Doc No: WG21 N4857

Title: WG21 Responses to National Body Comments, ISO/IEC PTDS 23619, C++ Extensions for Reflection

Date: Mar 3, 2020

Reply to the Attention of: Barry Hedquist, beh@peren.com

Attached please find the WG21 Responses to National Body Comments for ISO/IEC SC22 N5315, PDTS 23619, Information Technology -- Technical Specification -- C++ Extensions for Reflection.

Template for comments and secretariat observations

Date:2018-05-14	Document: SC22 N5315	Project: 23619

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
CA 3 001		02	Paragraph 1	ge	The use of WG 21 document N4750 as the base document implies that it, as modified by the instructions contained in the document under ballot, will be elevated in status to that of a Technical Specification.	Separately ballot N4750 as a Technical Specification if it would effectively be such.	Accept with Modification Resolved by instead using C++14 and the Coroutines TS as base documents.
CA 2 002		02	Paragraph 1	ed	WG 21 document N4750 is a working draft. ISO and IEC normatively referenced documents shall have reached at least the enquiry stage (as per the ISO/IEC Directives, Part 2:2018).	Rebase the Technical Specification on an appropriate document. For example, the DIS for the revision of ISO/IEC 14882 that is under way.	Accept with Modification Resolved by instead using C++14 and the Coroutines TS as base documents.
US 003		02	Paragraph 3	ed	The technical specification should be referred to as “this document” as opposed to “this Technical Specification”.	Use “this document” in place of “this Technical Specification”.	Accept
CH 004	7	04.03		te	No feature-testing macro has been provided to allow code to test the availability of this feature.	Consider adding a feature-test macro.	Accept with Modification See P1390R1
CH 005	21	06	1	te	Consider including namespace-alias in alias-declaration, such that reflect::Alias matches alias-declaration.	Include namespace-alias in alias-declaration. Adjust all uses of alias-declaration denoting a type.	Rejected An *alias-declaration* is not a *namespace-alias*.
CH 006	6	06	1	te	The newly introduced name “alias” is ambiguous as it might include namespace aliases.	Consider renaming "alias" to "type alias" to clearly disambiguate it from namespace aliases.	Rejected The term *alias* can also refer to variable aliases. Clarified by modifying the example. See P1390R1
US 007		06.02	Paragraph 1	ed	The only normative use of “static variable” in N4750 refers to a local static variable.	Use “variable of static storage duration” in place of “static variable”.	Accepted

CA 4 008		08		te	There is no update to the definition of “potentially constant evaluated” even though constant evaluation can be required for the determination of whether the meta-object type associated with a *reflexpr-qualifier* whose *reflexpr-operand* is an expression would satisfy `Constant`.	Update the definition of “potentially constant evaluated” to account for this situation, explaining the behaviour of cases like the following:
						`struct A {`
						`  A() = default;`
						`  template <typename T>`
							Accept with Modification
							See P1390R1

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat

						`  constexpr A(T &) {`
						`    static_assert(T::okay);`
						`  }`
						`};`
						`constexpr int foo(A) {`
						`  return 42;`
						`}`
						`auto *g(A a) {`
						`  return static_cast<`
						`      reflexpr(foo(a)) *`
						`      >(0);`
						`}`
						
						`template <reflect::Constant T>`
						`void f(T *);`
						
						`void bar() { f(g({})); }`	

PL 009		10.01.7.2 10.1.7.6		te	In [dcl.type.reflexpr]: "The type specified by the reflexpr-specifier is implementation-defined." – should we say "implementation-defined"? I understand this to mean "implementations must define what the type is", and that seems wrong. Same in [dcl.type.simple]: "For a reflexpr-operand x, the type denoted by reflexpr(x) is an implementation-defined type [...]".	Change “implementation-defined” to “unspecified”.	Accept with Modification See P1390R1
CH 010	25	10.01.7.2	1	te	The grammar seems to be missing a way to allow reflexpr(A::template B<C>), which could otherwise be valid and necessary if A is a dependent type and the containing template can be instantiated such that A is a class type and A::B names a static data member template.	Consider allowing it.	Accept
CA 5 011		10.01.7.2	Paragraph 1	te	The *reflexpr-operand* grammar is ambiguous between the *type-id* production and the *nested-name-specifier*$_{opt}$ *identifier* and *nested-name-specifier*$_{opt}$ *simple-template-id* productions. For example, an *identifier* may be a *class-name* and thus also a *type-id*. The latter two productions of *reflexpr-operand* are	Remove the latter two productions in favour of *id-expression*. Adjust Table 12 accordingly.	Accept

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					additionally problematic in that they are not established to be expression operands.		
CH 012	20	10.01.7.6		ed	Paragraph numbers are missing.	Add paragraph numbers.	Accept with Modification Added paragraph numbers both here and [reflect.general].
CH 013	19	10.01.7.6			There is no “function prototype scope”.	Replace “function prototype scope” by “function parameter scope”	Accept
CH 014	12	10.01.7.6		ed	Use of free text where grammar term was intended.	`parenthisized expression, function call expresion or functional type conversion expression.` should use the grammar terms, thus hyphenated and italics: *parenthesized expression, function-call-expression* or *functional-type-conv-expression*	Accepted with modification. Used the grammar term for *function-call-expression* and *functional-type-conv-expression*, but not parenthesized expression. The latter has precedent for not being used with italics.
CH 015	11	10.01.7.6		ed	parenthisized and expresion are misspelled.	Fix the spelling.	Accept
CA 12 016		10.01.7.6	Paragraph 1	te	There is no mention in the added subclause that a *reflexpr-operand* that is a constant expression produces a meta-object type that satisfies `Constant`.	Add a sentence indicating this fact to the new subclause in question.	Accept
CA 11 017		10.01.7.6	Paragraph 1	te	There is no mention in the added subclause that a *reflexpr-operand* in the form of a *decltype-specifier* produces a meta-object type that satisfies `Alias`.	Add a sentence indicating this fact to the new subclause in question.	Accept
CA 10 018		10.01.7.6	Paragraph 1	ed	The last sentence refers to `Alias` without qualifying it with `reflect`.	Replace “`Alias`” with “`reflect::Alias`”.	Accept

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	**Line number**	**Clause/ Subclause**	**Paragraph/ Figure/Table**	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
CA 9 019		10.01.7.6	Paragraph 1	te	The following wording implies a restriction on parenthesized expressions in general (without sufficient qualification): For a parenthesized expression `(E)`, whether or not itself nested inside a parenthesized expression, the expression `E` shall be either a parenthesized expression, a *function-call-expression* or a *functional-type-conv-expression*; otherwise the program is ill-formed.	Express the intended restriction using the following: For a *reflexpr-operand* that is a parenthesized expression `(E)`, `E` shall be a *function-call-expression*, *functional-type-conv-expression*, or an expression `(E′)` that satisfies the requirements for being an *reflexpr-operand*.	Accept with Modification See P1390R1
CA 8 020		10.01.7.6	Paragraph 1	ed	There is a typo: “parenthisized expression”.	Replace “parenthisized expression” with “parenthesized expression”.	Accept
CA 7 021		10.01.7.6	Paragraph 1	te	The directionality of the reflection-related relation is not consistent between the various bullets. Transformations applied to A yield B (but not vice versa) in at least the following cases: • A: variable, B: the variable’s type • A: enumeration, B: the underlying type • A: class, B: a base class • A: non-template alias, B: the designated entity Transformations applied to B yield A (but not vice versa) in at least the following cases: • B: parenthesized expression, A: the subexpression within the parentheses • B: *functional-type-conv-expression*, A: the type specified • B: *function-call-expression*, A: the function called • B: function, A: the return type • B: function, A: the type of a parameter	Ensure that every bullet is consistently written such that transformations applied to B yield A.	Accept with Modification See P1390R1

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					• B: function, A: the type of the function		
CA 6 022		10.01.7.6	Paragraph 1	ed	The added subclause should be referred to as a subclause (as opposed to a section).	Replace “section” with “subclause”.	Accept
US 023		10.01.7.6	Paragraph 1	ed	In bullet 1.13, add “a” before “parameter type”.	Say “[...] the return type, a parameter type, or the function type [...]”.	Accept
US 024		10.01.7.6	Paragraph 1	te	The intended meaning of “entity [...] at block scope” is unclear.	Use “local variable” and “local class”, etc. if the intent is to refer to such.	Accept with Modification See P1390R1

US 025		10.01.7.6	Paragraph 1	te	It is unclear whether the restriction on the *reflexpr-operand* in relation to block scope is meant to be a more general restriction that extends to entities and aliases that are reflection-related to the operand.	Add at least a note with an example that explains the rule and its operation in cases like the following:
						`void f() {`
						`  using namespace std::experimental::reflect;`
						
						`  enum class E : int { E0 };`
						`  using MyEnum0 = reflexpr(E); // ill-formed`
						`  using MyEnum = get_type_t<reflexpr(E::E0)>; // perhaps meant to be ill-formed?`
						`}`
							Accept with Modification
							See P1390R1

CH 026	26	17.07.2.1	09.10	te	A value-dependent constant expression is missing.	Add “or a value-dependent constant expression”	Accept

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2018-05-14	Document: SC22 N5315	Project: 23619

MB/NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
CA 13 027		17.07.2.1	Paragraph 1	te	The cases where the operand of reflexpr is a *function-call-expression* or *functional-type-conv-expression* are not addressed by the list.	Add a new bullet after 9.9: • denoted by `reflexpr`(*operand*), where *operand* is a type-dependent expression or a (possibly parenthesized) *functional-type-conv-expression* with at least one type-dependent immediate subexpression, or • […]	Accept
CA 14 028		20.05.1		ge	A freestanding implementation may claim conformance to the document under review without providing meaningful functionality.	Add `<experimental/reflect>` to the table of C++ headers for freestanding implementations.	Accept with Modification See P1390R1
CH 029	1	21.12		te	constexpr variables should be inline.	Consider making all `constexpr auto` variables inline constexpr variables.	Rejected *Inline* is not needed
CA 16 030		21.12		te	The synopsis presents comments indicating that certain concepts "refine" other concepts. These comments imply subsumption relationships; however, the definition of the concepts does not always support the existence of the implied subsumption relationships. For example, `Enumerator` is said to refine `Constant` and likewise `Variable` with respect to `ScopeMember`; however, in both cases, the detailed description does not indicate a subsumption relation between the concept being defined and the concept said to be refined. The mathematical truth of one concept being always satisfied when another concept is satisfied is insufficient to establish a subsumption relationship.	Make it so that the intended subsumption relationships are expressed by the detailed descriptions. Remove the comments or make it so that they minimally express all of the subsumption relationships. Note that the definition of `Constant` does not admit the subsumption relationship with `Enumerator`. An enumerator is not an expression (and therefore not a constant expression).	Accept with Modification See P1390R1
CA 15 031		21.12		te	Additional clarity is needed over whether member *using-declarations* are reflected by meta-object types satisfying `Alias` when applying transformations like `get_public_member_functions` upon a `Class`.	Add a statement to clarify.	Accept with Modification See P1390R1

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					Similarly, the same for an inherited constructor and `get_constructor` on a `FunctionalTypeConversion`.		
CH 032	5	21.12.02		te	constexpr variables of operations should be templated on the concept corresponding to the operation, instead of `<class T>`	Change `template <class T> constexpr auto is_public_v = is_public<T>::value;` to ` template <Object  T> constexpr auto is_public_v = is_public<T>::value;`; similar for many others.	Accept with Modification See P1390R1
CH 033	2	21.12.02		ed	The hierarchy of reflection concepts is not very accessible, despite being a key ingredient in the mental model.	Add a graphic representation of "inheritance" of concepts.	Reject No consensus for change.
CA 17 034		21.12.02		ed	The synopsis presents “forward declarations” of concepts, which is novel in the context of N4750.	Provide a definition of the concept in the synopsis in the style of [concepts.syn] in WG 21 document N4778.	Accept
PL 035		21.12.02		te	[reflect.synopsis], Concept Constructor refines Callable and Record Member. This means that there is no supported operation is_defaulted and is_implicitly_declared for Constructor type The aforementioned operations are defined for SpecialMemberFunction concept and indirectly supported in Destructor	Thus current definition for Constructor looks incomplete and it is proposed to add SpecialMemberFunction to a list of refined concepts to ad support for missing but valuable operations on a reflected constructor.	Reject See P1390R1
PL 036		21.12.02		te	The fundamental concept in this document is called Object. The term "object" has a defined meaning in the c++ standard [intro.object] different from that, and the proposal in the text part uses the term meta-object.	Change the concept name Object to Metaobject	Reject See P1390R1
JP 037		21.12.02	Para 1	ed	In header contents for 21.12.4.2, the last statement, “constexpr auto” for “unpack_sequence_t” seems to be typo, because it’s for “type”, not “value”.	It would be “using” instead of “constexpr auto”	Accept
CH 038	14	21.12.02, 21.12.3.10		ed	Synopsis and defining sub-clause are inconsistent: “// refines Named and Scope” versus “template <class T> concept Namespace = Scope<T> && see below ;“	Depending on the outcome of CH13, correct the defining sub-clause or correct the synopsis.	Accept

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat

CH 039	15	21.12.02, 21.12.3.9		te	Decide whether `Typed` should refine `Named`, as specified in the defining sub-clause, or `Object`, as shown in the comment in the synopsis	Change
						// refines Object
						to
						// refines Named
						in the synopsis, unless the defining sub-clause needs to be changed.
							Accept

CH 040	8	21.12.03		ed	It is surprising and potentially confusing to see e.g. `is_enum` which exists already in namespace `std`.	Consider adding a Note as written rationale for reusing the identifier, and their relation to the operation in namespace `std`.	Accept with Modification See P1390R1
CH 041	13	21.12.03.10		te	Namespace should refine Named: exposing their name is a key feature for reflection.	Change the definition of Namespace to template <class T> concept Namespace = Named<T> && Scope<T> && see below ;	Accept
CH 042	3	21.12.03.18		te	A base class has a name.	Consider making `Base` require `Named` instead of just `Object`. `get_name` could return the injected class name.	Reject A 'base' in this case has a class which is accessible with get_class. This class then has a name.
CH 043	17	21.12.03.6		te	Enumerator needs refinement regarding Constant (see also current inconsistency with synopsis).	Change template <class T> concept Enumerator = Typed<T> && ScopeMember<T> && see below ; to template <class T> concept Enumerator = ScopeMember<T> && Constant<T> && see below ;	Accept
CH 044	16	21.12.03.7		te	While the synopsis (correctly) says that `Variable` refines `Typed` and `ScopeMember`, the defining sub-clause only mentions `Typed`.	Change template <class T> concept Variable = Typed<T> && see below ; to template <class T> concept Variable = Typed<T>	Accept

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
						&& ScopeMember<T> && see below ; or adjust the synopsis.	
CA 19 045		21.12.04		te	Where NTBS is mentioned in the document under ballot, the encoding used for the string's value is unspecified.	Specify that the strings are first formed using the basic source character set (with *universal-character-name*s as necessary) then mapped in the manner applied to string literals with no encoding prefix in phases 5 and 6 of translation.	Accept with Modification See P1390R1
CA 18 046		21.12.04		te	Where NTBS is mentioned in the document under ballot, it is preferable that the string is in the initial shift state prior to the terminating NUL character.	Use NTMBS instead of NTBS.	Accept with Modification See P1390R1
PL 047		21.12.04		te	regarding ill-formed diagnostic should be required to fail early even if this costs compilation time. Without this it would be very easy to cause UB.	This part should sound like "If subsequent specializations of operations on the same reflected entity could give different constant expression results(...), the program is ill-formed."	Reject It is unclear how (or if) such a thing can be implemented. At any rate, this is something that can be added in the future if we get implementation experience that suggests this is possible and desirable.

CH
048
	10	21.12.04	3	ed	The comment has a line break.	Change
						`// ill-formed,`
						`no diagnostic required`
						to
						`// ill-formed,`
						`// no diagnostic required`
							Accept

CA 20
049
		21.12.04.1		te	The wording refers to "most recent". What is the "most recent" declaration in the following?
					`#line 6`
					`void g(char (*)[7]);`
					`class A {`
					`  static void f() {`
					`    void g(char (*)[GLine]);`
						Replace the mentions of "most recent" with something that is more well-defined (even if less specific).	Accept with Modification
							See P1390R1

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat

					`    g(0);`
					`  }`
					`  static void h();`
					`  static constexpr auto GLine = reflect::get_source_line_v<reflect::get_callable_t<reflexpr(::g(0))>>;`
					`};`
					`void A::h() { f(); }`		

CH 050	32	21.12.04.10	1	te	get_class_t<T> might satisfy Alias, because it is “alias to reflexpr(X)” (thus an immediate reflexpr invocation), as per 21.12.3.4/2. This was likely not the intent.	Consider rewording to “type is an alias to a meta-object type that reflects X”	Accept with Modification See P1390R1
CA 37 051		21.12.04.16	Paragraph 1	ed	The statement regarding conditions that render the program is ill-formed appears under the *Remarks* element in the description of other operations within the document under ballot.	Place the statement under a *Remarks* element.	Accept
CA 36 052		21.12.04.16	Paragraph 1	te	Guaranteed copy elision should be mentioned.	Mirroring the example in N4750 [dcl.init] paragraph 17 bullet 17.6.1, add an example clarifying that `get_constructor<get_subexpression_t<reflexpr((T(T(T()))))>>` is ill-formed because the *functional-type-conv-expression* does not perform overload resolution for a constructor.	Accept with Modification See P1390R1

CA 35 053		21.12.04.16	Paragraph 1	te	It is unclear whether get_constructor is expected to be a valid operation in cases where reference binding involves overload resolution for a constructor.	Clarify the status of the following:
						`struct S { S(const std::string &); };`
						
						`template <typename T>`
						`auto *f(T &&t) {`
						`  return (get_constructor_t<reflexpr(T{"Popeye"`
							Accept with Modification
							See P1390R1

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat

						`})> *)0;`
						
						`}`
						
						`bool g(const S &s) { return f(s); }`	

PL 054		21.12.04.18		te	is_override is defined as true if the respective member function is annotated with override and not when it is an actual override. The latter information seems unavailable. It seems more rational to be asking whether a function is an actual override.	Consider changing the meaning of is_override to reflect on actual overrides or introducing a second trait for that purpose.	Accept with Modification See P1390R1
CH 055	31	21.12.04.2	3	ed	Other sub-clauses mention “elements”, this one does not.	Rephrse the defining sub-clause of reflect::unpack_sequence to also use the word "elements".	Accept

CH 056	18	21.12.04.2	3	te	Unpack_sequence_t is a variable template initialized by a type.	In [reflect.synopsis], the template unpack_sequence_t is defined as follows:
						template <template <class...> class Tpl, ObjectSequence T>
						constexpr auto unpack_sequence_t = unpack_sequence<Tpl, T>::type;
						but according to [reflect.ops.objseq] p3:
						All specializations of unpack_sequence<Tpl, T> shall meet the TransformationTrait requirements (23.15.1). The nested type named type is an alias to the template Tpl specialized with the types in T.
						So it seems that what unpack_sequence_t actually defines would be a type alias instead of a variable template and should be corrected to
						template <template <class...> class Tpl, ObjectSequence T>
						using unpack_sequence_t = typename unpack_sequence<Tpl, T>::type;
							Accept with Modification
							See P1390R1

CH 057	28	21.12.04.3	02.1	te	Compilers generate “names” for unnamed entities, such as "(lambda at main.cpp:7:14)".	Consider allowing implementation-defined values for get_display_name on unnamed entities.	Reject Lambda objects are not unnamed entities.

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
							See P1390R1
CH 058	29	21.12.04.3	02.X	te	If get_name_v<reflexpr(n::A<int>)> is "A" for a class template A, the same should probably apply for an alias template or variable template A.	Consider specifying or clarifying the "name" of template specializations, for functions and variables.	Accept with Modification See P1390R1
CH 059	9	21.12.04.3	5	ed	Inconsistent indentation	Adjust indentation to match the surrounding text.	Accept
CA 32 060		21.12.04.3	Paragraph 2	te	There is no non-empty value specified for `get_name<T>` in the case of a literal operator.	Add a bullet to provide the preferred format for the case of a literal operator.	Accept with Modification See P1390R1
CA 31 061		21.12.04.3	Paragraph 2	te	Unlike for specializations of class templates and template functions, the case of an instantiated variable is not similarly handled.	Add a bullet for variable template specializations.	Accept with Modification See P1390R1
CA 30 062		21.12.04.3	Paragraph 2	ed	Bullet 2.4.12 refers to the unqualified name of a function parameter.	Replace "unqualified name" with "name" in said bullet.	Accept
CA 29 063		21.12.04.3	Paragraph 2	ed	Bullet 2.4.6 refers to Table 9; however, the correct table is Table 11.	Replace the reference to Table 9 with a reference to Table 11.	Accept
CA 28 064		21.12.04.3	Paragraph 2	te	Bullet 2.4.6 refers to *simple-type-specifiers* (not the grammar term, which does not end with an 's') as a kind of type.	Replace "all other *simple-type-specifiers*" with "a cv-unqualified fundamental type other than `std::nullptr_t`".	Accept with Modification See P1390R1
CA 27 065		21.12.04.3	Paragraph 2	te	Bullet 2.4.1 addresses neither the case of a `typedef` declaration nor the case of an *alias-declaration*. The spurious "alias" at the end of the bullet indicates possible accidental truncation.	Replace "a type name introduced by a *using-declaration*, alias" with "the type name introduced by a *using-declaration*, *alias-declaration*, or a `typedef` declaration".	Accept with Modification See P1390R1

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Date:2018-05-14	Document: SC22 N5315	Project: 23619

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
CA 26 066		21.12.04.3	Paragraph 2	te	In bullet 2.4.1, a template *type-parameter* is not an alias (see [basic]).	Split the template *type-parameter* case out to a separate bullet.	Accept with Modification See P1390R1
CA 25 067		21.12.04.3	Paragraph 2	ed	Bullet 2.4.1 for the case of "`T` reflecting an `Alias`" is not meant for the case where `T` reflects a meta-object type satisfying the `Alias` concept.	Replace "`T` reflecting an `Alias`" with "`T` reflecting an alias ([basic])".	Accept

CA 24 068		21.12.04.3	Paragraph 2	te	It is unclear what string is associated with get_name as applied to a meta-object type reflecting upon a specialization of a conversion function template.	Add an example describing the string associated with `s` in the following:
						`struct A { template <typename T> operator T *(); };`
						`const auto &s = get_name_v<get_callable_t<reflexpr (A().operator int *())>>;`
							Accept with Modification
							See P1390R1

CA 23 069		21.12.04.3	Paragraph 2	ed	For bullet 2.4, the additional clarity of a string's value being a representation of an *identifier* would be useful in the cases where that would be the case.	Place the bullets that specify the string's value to be a representation of an *identifier* under a separate bullet. Have that bullet indicate that the string's value is a representation of an *identifier*.	Reject No consensus for this change.
CA 22 070		21.12.04.3	Paragraph 2	ed	The sub-bullets under bullet 2.4 do not describe disjoint cases and bullet 2.4 does not exclude unnamed entities from its scope.	Reorganize the bullets in paragraph 2 to avoid implicit ordering while minimizing the need to introduce explicit ordering.	Accept with Modification See P1390R1
CA 21 071		21.12.04.3	Paragraph 2	ed	Bullet 2.3 has text reading: "an array, pointer, reference of function type, or a cv-qualified type". This is presumably meant to cover cv-qualified types and compound types aside from classes, unions, and enumerations; in which case, the "of" in "reference of function type" is a typographical error, and pointer-to-member types should be	Bullet 2.3 can be made redundant with bullet 2.5 by changing bullet 2.4 to specify cv-unqualified on each case of `T` reflecting a type. Apply said change to bullet 2.4 and remove bullet 2.3.	Accept with Modification See P1390R1 Non-editorial

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					mentioned.		
CH 072	23	21.12.04.5		te	The name of the operation is_class clashes with many other usages, e.g. in namespace std. is_class and is_struct suggest a difference in the underlying C++ entity.	Rename the operations is_class, is_struct into uses_class_key, uses_struct_key, respectively.	Accept with Modification See P1390R1
US 073		21.12.04.5	Paragraph 7	ed	The description of is_enum and is_union contains confusing text such as "an enumeration type (a union)".	Follow the style used for is_class and is_struct, i.e.: If T reflects an enumeration type (for is_enum<T>) or a union type (for is_union<T>) the base characteristic is [...].	Reject No consensus for change.
CH 074	27	21.12.04.6		te	10.1.7.6 states that a reflected template type-parameter satisfies both reflect::Type and reflect::Alias. Alias refines ScopeMember; but the operation get_scope_t<X> for X reflecting a template type-parameter is not specified.	Specify it, likely representing the template class specialization of the reflected template type-parameter.	Accept with Modification See P1390R1
US 075		21.12.04.6	Paragraph 2	te	It is unclear whether the parenthetical "(for the function's parameters)" is intended to be restrictive. In other words, it is unclear whether the function scope is only to be considered when T reflects a function parameter.	Remove the parenthetical. If it is the case that get_scope would only yield a Scope reflecting a function scope when applied to a meta-object type reflecting upon a function parameter, then add a clearly non-normative note to that effect.	Accept with Modification See P1390R1
CA 34 076		21.12.04.7	Paragraph 2	ed	The description folds multiple cases together using parentheses like in the following: "all data (function, including constructor and destructor) members". This use of parentheses does not help clarity.	Split the description (in all of the affected bullets) so that the parentheses are not necessary. Alternatively, formulate the description with the use of "respectively".	Reject The text is clear as written.
CA 33 077		21.12.04.7	Paragraph 2	ed	It would be more clear to indicate that members inherited from a base class are also considered.	Replace "subset of non-template members" with "subset of (possibly inherited ([class.derived])) non-template members".	Accept with Modification See P1390R1
CH 078	30	21.12.04.7 and many others		te	"The nested type named type is an alias to an ObjectSequence specialized with..." is incorrect, since the meta-object types are not template arguments to ObjectSequence.	Consider: "The nested type named type is an alias to a meta-object type satisfying ObjectSequence, containing elements which reflect..." The phrase "containing elements" matches with the phrase "element[s] in" used to define reflect::get_size and reflect::get_element.	Accept with Modification See P1390R1

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
CH 079	4	21.12.04.9		te	No interface exists to determine whether a Variable has thread storage duration.	Consider adding `is_thread_local<Variable>`.	Accept with Modification See P1390R1

CH 080	24	21.12.04.9	4	te	Is the following ill-formed?
					Namespace A {
					  int x;
					  int& ref = x;
					  using MetaRef = reflexpr(ref);
					  constexpr auto val =
					reflect::get_pointer_v<MetaRef>;
					}
						Specify what happens for reference types. Possibly, to clarify this comment: “reflect::get_pointer<T> is ill-formed if T reflects a reference and the name of the reference in a context with no lvalue-to-rvalue conversion would not be a constant expression.”	Accept with Modification
							See P1390R1

US 081		21.12.04.9	Paragraph 4	ed	There are no normative uses of “member variable” in N4750.	Use “non-static data member” in place of “member variable”.	Accept
CH 082	22	Annex C		te	The new keyword needs to be mentioned in Annex C.	Edit paragraph 1 in C.5.1 Clause 5: lexical conventions [diff.cpp17.lex] After "The concept keyword is added to enable the definition of concepts (12.6.8)." append " The reflexpr keyword is added to introduce meta-data through a reflexpr-specifier." And s/Valid ISO C++ 2017 code using concept or requires as an identifier/Valid ISO C++ 2017 code using concept , requires, or reflexpr as an identifier/	Accept with Modification See P1390R1

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
GB 083		Annex C	05.1	Ge	Add the new keyword, reflexpr, to the list in C.5.1 in the base document	Add an edit for paragraph 1 in C.5.1 Clause 5: lexical conventions [diff.cpp17.lex] After "The concept keyword is added to enable the definition of concepts (12.6.8)." append " The reflexpr keyword is added to introduce meta-data through a reflexpr-specifier." And s/Valid ISO C++ 2017 code using concept or requires as an identifier/Valid ISO C++ 2017 code using concept, requires, or reflexpr as an identifier/	Accept with Modification See P1390R1
GB 084		Classes	10.1.7.6	Te	The final paragraph 10.1.7.6 (final paragraph) If the reflexpr operarand … or function prototype scope (6.3.4)" The base document does not describe “function prototype scope” but 6.3.4 [basic.scope.param] but it does describe "function parameter scope".	Replace "function prototype scope" with "function parameter scope"	Accept with Modification See P1390R1
CA 1 085		General		te	Whether the ability to reflect upon `std::align_val_t` without including `<new>` is intended or not should be made clear.	Explicitly address cases like the following: `#include <experimental/reflect>` `#include <stdio.h>` `#include <stdlib.h>` `struct A { template <typename T> operator T(); };` `template <typename T>` `A::operator T() {` `using std::experimental::reflect;` `printf("%s\n", get_name_v<get_scope_t<get_aliased_t<reflexpr(T)>>>);` `exit(0);` `}` `int main(void) {` `(void) ::operator new(sizeof(int), A());` `}`	Accept with Modification See P1390R1

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

Template for comments and secretariat observations

Date:2018-05-14	Document: SC22 N5315	Project: 23619

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
GB 086		General	04.2	Te	The TS should define a feature test macro.	I propose __cpp_reflect as a suitable name and 201811 as a suitable value. Insert new 4.3 between 4.2 and 4.3: 4.3 Feature-testing recommendations [intro.features] 1 An implementation that provides support for this Technical Specification shall define the feature test macro(s) in Table 2. Table 2 — Feature-test macro(s) Macro name Value __cpp_reflect 201811	Accept with Modification See P1390R1

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat

ISO_IEC PDTS 23619 - JTC001-SC22-N5315_ANSI.docx: Collation successful

ISO_IEC PDTS 23619 - JTC001-SC22-N5315_BSI.doc: Collation successful

ISO_IEC PDTS 23619 - JTC001-SC22-N5315_JISC.doc: Collation successful

ISO_IEC PDTS 23619 - JTC001-SC22-N5315_PKN.doc: Collation successful

ISO_IEC PDTS 23619 - JTC001-SC22-N5315_SCC.doc: Collation successful

ISO_IEC PDTS 23619 - JTC001-SC22-N5315_SNV.doc: Collation successful

Collation of files was successful. Number of collated files: 6

SELECTED (number of files): 6

PASSED TEST (number of files conformed to CCT table model): 6

FAILED TEST (number of files conformed to CCT table model): 0

CCT - Version 2018.2

1 **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)
2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial