'i', 'o', 'n', '_',
00753 'i', 'n', 't', 'e', 'r', 'n', 'a', 'l', '[',
00754 COMPILER_VERSION_INTERNAL, ']', '\0'};
00755 #elif defined(COMPILER_VERSION_INTERNAL_STR)
00756 char const* info_version_internal = "INFO" ":" "compiler_version_internal["
COMPILER_VERSION_INTERNAL_STR "];"
00757 #endif
00758
00759 /* Construct a string literal encoding the version number components. */
00760 #ifndef SIMULATE_VERSION_MAJOR
00761 char const info_simulate_version[] = {
00762 'I', 'N', 'F', 'O', ':',
00763 's', 'i', 'm', 'u', 'l', 'a', 't', 'e', '_', 'v', 'e', 'r', 's', 'i', 'o', 'n', '[',
00764 SIMULATE_VERSION_MAJOR,
00765 # ifdef SIMULATE_VERSION_MINOR
00766 '.', SIMULATE_VERSION_MINOR,
00767 # ifdef SIMULATE_VERSION_PATCH
00768 '.', SIMULATE_VERSION_PATCH,
00769 # ifdef SIMULATE_VERSION_TWEAK
00770 '.', SIMULATE_VERSION_TWEAK,
00771 # endif
00772 # endif
00773 # endif
00774 ']', '\0'};
00775 #endif
00776
00777 /* Construct the string literal in pieces to prevent the source from
00778 getting matched. Store it in a pointer rather than an array
00779 because some compilers will just produce instructions to fill the
00780 array rather than assigning a pointer to a static array. */
00781 char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "];"
00782 char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "];"
00783
00784
00785
00786 #if defined(__INTEL_COMPILER) && defined(_MSVC_LANG) && _MSVC_LANG < 201403L
00787 # if defined(__INTEL_CXX11_MODE__)
00788 # if defined(__cpp_aggregate_nsdmi)
00789 # define CXX_STD 201402L
00790 # else
00791 # define CXX_STD 201103L
00792 # endif
00793 # else
00794 # define CXX_STD 199711L
00795 # endif
00796 #elif defined(_MSC_VER) && defined(_MSVC_LANG)

```

```

00797 # define CXX_STD _MSVC_LANG
00798 #else
00799 # define CXX_STD __cplusplus
00800 #endif
00801
00802 const char* info_language_standard_default = "INFO" ":" "standard_default["
00803 #if CXX_STD > 202002L
00804     "23"
00805 #elif CXX_STD > 201703L
00806     "20"
00807 #elif CXX_STD >= 201703L
00808     "17"
00809 #elif CXX_STD >= 201402L
00810     "14"
00811 #elif CXX_STD >= 201103L
00812     "11"
00813 #else
00814     "98"
00815 #endif
00816 "];
00817
00818 const char* info_language_extensions_default = "INFO" ":" "extensions_default["
00819 #if (defined(__clang__) || defined(__GNUC__) || defined(__xlc__) ||
00820     defined(__TI_COMPILER_VERSION__)) &&
00821     !defined(__STRICT_ANSI__)
00822     "ON"
00823 #else
00824     "OFF"
00825 #endif
00826 "];
00827
00828 /*-----*/
00829
00830 int main(int argc, char* argv[])
00831 {
00832     int require = 0;
00833     require += info_compiler[argc];
00834     require += info_platform[argc];
00835     require += info_arch[argc];
00836 #ifdef COMPILER_VERSION_MAJOR
00837     require += info_version[argc];
00838 #endif
00839 #ifdef COMPILER_VERSION_INTERNAL
00840     require += info_version_internal[argc];
00841 #endif
00842 #ifdef SIMULATE_ID
00843     require += info_simulate[argc];
00844 #endif
00845 #ifdef SIMULATE_VERSION_MAJOR
00846     require += info_simulate_version[argc];
00847 #endif
00848 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00849     require += info_cray[argc];
00850 #endif
00851     require += info_language_standard_default[argc];
00852     require += info_language_extensions_default[argc];
00853     (void)argv;
00854     return require;
00855 }

```

7.7 /Users/samanthapope/msdscriptRepo/msdScript/cmdline.cpp File Reference

```

#include <iostream>
#include <cstring>
#include "cmdline.h"
#include "catch.h"

```

Macros

- #define CATCH_CONFIG_RUNNER

Functions

- void [useArgs](#) (int argc, const char *argv[])

7.7.1 Macro Definition Documentation

7.7.1.1 CATCH_CONFIG_RUNNER

```
#define CATCH_CONFIG_RUNNER
```

Definition at line 7 of file [cmdline.cpp](#).

7.7.2 Function Documentation

7.7.2.1 useArgs()

```
void useArgs (
    int argc,
    const char * argv[] )
```

Definition at line 11 of file [cmdline.cpp](#).

7.8 cmdline.cpp

[Go to the documentation of this file.](#)

```
00001 //
00002 // Created by Samantha Pope on 1/19/24.
00003 //
00004
00005 #include <iostream>
00006 #include <cstring> // For strcmp
00007 #define CATCH_CONFIG_RUNNER
00008 #include "cmdline.h"
00009 #include "catch.h"
00010
00011 void useArgs(int argc, const char *argv[]) {
00012     static bool testAlreadyPressed = false;
00013
00014     for (int i = 1; i < argc; ++i) {
00015         //loop for all of it
00016         if (strcmp(argv[i], "--help") == 0) {
00017             std::cout << "Usage: ./msdscrip --[option]\n"
00018                 << "Options:\n"
00019                 << "  --help  Show this help message\n"
00020                 << "  --test  Run tests\n";
00021             exit(0);
00022         }
00023
00024         //have to control for if it is the first test that they wrote or not
00025         else if (strcmp(argv[i], "--test") == 0) {
00026             if (!testAlreadyPressed) {
00027                 testAlreadyPressed = true;
00028                 if (Catch::Session().run(1, argv) != 0) {
00029                     //if catch returns a non-zero value, then kill the program
00030                     exit(1);
00031                 }
00032             }
00033             std::cout << "Tests passed\n";
00034         }
00035         //--test flag already seen
00036         else {
00037             std::cerr << "Error: --test argument already seen\n";
00038             exit(1);
00039         }
00040     } else {
00041         std::cerr << "Error: Unknown argument:  " << argv[i] << "\n";
00042         exit(1);
00043     }
00044 }
00045
00046 }
```

7.9 /Users/samanthapope/msdscriptRepo/msdScript/cmdline.h File Reference

Functions

- void [useArgs](#) (int argc, const char *argv[])

7.9.1 Function Documentation

7.9.1.1 useArgs()

```
void useArgs (
    int argc,
    const char * argv[] )
```

Definition at line 11 of file [cmdline.cpp](#).

7.10 cmdline.h

[Go to the documentation of this file.](#)

```
00001 //
00002 // Created by Samantha Pope on 1/19/24.
00003 //
00004
00005 #ifndef CS6015PROJECT_CMDLINE_H
00006 #define CS6015PROJECT_CMDLINE_H
00007
00008
00009     void useArgs(int argc, const char * argv[]);
00010
00011
00012 #endif //CS6015PROJECT_CMDLINE_H
```

7.11 /Users/samanthapope/msdscriptRepo/msdScript/Expr.cpp File Reference

```
#include <sstream>
#include "Expr.h"
```

7.12 Expr.cpp

[Go to the documentation of this file.](#)

```
00001 #include <sstream>
00002 #include "Expr.h"
00003
00008 Num::Num(int val) {
00009     this->val = val;
00010 }
00011
00017 Add::Add(Expr *lhs, Expr *rhs) {
00018     this->lhs = lhs;
00019     this->rhs = rhs;
00020 }
```



```

00021
00027 Mult::Mult(Expr *lhs, Expr *rhs) {
00028     this->lhs = lhs;
00029     this->rhs = rhs;
00030 }
00031
00036 Var::Var(const std::string& varPassed) : var(std::move(varPassed)) {
00037 }
00038
00044 bool Num::equals(Expr *e) {
00045     Num* numPtr = dynamic_cast<Num*>(e);
00046     return numPtr && this->val == numPtr->val;
00047 }
00048
00054 bool Add::equals(Expr *e) {
00055     Add* addPtr = dynamic_cast<Add*>(e);
00056     return addPtr && ((this->lhs->equals(addPtr->lhs) && this->rhs->equals(addPtr->rhs)) ||
00057         (this->lhs->equals(addPtr->rhs) && this->rhs->equals(addPtr->lhs)));
00058 }
00059
00065 bool Mult::equals(Expr *e) {
00066     Mult* multPtr = dynamic_cast<Mult*>(e);
00067     return multPtr && ((this->lhs->equals(multPtr->lhs) && this->rhs->equals(multPtr->rhs)) ||
00068         (this->lhs->equals(multPtr->rhs) && this->rhs->equals(multPtr->lhs)));
00069 }
00070
00076 bool Var::equals(Expr *e) {
00077     Var* varPtr = dynamic_cast<Var*>(e);
00078     return varPtr && this->var == varPtr->var;
00079 }
00080
00085 int Num::interp() {
00086     return this->val;
00087 }
00088
00093 int Add::interp() {
00094     return this->lhs->interp() + this->rhs->interp();
00095 }
00096
00101 int Mult::interp() {
00102     return this->lhs->interp() * this->rhs->interp();
00103 }
00104
00109 int Var::interp() {
00110     throw std::runtime_error("no value for variable");
00111 }
00112
00117 bool Num::hasVariable() {
00118     return false;
00119 }
00120
00125 bool Add::hasVariable() {
00126     return this->lhs->hasVariable() || this->rhs->hasVariable();
00127 }
00128
00133 bool Mult::hasVariable() {
00134     return this->lhs->hasVariable() || this->rhs->hasVariable();
00135 }
00136
00141 bool Var::hasVariable() {
00142     return true;
00143 }
00144
00151 Expr* Num::subst(std::string stringInput, Expr* e) {
00152     return new Num(this->val);
00153 }
00154
00161 Expr* Add::subst(std::string stringInput, Expr* e) {
00162     Expr* newLHS = this->lhs->subst(stringInput,e);
00163     Expr* newRHS = this->rhs->subst(stringInput,e);
00164     return new Add(newLHS,newRHS);
00165 }
00166
00173 Expr* Mult::subst(std::string stringInput, Expr *e) {
00174     Expr* newLHS = this->lhs->subst(stringInput,e);
00175     Expr* newRHS = this->rhs->subst(stringInput,e);
00176     return new Mult(newLHS,newRHS);
00177 }
00178
00185 Expr* Var::subst(std::string stringInput, Expr *e) {
00186     if (this->var == stringInput) {
00187         return e;
00188     } else {
00189         return new Var(this->var);
00190     }
00191 }
00192

```

```

00197 void Num::print(std::ostream &stream) {
00198     stream << std::to_string(val);
00199 }
00200
00205 void Add::print(std::ostream &stream) {
00206     stream << "(";
00207     lhs->print(stream);
00208     stream << "+";
00209     rhs->print(stream);
00210     stream << ")";
00211 }
00212
00217 void Mult::print(std::ostream &stream) {
00218     stream << "(";
00219     lhs->print(stream);
00220     stream << "*";
00221     rhs->print(stream);
00222     stream << ")";
00223 }
00224
00229 void Var::print(std::ostream &stream) {
00230     stream << var;
00231 }
00232
00238 void Num::pretty_print(std::ostream &ot, precedence_t prec) {
00239     ot << val;
00240 }
00241
00247 void Add::pretty_print(std::ostream &ot, precedence_t prec) {
00248     bool needParens = prec > prec_add;
00249     if (needParens) ot << "(";
00250     lhs->pretty_print(ot, static_cast<precedence_t>(prec_add + 1));
00251     ot << " + ";
00252     rhs->pretty_print(ot, prec_add);
00253     if (needParens) ot << ")";
00254 }
00255
00261 void Mult::pretty_print(std::ostream &ot, precedence_t prec) {
00262     bool needParens = prec > prec_mult;
00263     if (needParens) ot << "(";
00264     lhs->pretty_print(ot, static_cast<precedence_t>(prec_mult + 1));
00265     ot << " * ";
00266     rhs->pretty_print(ot, prec_mult);
00267     if (needParens) ot << ")";
00268 }
00269
00275 void Var::pretty_print(std::ostream &ot, precedence_t prec) {
00276     ot << var;
00277 }

```

7.13 /Users/samanthapope/msdscriptRepo/msdScript/Expr.h File Reference

```

#include <string>
#include <stdexcept>
#include <sstream>

```

Classes

- class [Expr](#)
- class [Num](#)
- class [Add](#)
- class [Mult](#)
- class [Var](#)

Enumerations

- enum [precedence_t](#) { [prec_none](#) , [prec_add](#) , [prec_mult](#) }

7.13.1 Enumeration Type Documentation

7.13.1.1 precedence_t

enum [precedence_t](#)

Enumerator

prec_none	
prec_add	
prec_mult	

Definition at line 11 of file [Expr.h](#).

7.14 Expr.h

[Go to the documentation of this file.](#)

```

00001 // Expr.h
00002
00003 #ifndef CS6015PROJECT_EXPR_H
00004 #define CS6015PROJECT_EXPR_H
00005 #include <string>
00006 #include <stdexcept>
00007 #include <sstream>
00008
00009
00010
00011 typedef enum {
00012     prec_none,    // = 0
00013     prec_add,     // = 1
00014     prec_mult     // = 2
00015 } precedence_t;
00016
00017 class Expr {
00018 public:
00019     virtual ~Expr() = default; //virtual destructor - allows me to write tests using "delete" to test
00020     my DeepCopy (prevents memory links)
00021     virtual bool equals(Expr *e) = 0;
00022     virtual int interp() =0;
00023     virtual bool hasVariable()=0;
00024     virtual Expr* subst(std::string stringInput, Expr* e)=0;
00025
00026     std::string to_string() {
00027         std::stringstream st("");
00028         this->print(st);
00029         return st.str();
00030     }
00031
00032     std::string to_pp_string(){
00033         std::stringstream st("");
00034         this->pretty_print_at(st);
00035         return st.str();
00036     }
00037
00038     virtual void print(std::ostream& stream)=0;
00039
00040     virtual void pretty_print(std::ostream& ot, precedence_t prec)=0;
00041
00042     void pretty_print_at(std::ostream &ot){
00043         this->pretty_print(ot,prec_none);
00044     }
00045 };
00046
00047 class Num : public Expr {
00048 public:
00049     int val;
00050     Num(int val);
00051     bool equals(Expr *e) override;
00052     int interp() override;
00053     bool hasVariable() override;
00054     Expr* subst(std::string stringInput, Expr* e) override;
00055     void print(std::ostream& stream) override;

```

```

00055 protected:
00056     void pretty_print(std::ostream& ot, precedence_t prec) override;
00057 };
00058
00059 class Add : public Expr {
00060 public:
00061     Expr *lhs;
00062     Expr *rhs;
00063
00064     Add(Expr *lhs, Expr *rhs);
00065     bool equals(Expr *e) override;
00066     int interp() override;
00067     bool hasVariable() override;
00068     Expr* subst(std::string stringInput, Expr* e) override;
00069     void print(std::ostream& stream) override;
00070 protected:
00071     void pretty_print(std::ostream& ot, precedence_t prec) override;
00072
00073 };
00074
00075 class Mult : public Expr {
00076 public:
00077     Expr *lhs;
00078     Expr *rhs;
00079
00080     Mult(Expr *lhs, Expr *rhs);
00081     bool equals(Expr *e) override;
00082     int interp() override;
00083     bool hasVariable() override;
00084     Expr* subst(std::string stringInput, Expr* e) override;
00085     void print(std::ostream& stream) override;
00086 protected:
00087     void pretty_print(std::ostream& ot, precedence_t prec) override;
00088
00089 };
00090
00091 class Var : public Expr {
00092 public:
00093     std::string var;
00094     Var(const std::string& varPassed);
00095     bool equals(Expr *e) override;
00096     int interp() override;
00097     bool hasVariable() override;
00098     Expr* subst(std::string stringInput, Expr* e) override;
00099     void print(std::ostream& stream) override;
00100 protected:
00101     void pretty_print(std::ostream& ot, precedence_t prec) override;
00102
00103 };
00104
00105
00106 #endif //CS6015PROJECT_EXPR_H

```

7.15 /Users/samanthapope/msdscriptRepo/msdScript/ExprTest.cpp File Reference

```

#include "catch.h"
#include "Expr.h"
#include "cmdline.h"
#include <stdexcept>
#include <sstream>

```

Functions

- [TEST_CASE](#) ("Num equals tests", "[Num]")
- [TEST_CASE](#) ("Num Edge Cases")
- [TEST_CASE](#) ("Add equals tests", "Add")
- [TEST_CASE](#) ("Mult equals tests", "[Mult]")
- [TEST_CASE](#) ("Var equals tests", "[Var]")
- [TEST_CASE](#) ("interp tests", "All Expressions")

- [TEST_CASE](#) ("hasVariable tests", "All Expressions")
- [TEST_CASE](#) ("subst tests", "All Expressions")
- [TEST_CASE](#) ("to_string tests", "all expressions")
- [TEST_CASE](#) ("pretty_print_at Tests", "All expression classes")
- [TEST_CASE](#) ("Nabil's given tests")

7.15.1 Function Documentation

7.15.1.1 TEST_CASE() [1/11]

```
TEST_CASE (
    "Add equals tests" ,
    "Add" )
```

Definition at line 43 of file [ExprTest.cpp](#).

7.15.1.2 TEST_CASE() [2/11]

```
TEST_CASE (
    "hasVariable tests" ,
    "All Expressions" )
```

Definition at line 173 of file [ExprTest.cpp](#).

7.15.1.3 TEST_CASE() [3/11]

```
TEST_CASE (
    "interp tests" ,
    "All Expressions" )
```

Definition at line 105 of file [ExprTest.cpp](#).

7.15.1.4 TEST_CASE() [4/11]

```
TEST_CASE (
    "Mult equals tests" ,
    "" [Mult] )
```

Definition at line 65 of file [ExprTest.cpp](#).

7.15.1.5 TEST_CASE() [5/11]

```
TEST_CASE (
    "Nabil's given tests" )
```

Definition at line 352 of file [ExprTest.cpp](#).

7.15.1.6 TEST_CASE() [6/11]

```
TEST_CASE (
    "Num Edge Cases" )
```

Definition at line 30 of file [ExprTest.cpp](#).

7.15.1.7 TEST_CASE() [7/11]

```
TEST_CASE (
    "Num equals tests" ,
    " [Num] )
```

Definition at line 12 of file [ExprTest.cpp](#).

7.15.1.8 TEST_CASE() [8/11]

```
TEST_CASE (
    "pretty_print_at Tests" ,
    "All expression classes" )
```

Definition at line 315 of file [ExprTest.cpp](#).

7.15.1.9 TEST_CASE() [9/11]

```
TEST_CASE (
    "subst tests" ,
    "All Expressions" )
```

Definition at line 240 of file [ExprTest.cpp](#).

7.15.1.10 TEST_CASE() [10/11]

```
TEST_CASE (
    "to_string tests" ,
    "all expressions" )
```

Definition at line 298 of file [ExprTest.cpp](#).

7.15.1.11 TEST_CASE() [11/11]

```
TEST_CASE (
    "Var equals tests" ,
    " [Var] )
```

Definition at line 86 of file [ExprTest.cpp](#).

7.16 ExprTest.cpp

[Go to the documentation of this file.](#)

```

00001 //
00002 // Created by Samantha Pope on 1/22/24.
00003 //
00004
00005
00006 #include "catch.h"
00007 #include "Expr.h"
00008 #include "cmdline.h"
00009 #include <stdexcept>
00010 #include <sstream>
00011
00012 TEST_CASE("Num equals tests", "[Num]") {
00013     Num num1(5);
00014     Num num2(5);
00015     Num num3(10);
00016
00017     SECTION("Equal numbers") {
00018         REQUIRE(num1.equals(&num2));
00019     }
00020
00021     SECTION("Not equal numbers") {
00022         REQUIRE_FALSE(num1.equals(&num3));
00023     }
00024
00025     SECTION("Comparison with null") {
00026         REQUIRE_FALSE(num1.equals(nullptr));
00027     }
00028 }
00029
00030 TEST_CASE("Num Edge Cases"){
00031     SECTION("Negative Numbers"){
00032         Num num1(-5);
00033         Num num2(-5);
00034         Num num3(-10);
00035         Num num4(5);
00036         REQUIRE(num1.equals(&num2));
00037         REQUIRE_FALSE(num1.equals(&num3));
00038         REQUIRE_FALSE(num1.equals(&num4));
00039         REQUIRE_FALSE(num1.equals(nullptr));
00040     }
00041 }
00042
00043 TEST_CASE("Add equals tests", "Add") {
00044     Num n1(3), n2(4), n3(3), n4(4);
00045     Add add1(&n1, &n2);
00046     Add add2(&n3, &n4);
00047     Add add3(&n2, &n1); // Different order
00048     Num differentNum(7);
00049     Add differentAdd(&n1, &differentNum);
00050
00051     SECTION("Equal Adds") {
00052         REQUIRE(add1.equals(&add2));
00053     }
00054     SECTION("Equal Adds with different order") {
00055         REQUIRE(add1.equals(&add3));
00056     }
00057     SECTION("Not equal Adds") {
00058         REQUIRE_FALSE(add1.equals(&differentAdd));
00059     }
00060     SECTION("Comparison with null") {
00061         REQUIRE_FALSE(add1.equals(nullptr));
00062     }
00063 }
00064
00065 TEST_CASE("Mult equals tests", "[Mult]") {
00066     Num n1(2), n2(5), n3(2), n4(5);
00067     Mult mult1(&n1, &n2);
00068     Mult mult2(&n3, &n4);
00069     Mult mult3(&n2, &n1); // Different order
00070     Num differentNum(9);
00071     Mult differentMult(&n1, &differentNum);
00072     SECTION("Equal Mults") {
00073         REQUIRE(mult1.equals(&mult2));
00074     }
00075     SECTION("Equal Mults with different order") {
00076         REQUIRE(mult1.equals(&mult3));
00077     }
00078     SECTION("Not equal Mults") {
00079         REQUIRE_FALSE(mult1.equals(&differentMult));
00080     }
00081     SECTION("Comparison with null") {
00082         REQUIRE_FALSE(mult1.equals(nullptr));

```

```

00083 }
00084 }
00085
00086 TEST_CASE("Var equals tests", "[Var]") {
00087     Var varExpr1("x");
00088     Var varExpr2("x");
00089     Var varExpr3("y");
00090     Num numExpr(5);
00091     SECTION("Equal VarExprs with same name") {
00092         REQUIRE(varExpr1.equals(&varExpr2));
00093     }
00094     SECTION("Not Equal VarExprs with different names") {
00095         REQUIRE_FALSE(varExpr1.equals(&varExpr3));
00096     }
00097     SECTION("Not Equal when compared with different type (Num)") {
00098         REQUIRE_FALSE(varExpr1.equals(&numExpr));
00099     }
00100     SECTION("Not Equal when compared with nullptr") {
00101         REQUIRE_FALSE(varExpr1.equals(nullptr));
00102     }
00103 }
00104
00105 TEST_CASE("interp tests", "All Expressions") {
00106     SECTION("Num interp") {
00107         Num num1(5);
00108         REQUIRE(num1.interp() == 5);
00109         Num num2(-3);
00110         REQUIRE(num2.interp() == -3);
00111         Num num3(0);
00112         REQUIRE(num3.interp() == 0);
00113         Num num4(100);
00114         REQUIRE(num4.interp() == 100);
00115         Num num5(-50);
00116         REQUIRE(num5.interp() == -50);
00117     }
00118
00119     SECTION("Add interp") {
00120         Num num1(5);
00121         Num num2(10);
00122         Add add(&num1, &num2);
00123         REQUIRE(add.interp() == 15);
00124         Num num3(-5);
00125         Add addNegative(&num1, &num3); // 5 + (-5)
00126         REQUIRE(addNegative.interp() == 0);
00127         Num num4(0);
00128         Add add3(&num2, &num4); // 10 + 0
00129         REQUIRE(add3.interp() == 10);
00130         Num num5(-20);
00131         Add add4(&num5, &num5); // -20 + (-20)
00132         REQUIRE(add4.interp() == -40);
00133         Num num6(100);
00134         Add add5(&num6, &num3); // 100 + (-5)
00135         REQUIRE(add5.interp() == 95);
00136     }
00137
00138     SECTION("Mult interp") {
00139         Num num1(5);
00140         Num num2(3);
00141         Mult mult(&num1, &num2);
00142         REQUIRE(mult.interp() == 15);
00143         Num num3(-2);
00144         Mult multNegative(&num1, &num3); // 5 * (-2)
00145         REQUIRE(multNegative.interp() == -10);
00146         Num num4(0);
00147         Mult mult3(&num2, &num4); // 3 * 0
00148         REQUIRE(mult3.interp() == 0);
00149         Num num5(1);
00150         Mult mult4(&num5, &num2); // 1 * 3
00151         REQUIRE(mult4.interp() == 3);
00152         Num num6(-1);
00153         Mult mult5(&num6, &num3); // -1 * (-2)
00154         REQUIRE(mult5.interp() == 2);
00155     }
00156
00157     SECTION("Var interp throws exception") {
00158         Var varExpr("x");
00159         REQUIRE_THROWS_AS(varExpr.interp(), std::runtime_error);
00160         Var varExpr2("y");
00161         REQUIRE_THROWS_AS(varExpr2.interp(), std::runtime_error);
00162         Var varExpr3("var");
00163         REQUIRE_THROWS_AS(varExpr3.interp(), std::runtime_error);
00164         Var varExpr4("123");
00165         REQUIRE_THROWS_AS(varExpr4.interp(), std::runtime_error);
00166         Var varExpr5("testVar");
00167         REQUIRE_THROWS_AS(varExpr5.interp(), std::runtime_error);
00168     }
00169 }

```



```

00170 }
00171
00172
00173 TEST_CASE("hasVariable tests", "All Expressions"){
00174     SECTION("Num hasVariable") {
00175         Num num1(5);
00176         REQUIRE_FALSE(num1.hasVariable());
00177         Num num2(-3);
00178         REQUIRE_FALSE(num2.hasVariable());
00179         Num num3(0);
00180         REQUIRE_FALSE(num3.hasVariable());
00181         Num num4(100);
00182         REQUIRE_FALSE(num4.hasVariable());
00183         Num num5(-50);
00184         REQUIRE_FALSE(num5.hasVariable());
00185     }
00186
00187     SECTION("Add hasVariable") {
00188         Num num1(5);
00189         Num num2(10);
00190         Var var1("x");
00191         Add add1(&num1, &num2);
00192         REQUIRE_FALSE(add1.hasVariable());
00193         Add add2(&num1, &var1);
00194         REQUIRE(add2.hasVariable());
00195         Add add3(&var1, &var1);
00196         REQUIRE(add3.hasVariable());
00197         Num num3(0);
00198         Add add4(&num2, &num3);
00199         REQUIRE_FALSE(add4.hasVariable());
00200         Var var2("y");
00201         Add add5(&var1, &var2);
00202         REQUIRE(add5.hasVariable());
00203     }
00204
00205     SECTION("Mult hasVariable") {
00206         Num num1(5);
00207         Num num2(3);
00208         Var var1("x");
00209         Mult mult1(&num1, &num2);
00210         REQUIRE_FALSE(mult1.hasVariable());
00211         Mult mult2(&num1, &var1);
00212         REQUIRE(mult2.hasVariable());
00213         Mult mult3(&var1, &var1);
00214         REQUIRE(mult3.hasVariable());
00215         Num num3(0);
00216         Mult mult4(&num2, &num3);
00217         REQUIRE_FALSE(mult4.hasVariable());
00218         Var var2("y");
00219         Mult mult5(&var1, &var2);
00220         REQUIRE(mult5.hasVariable());
00221     }
00222
00223     SECTION("Var hasVariable") {
00224         Var varExpr1("x");
00225         REQUIRE(varExpr1.hasVariable());
00226         Var varExpr2("y");
00227         REQUIRE(varExpr2.hasVariable());
00228         Var varExpr3("var");
00229         REQUIRE(varExpr3.hasVariable());
00230         Var varExpr4("123");
00231         REQUIRE(varExpr4.hasVariable());
00232         Var varExpr5("testVar");
00233         REQUIRE(varExpr5.hasVariable());
00234     }
00235 }
00236 }
00237
00238
00239
00240 TEST_CASE("subst tests", "All Expressions") {
00241     Num num5(5);
00242     Num num10(10);
00243     Var varX("x");
00244     Var varY("y");
00245
00246     SECTION("Num subst") {
00247         Expr* result1 = num5.subst("x", &num10);
00248         REQUIRE(result1->equals(&num5));
00249         delete result1;
00250         Expr* result2 = num10.subst("y", &num5);
00251         REQUIRE(result2->equals(&num10));
00252         delete result2;
00253     }
00254
00255     SECTION("Add subst") {
00256         Add add1(&num5, &varX);

```

```

00257     Expr* result1 = add1.subst("x", &num10);
00258     Add expected1(&num5, &num10);
00259     REQUIRE(result1->equals(&expected1));
00260     delete result1;
00261     Add add2(&varX, &varY);
00262     Expr* result2 = add2.subst("x", &num5);
00263     Add expected2(&num5, &varY);
00264     REQUIRE(result2->equals(&expected2));
00265     delete result2;
00266 }
00267
00268 SECTION("Mult subst") {
00269     Mult mult1(&num5, &varX);
00270     Expr* result1 = mult1.subst("x", &num10);
00271     Mult expected1(&num5, &num10);
00272     REQUIRE(result1->equals(&expected1));
00273     delete result1;
00274     Mult mult2(&varX, &varY);
00275     Expr* result2 = mult2.subst("x", &num5);
00276     Mult expected2(&num5, &varY);
00277     REQUIRE(result2->equals(&expected2));
00278     delete result2;
00279 }
00280
00281 SECTION("Var subst") {
00282     Expr* result1 = varX.subst("x", &num5);
00283     REQUIRE(result1 == &num5); // Check for same pointer, not just equality
00284     Var varAnotherX("x");
00285     Expr* result2 = varAnotherX.subst("x", &varY);
00286     REQUIRE(result2 == &varY); // Again, checking pointers
00287     Expr* result11 = varX.subst("y", &num5);
00288     REQUIRE(result11->equals(&varX));
00289     delete result11;
00290     Expr* result22 = varY.subst("x", &num5);
00291     REQUIRE(result22->equals(&varY));
00292     delete result22;
00293 }
00294 }
00295 }
00296
00297
00298 TEST_CASE("to_string tests", "all expressions"){
00299     CHECK( (new Var("x"))->to_string() == "x" );
00300     CHECK( (new Add(new Num(1), new Num(2)))->to_string() == "(1+2)" );
00301     CHECK( (new Mult(new Num(3), new Num(4)))->to_string() == "(3*4)" );
00302     CHECK( (new Num(-5))->to_string() == "-5" );
00303     CHECK( (new Num(0))->to_string() == "0" );
00304     CHECK( (new Add(new Num(1), new Num(-2)))->to_string() == "(1+-2)" );
00305     CHECK( (new Mult(new Num(0), new Num(4)))->to_string() == "(0*4)" );
00306     CHECK( (new Mult(new Num(3), new Num(-4)))->to_string() == "(3*-4)" );
00307     CHECK( (new Add(new Num(-1), new Mult(new Num(2), new Num(-3)))->to_string() == "(-1+(2*-3))" );
00308     CHECK( (new Add(new Var("x"), new Mult(new Add(new Num(0), new Num(-5)), new
Var("y"))))->to_string() == "(x+((0+-5)*y))" );
00309     CHECK( (new Mult(new Num(0), new Num(-2)), new Add(new Num(-3), new Num(4)))->to_string()
== "((0+-2)*(-3+4))" );
00310     CHECK( (new Add(new Var("z"), new Num(-10)))->to_string() == "(z+-10)" );
00311     CHECK( (new Mult(new Add(new Var("a"), new Mult(new Num(-1), new Var("b"))), new Add(new Num(0),
new Var("c"))))->to_string() == "((a+(-1*b))*(0+c))" );
00312 }
00313
00314
00315 TEST_CASE("pretty_print_at Tests", "All expression classes"){
00316     // Test for simple addition
00317     std::stringstream ss3;
00318     (new Add(new Num(4), new Num(5)))->pretty_print_at(ss3);
00319     CHECK(ss3.str() == "4 + 5");
00320
00321     // Test for more complex expression with addition and multiplication
00322     std::stringstream ss4;
00323     (new Add(new Num(4), new Mult(new Num(5), new Num(6)))->pretty_print_at(ss4);
00324     CHECK(ss4.str() == "4 + 5 * 6");
00325
00326     // Test for nested expressions with multiple parentheses
00327     std::stringstream ss5;
00328     (new Add(new Mult(new Num(1), new Num(2)), new Add(new Num(3), new Mult(new Num(4), new
Num(5)))))>pretty_print_at(ss5);
00329     CHECK(ss5.str() == "1 * 2 + 3 + 4 * 5");
00330
00331     // Test to_pp_string method for complex nested expression
00332     CHECK((new Add(new Mult(new Num(1), new Add(new Num(2), new Num(3))), new Num(4)))->to_pp_string()
== "1 * (2 + 3) + 4");
00333
00334     // Test for multiplication precedence over addition
00335     CHECK((new Mult(new Add(new Num(2), new Num(3)), new Num(4)))->to_pp_string() == "(2 + 3) * 4");
00336
00337     // Test for deep nesting with multiple operations
00338     CHECK((new Add(new Num(2), new Mult(new Add(new Num(3), new Num(4)), new Num(5)))->to_pp_string()

```

```

    == "2 + (3 + 4) * 5");
00339
00340 // Test involving variables and operations
00341 CHECK((new Add(new Var("x"), new Mult(new Var("y"), new Num(2))))->to_pp_string() == "x + y * 2");
00342
00343 // Test for expression with multiple variables and numbers
00344 CHECK((new Mult(new Add(new Var("a"), new Var("b")), new Add(new Num(5), new
    Var("c"))))->to_pp_string() == "(a + b) * (5 + c)");
00345
00346 // Test for nested multiplications with a variable
00347 CHECK((new Mult(new Mult(new Var("z"), new Num(3)), new Add(new Num(4), new
    Num(5))))->to_pp_string() == "(z * 3) * (4 + 5)");
00348
00349 }
00350
00351
00352 TEST_CASE("Nabil's given tests") {
00353     SECTION("Given Tests Assignment 2") {
00354         CHECK((new Var("x"))->equals(new Var("x")) == true);
00355         CHECK((new Var("x"))->equals(new Var("y")) == false);
00356         CHECK((new Num(1))->equals(new Var("x")) == false);
00357         CHECK((new Add(new Num(2), new Num(3)))>equals(new Add(new Num(2), new Num(3))) == true);
00358         CHECK((new Add(new Num(2), new Num(3)))>equals(new Add(new Num(3), new Num(2))) == true);
00359     }
00360
00361     SECTION("Given Tests Assignment 3") {
00362         CHECK((new Mult(new Num(3), new Num(2)))
00363             ->interp() == 6);
00364         CHECK((new Add(new Add(new Num(10), new Num(15)), new Add(new Num(20), new Num(20))))
00365             ->interp() == 65);
00366         CHECK_THROWS_WITH((new Var("x"))->interp(), "no value for variable");
00367         CHECK((new Add(new Var("x"), new Num(1)))>hasVariable() == true);
00368         CHECK((new Mult(new Num(2), new Num(1)))>hasVariable() == false);
00369         CHECK((new Add(new Var("x"), new Num(7)))
00370             ->subst("x", new Var("y"))
00371             ->equals(new Add(new Var("y"), new Num(7))));
00372         CHECK((new Var("x"))
00373             ->subst("x", new Add(new Var("y"), new Num(7)))
00374             ->equals(new Add(new Var("y"), new Num(7))));
00375     }
00376
00377     SECTION("Given Tests for Assignment 4") {
00378         CHECK((new Num(10))>to_string() == "10");
00379
00380         // Create a stringstream to capture the output of pretty_print
00381         std::stringstream ssl;
00382         (new Add(new Num(1), new Mult(new Num(2), new Num(3))))>pretty_print_at(ssl);
00383         CHECK(ssl.str() == "1 + 2 * 3");
00384
00385         std::stringstream ss2;
00386         (new Mult(new Num(1), new Add(new Num(2), new Num(3))))>pretty_print_at(ss2);
00387         CHECK(ss2.str() == "1 * (2 + 3)");
00388
00389         CHECK ((new Mult(new Num(1), new Add(new Num(2), new Num(3))))->to_pp_string() == "1 * (2 +
00390 3)");
00391         CHECK ((new Mult(new Mult(new Num(8), new Num(1)), new Var("y"))->to_pp_string() == "(8 * 1
00392 * y)");
00393         CHECK ((new Mult(new Add(new Num(3), new Num(5)), new Mult(new Num(6), new
00394 Num(1))))->to_pp_string() ==
00395             "(3 + 5) * 6 * 1");
00396         CHECK ((new Mult(new Mult(new Num(7), new Num(7)), new Add(new Num(9), new
00397 Num(2))))->to_pp_string() ==
00398             "(7 * 7) * (9 + 2)");
00399     }
00400 }

```

7.17 /Users/samanthapope/msdscriptRepo/msdScript/main.cpp File Reference

```

#include <iostream>
#include "cmdline.h"
#include "Expr.h"

```

Functions

- `int main (int argc, const char *argv[])`

7.17.1 Function Documentation

7.17.1.1 `main()`

```
int main (
    int argc,
    const char * argv[ ] )
```

Definition at line 5 of file [main.cpp](#).

7.18 `main.cpp`

[Go to the documentation of this file.](#)

```
00001 #include <iostream>
00002 #include "cmdline.h"
00003 #include "Expr.h"
00004
00005 int main(int argc, const char * argv[]) {
00006     useArgs(argc, argv);
00007     return 0;
00008 }
00009
00010
```