Document No: WG21 N4550
Date: 2015-07-25
References: ISO/IEC PDTS 19217
Reply To: Barry Hedquist <beh@peren.com>
INCITS/PL22.16 IR

Record of Response: National Body Comments

ISO/IEC PDTS 19217

Technical Specification: C++ Extensions for Concepts

Attached is WG21 N4550, Record of Response to National Body Comments for ISO/IEC PDTS 19217, Technical Specification – C++ Extensions for Concepts.

Document numbers referenced in this document are from WG21 unless otherwise stated.

ISO/IEC PDTS 19217, NB comments and secretariat observations

Date:2015-07-25	Document: SC22/WG21 N4550	Project: ISO/IEC PDTS 19217

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
FI 1	page v			ed	Page v has an empty "list of figures".	Remove the empty list of figures.	ACCEPTED
FI 13				ge	Insert/delete markers are inconsistently applied. For example, in 5.1.2, a full paragraph is marked as inserted, but other such situations are not marked accordingly.		ACCEPTED

NL.1					It should be possible to use a concept where ever auto can be used, constraining the set of possible types matched.
					More specifically, the two examples in 7.1.6.4 on page 13:
					C z = 0; // error: constrained-type-specifier in declaration of z
					auto cf() -> C; // error: constrained-type-specifier declared in return type of cf
					should be well formed.
							ACCEPTED

US 1				ge	It is unclear to us whether or not proper implementation experience can be gained without also specifying concepts for the standard library along with the core language facility.		REJECTED Adding concepts to the standard library will be a large part of the experience used to test the design presented in the TS.
US 2				ge	We would have liked to have seen the wording flushed through Core a few more times before moving it to a PDTS, as Core was still uncovering significant issues in each review.		REJECTED The specification is good enough to allow implementation and the disadvantages of further delay outweigh the possible incremental improvement.
US 3				ge	In [dcl.spec.auto][6] allow the two examples that are disallowed: C z = 0; // error: constrained-type-specifier		ACCEPTED

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
					in declaration of z auto cf() -> C; // error: constrained-type-specifier declared in return type of cf		
US 4				ge	We have a broad concern that the ambiguity for a reader between a constrained function template without a template-introducer and a 'regular' function will make the language unnecessarily difficult to teach, read, and maintain code. We note that a TS is the perfect vehicle to have an experiment to establish if these concerns are real, but want to exercise caution as we proceed, and be sure that there is a real feedback plan in place before considering moving this feature from a TS and into a future standard.		REJECTED Not a suggestion for change. An issues list will be maintained by the project editor.
US 5				ge	There are too many redundant ways to express the same set of requirements. While each presents a reasonable use-case in isolation, the combined effect is overwhelming.	Review the overlapping syntaxes, and eliminate those that add least value, or are least frequently used. This may mean shipping the TS in close to its current form to obtain such feedback though.	REJECTED As noted, the TS is the appropriate vehicle for determining the validity of this concern.
US 6				ge	We have a broad concern that it is hard to understand the feature purely from the specification, especially the subsumption rules, and equivalence rules to know when two signatures declare the same function or are ambiguous equally constrained overloads, yet there is a lack of readily available implementations to test our understanding against. While the feature set of the TS looks good, we think one more iteration on the specification would be useful.	Recast the rules for subsumption as a mini- grammar (distinct from the C++ grammar) as the English text appears to be trying to describe a grammar, but less formally, which leads to a potential lack of precision, and more confusion for the reader. We are not highlight specific lack of precision at this time, as we have not emerged from confusion in time to file appropriate comments.	REJECTED There was no consensus for a change at this time.
FI 2		1.1	p3	ed	1.1p3 talks about C++ 14882:2017, which doesn't	Change to "is planned to be included in the next	ACCEPTED

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					exist, and might not come into existence.	revision of the C++ standard".	
GB 1		1.1 [scope.intro]		Ed	"this feature ... is present in ISO/IEC 14882:2017" - cannot refer to a (proposed) future standard in this manner.	Elide ", but it is present in ISO/IEC 14882:2017"	ACCEPTED WITH MODIFICATION The wording suggested by FI 2 was adopted.
US 7	1	1.1	1	ge	A concepts Technical Specification lacking a concept-enabled standard library provides little value to either the C++ community or the committee itself. End users will see none of the benefits from the introduction of concepts unless the standard library uses concepts throughout. Moreover, it means that the Technical Specification will not serve it's primary purpose of building real-world experience with real (non- expert) programmers, and we will not even have properly gone through the exercise of trying to use this new feature to specify our own standard library.	Introduce concepts and constraints for the C++ Standard Library at the same time as the language feature designed specifically for that purpose.	REJECTED Adding concepts to the standard library will be a large part of the experience used to test the design presented in the TS.
FI 3		1.5	p2	ed	typo: "Technical Specification" should have uppercase letters.		ACCEPTED
FI 4		1.5	p2	ed	"standard feature": This is a TS, so it cannot specify a standard feature.		ACCEPTED WITH MODIFICATION The wording in question was replaced with suggested wording from WG21 SG10.
FI 5		1.5	p2	ed	Rephrase to avoid the one-line table. We only have a single feature, so no need for extra generality such as "a new standard feature".		REJECTED Having a standardized format across TSes is regarded as more important than avoiding a table with only one entry.
FI 6		1.6		ed	Add a cross-reference to Douglas Gregor's earlier concepts work, which is assumed to have helped shaping the approach in N3351.		REJECTED No other TSes have "related work" sections or non-normative references.

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
US 8		5	p12	ed	Duplicate ‘the the’ at the end of the example.	It may be replaced later through the the explicit specification of template arguments.	ACCEPTED

CA 12	n/a	5.1.1	Para 12 example	te	This example includes the following code:
					struct S1 { int n; };
					auto::* p1 = &S1::n;
					Is this intended to be valid as written? It seems like the ‘auto’ is intended to be a placeholder for the class name in a pointer-to-member type, but where is the type of the member? Is there a rule that says that a single ‘auto’ can simultaneously stand for the class type, and the member type?
					Or was the example perhaps meant to be:
					int (auto::*p1) = &S1::n;
						Replace the line
						auto::* p1 = &S1::n;
						with
						int (auto::*p1) = &S1::n;
						and similarly for other lines in the example where a placeholder appears in the place of a class name for a pointer-to-member, without the type of the member (or a placeholder for it) appearing anywhere.
							ACCEPTED WITH MODIFICATION
							The parentheses in the suggestion were omitted.

FI 10		5.1.1	p12	te	The provisions about "placeholder type" together with the note are confusing. It seems some of the provisions are intended to apply only for "auto" and others to also apply to constrained-type-name.		ACCEPTED
FI 11		5.1.1	p12	ed	(example) There is no S1::c member.		ACCEPTED
FI 12		5.1.1	p12	te	(example, last line): It's unclear whether the formation of "D::*" is ill-formed, or the initialization as a whole. Initializing with nullptr might help.		ACCEPTED WITH MODIFICATION All the variable declarations should have types.
FI 9		5.1.1	p12	te	"The replacement type ... shall be a class or enumeration type.": It is unclear whether a violation immediately causes a program to be ill-formed, or whether the usual rules for deduction failure apply.	Suggestion: "If the replacement type ... is not a class or enumeration type, type deduction fails." In any case, this provision should move to the section about template argument deduction.	REJECTED The context determines whether a violation is a SFINAE failure or a hard error.
US 9		5.1.1		ed	When augmenting an existing grammar term, list the existing terms as well as the new additions, so that the context is clear, and it cannot be confused as a replacement.	Provide a complete grammar (with insert annotations) for primary-expression	ACCEPTED
US 10		5.1.1	p8	ed	When augmenting an existing grammar term, list the existing terms as well as the new additions, so that	Provide a complete grammar (with insert annotations) for nested-name-specifier	ACCEPTED

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
					the context is clear, and it cannot be confused as a replacement.		
CA6	n/a	5.1.2	Para 5 example	te	we state "the generic lambda gl and the function object fun" have equivalent behaviour, but the lambda has no return statement while the function does	Change the lambda's body to: return a = *b;	ACCEPTED
US 11		5.1.2	p5	ed	Rephrasing generic lambdas lost a little detail in its terseness that should probably be preserved.	The closure type for a generic lambda has a public inline function call operator member template (14.5.2) that is ...	ACCEPTED
US 12		[expr.prim. req] (5.1.4)	3	ed	Some words seem transposed.	Change "to in order" to "in order to".	ACCEPTED
US 13		[expr.prim. req] (5.1.4)	4	te	The following requirement seems overly restrictive, as it can be fairly easily (but tediously) be worked around: "A requires-expression shall appear only within a concept definition (7.1.7), or within the requires-clause of a template- declaration (Clause 14) or function declaration (8.3.5)." (The tedious workaround for each concept C is to define an overload set consisting of two function templates, one unconstrained and returning false, the other constrained by C and returning true.)	Eliminate the requirement, thereby permitting other uses for this new kind of expression of type bool. (For example, requires-expressions might replace many or all of the Boolean type traits.) Additionally, in any context where a bool value is permitted, allow a concept's name plus suitable arguments to denote the truth value of the claim that "this combination of arguments satisfy this concept." (This syntax is currently valid in only certain contexts such as requires-expressions.)	REJECTED There was no consensus for a change at this time, but an issue will be opened for future consideration by WG21's Evolution Working Group.
US 14		[expr.prim. req] (5.1.4)	7	te	The following Note seems to specify a normative requirement rather than a clarification: "[Note: But if the substitution of template arguments into a requirement would always result in a substitution failure, the program is ill-formed; no diagnostic required (14.7). — end note]"	Strike the Note delimiters, thus elevating this specification to normative text.	ACCEPTED
US 15		5.1.4.3 14.10.1.7		te	We believe that (generally speaking) the non-throwing of exceptions is a part of the runtime contract of a function, not something that should be advertised in the type system outside a few very specific cases related to move operations. As a 'requires' expression is always free to invoke the 'noexcept' operator to produce a predicate,	Simplify the compound-requirement: term in 5.1.4.3: { expression } noexceptopt trailing-return-typeopt	REJECTED There was no consensus for a change at this time, but an issue will be opened for future consideration by WG21's Evolution Working Group.

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
					we believe that is sufficient support for exception constraints in the language, and directly supporting this additional term in the grammar would be harmful, encouraging compile-time contracts taking away an important library implementer freedom. As the TS is intended to provide feedback, we believe it would be better to proceed without this, and see how much demand arises from using the alternate form, and whether that alternate form alone is too cumbersome for real world use.	Strike 14.10.1.7.	
US 16		5.1.4.3		te	If we retain exception constraints, the optional noexcept specifier should support the full range of the noexcept grammar	Amend compound-requirement: : { expression } noexceptopt noexcept-specificationopt trailing-return-typeopt	REJECTED There was no consensus for a change at this time, but an issue will be opened for future consideration by WG21's Evolution Working Group.
FI 8		5.5.1	p8	ed	5.5.1p8 omits underlining for the grammar changes, and the introductory sentence does not (but should) mention constrained-type-name, too.		ACCEPTED
US 17		7.1		ed	When augmenting an existing grammar term, list the existing terms as well as the new additions, so that the context is clear, and it cannot be confused as a replacement.	Provide a complete grammar (with insert annotations) for decl-specifier	ACCEPTED
CA7	n/a	7.1.6.2	Para 2	te	we state "The auto specifier and constrained-type-specifiers are placeholders for values (type, non-type, kind)" – should that be "(type, non-type, template)" instead? same in the table below		ACCEPTED
FI 7		7.1.6.2	p1	ed	7.1.6.2p1 talks about "Table 10", but then, a "table 2" follows.		ACCEPTED
US 18		7.1.6.2		ed	When augmenting an existing grammar term, list the existing terms as well as the new additions, so that the context is clear, and it cannot be confused as a	Provide a complete grammar (with insert annotations) for simple-type-specifier	ACCEPTED

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:2015-07-25	Document: SC22/WG21 N4550	Project: ISO/IEC PDTS 19217

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					replacement.		
US 19		7.1.6.2	Table 2	ed	Table 2 should be numbered the same as the table it updates, to avoid confusion.	Renumber Table 2 as Table 10	ACCEPTED
US 20		7.1.6.2	Table 2	ed	When updating an existing table, display sufficient of the existing contents to provide context, as well as the new additions, so that the edit cannot be confused as a replacement.	Add several of the surrounding rows of Table 2 (10 in C++14), or reprint the whole table with the mark-up grammar highlighting the new row.	ACCEPTED
GB 2		7.1.6.4 [dcl.spec.auto]	P6	Ed	There is an incorrect example: void (auto::*)(auto) p1 = &Size<0>::f; Additionally the comment refers to p and not **p1** (and the new example comment also refers to p, not **p2**)	Correct to: void (auto::* **p1**)(auto) = &Size<0>::f; And make the variable name in the two comments match the example	ACCEPTED
US 21		7.1.6.4		ed	The ‘and’ in the first sentence could confusingly bind two ways.	Given: The auto and decltype(auto) type-specifier s and constrained-type-specifier s designate rewrite as: The constrained-type-specifier s and the auto and decltype(auto) type-specifier s designate	ACCEPTED
US 22		[dcl.spec. auto] (7.1.6.4)	above 1	ed	The phrase “the meaning of constrained-type-specifiers are described” seems grammatically incorrect.	Replace “are” by “is”.	ACCEPTED
CA8	n/a	7.1.6.4.2	Para 1 example	te	the declaration of “C3” is missing the “concept bool”	Change the declaration of C3 to: template <template<typename> class X> concept bool C3 = false;	ACCEPTED
FI 14		7.1.7	p1	te	It seems unfortunate that concepts cannot be declared as members of class templates. This seemingly makes it impossible to define concepts for constraining multiple template parameter packs (if concepts as static member functions were possible,		REJECTED There was no consensus for a change at this time, but an issue will be opened for later consideration by

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
					one could provide e.g. two packs so that the class template gets the first pack and the member function template gets the second. That can't be done with a namespace-scope concept because multiple packs in function templates require deduction, and concepts don't take arguments that could be deduced.). As an example, practical needs for such constraining arise in standard library implementations, when constraining the variadic converting constructors of std::tuple.		WG21's Evolution Working Group.
US 23		7.1.7		te	Using 'concept' as a decl-specifer, rather than forming a first class entity like a type or template, makes the feature appear more complex than it needs to be. Concepts would be simpler (for user and [we believe] the specification) if there was only one kind, rather than both function and variable syntax; the 'bool' keyword would become redundant and the set of restrictions on concepts based on them being functions or variables would disappear.	We will provide a paper in time for the Lenexa pre- meeting mailing proposing a grammar that would give all concepts the form: template <typename T> concept C = predicate; where 'predicate' is a compile-time evaluated Boolean expression.	REJECTED There was no consensus for a change at this time, but an issue will be opened for future consideration by WG21's Evolution Working Group.
US 24		[dcl.spec. concept] (7.1.7)	1, 5, 6	te	The syntactic distinction between a function concept and a variable concept seems to serve no useful purpose. A single concept syntax seems sufficient, and especially so once redundant elements are removed.	Merge the two concept forms into one, streamlining the syntax by eliminating at least the following redundant elements: explicit bool (see comment below), explicit return, and the always- empty parentheses constituting the function parameter list.	REJECTED There was no consensus for a change at this time, but an issue will be opened for future consideration by WG21's Evolution Working Group.
US 25		[dcl.spec. concept] (7.1.7)	5.2, 6.1	te	Since a concept's type always must be bool, there seems little reason to require the source code to say so explicitly. Typing concept should be sufficient without also typing bool immediately afterward.	Allow the compiler to supply bool (a) as the implicit return type for a function concept and (b) as the implicit type for a variable concept. (Note: this comment is implicitly accepted if the previous comment is accepted.)	REJECTED There was no consensus for a change at this time, but an issue will be opened for future consideration by WG21's Evolution Working Group.
US 26		8.3.5	p1/2	ed	The extra line-breaks confuse the grammar as if it	Reflow the text, moving more terms up to the first line, so that the whole term clearly presents	ACCEPTED

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
					had an alternate production.	on just two lines.	
US 27		[dcl.fct] (8.3.5)	end of 21	ed/te	The last part of the specification "… where T is the template parameter invented for head and U is the template parameter invented for U …" seems erroneous.	Replace "invented for U" by "invented for tail".	ACCEPTED
CA9	n/a	14.1	Para 10 example	te	should the comment "associates C1<T>…" read "associates C1<T> && …"?		ACCEPTED WITH MODIFICATION Necessary parentheses were added.
US 28		14.1	p1	ed	When augmenting an existing grammar term, list the existing terms as well as the new additions, so that the context is clear, and it cannot be confused as a replacement.	Provide a complete grammar (with insert annotations) for template-parameter	ACCEPTED
US 29		14.1	p1	te	constrained-parameter allows parameter pack with default argument, and it is it not clear what that should mean.	constrained-parameter: qualified-concept-name ...opt identifieropt default-template-argumentopt qualified-concept-name identifieropt default-template-argumentopt	ACCEPTED WITH MODIFICATION The suggested grammar is ambiguous and has been disambiguated.
US 30		[temp.param] (14.1)	bullet (10.3)	ed	There seems to be an article missing in the phrase "If C is variable concept …"	Insert the article "a" before "variable".	ACCEPTED

GB 3		14.6.4 [temp.friend]	P10	Ed	Three examples missing a return type:
					template<C1 T> g0(T);
					template<C1 T> g1(T);
					template<C2 T> g2(T);
						Add void return type to each example	ACCEPTED

US 31		14.10.1	p2	ed	Duplicate word 'the the' after the note.	Determining if a constraint is satisfied entails the the substitution of template arguments into that constraint	ACCEPTED
CA1	N/A	14.10.1.1	Para 2 example	General	there is a «fail()» function concept that seems unused for the example, and is not mentioned in the accompanying text; was that voluntary?		ACCEPTED
US 32		14.10.1.1	p2	ed	P and Q are defined with two different meanings in the same numbered paragraph. Substitute different	A conjunction PA is equivalent to another conjunction QB if and only if the left operands of	ACCEPTED

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
					letters for one of the uses	PA and QB are equivalent and the right operands of PA and QB are equivalent.	
US 33		14.10.1.1	P3	ed	P and Q are defined with two different meanings in the same numbered paragraph. Substitute different letters for one of the uses	A disjunction PA is equivalent to another disjunction QB if and only if the left operands of PA and QB are equivalent and the right operands of PA and QB are equivalent.	ACCEPTED
CA2	N/A	14.10.1.2	Para 1 example	Technical	we have template <typename T> concept bool C = sizeof(T) == 4 && !true; The associated comment states: «requires predicate constraints sizeof(T)==4 and !t». The «!t» is what is confusing here, in my opinion, as there is no occurrence of «t» in the concept C. Was «t» supposed to be «true» or is there a variable missing?		ACCEPTED
US 34		14.10.1.2	p2	ed	Example has a typo of '!t' instead of '!true' in the first line of commented code.	// sizeof(T) == 4 and !true	ACCEPTED
US 35		14.10.1.3	p1	ed	Duplicate 'the the' at the end of the example.	The type argument int satisfies this constraint because the the expression ++t is valid after substituting int for T.	ACCEPTED

CA3	N/A	14.10.1.5	Para 1 example	ge	more a suggestion than a comment: would a convertible-to-type example like the following be appropriate?
					
					template <typename T> concept bool D =
					  requires (T a) {
					    { a } -> int; //
							REJECTED
							The intent of the wording was that determining whether the constraint is satisfied was done in the context in which it appears, not in a context-independent way as

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:2015-07-25	Document: SC22/WG21 N4550	Project: ISO/IEC PDTS 19217

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					std::is_convertible<T,int>::value ? }; It could be here or in 14.10.1.6, but I get the feeling it would follow the a==b example nicely. There is something similar in the middle of the page, with concept C2, but it is more involved and contributes something else to reader comprehension.		suggested. This approach causes problems with partial ordering, however, so an issue against the TS will be opened for further consideration at a later date.

US 36		[temp.constr .conv] (14.10.1.5)	1	te	This paragraph introduces implicit conversion constraints to specify (via the trailing-return-type notation ->) that a constraint is satisfied iff an expression E is convertible to a type T. It would be very useful to have similar constraints that are satisfied iff decltype(E) is exactly the type T.	Introduce new notation (e.g., E => T) to denote a constraint that is satisfied iff the expression E has precisely the type T. Here is a practical example of the utility of such a feature:
						template <typename T>
						concept bool CopyAssignable =
						requires (T a, T b) {
						{ a = b } => T const&;
						};
							REJECTED
							There was no consensus for a change at this time, but an issue will be opened for future consideration by WG21's Evolution Working Group.

US 37		[temp.constr .decl] (14.10.2)	2 just before bullet (2.1)	ed	There seems to be an extraneous word in "The ordering of operands in the that conjunction is:".	Strike one of the words in "the that".	ACCEPTED
US 38		14.10.1.5	p2	ed	Duplicate 'the the'.	… using the rules in 14.6.6.1 to compare expressions, and the the types of P and Q are equivalent …	ACCEPTED
CA4	n/a	14.10.1.6	Para 2 example	te	I don't think g((int*)nullptr); is an error, as g() is an unconstrained template in this example (unless I missed something). Did the author mean to declare g() as follows?		ACCEPTED

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:2015-07-25	Document: SC22/WG21 N4550	Project: ISO/IEC PDTS 19217

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
					template <typename T> requires C2<T> void g(T); or as follows? template <C2 T> void g(T); or as follows? C2{T} void g(T);		
GB 4		14.10.1.6 [temp.constr.deduct]	P2	Ed	There is no constraint in the example: template<typename T> void g(T);	Add a constraint	ACCEPTED
GB 5		14.10.2 [temp.constr.decl]	P3.4	Ed	The text refers to " Note that the normalized constraints of #2 includes two atomic constraints: sizeof(char) == 1 and 1 == 2." This is incorrect.	Change the example or the note to match each other.	ACCEPTED WITH MODIFICATION The referenced text was deleted.
CA 10	n/a	14.10.3	Para 5	te	The definition of “at least as constrained” is: “A declaration D1 is at least as constrained as another declaration D2 when D1 is more constrained than D2, and D2 is not more constrained than D1.” Doesn’t this definition make two declarations with equivalent constraints not be “at least as constrained” as each other? For example: void foo(C c); // D1	Replace paragraph 5 with the following paragraphs: Two declarations D1 and D2 are *equally constrained* if - D1 and D2 are both unconstrained; or - D1 and D2 are both constrained, D1’s associated constraints subsume those of D2, and D2’s associated constraints subsume those of D1	ACCEPTED WITH MODIFICATION

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:2015-07-25	Document: SC22/WG21 N4550	Project: ISO/IEC PDTS 19217

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

					void foo(C c); // D2
					Here, D1 is not "more constrained than" D2, so according to this definition, it's not "at least as constrained as" D2, either – but don't we want it to be?
						A declaration D1 is *at least constrained* as another declaration D2 if
						- D1 is more constrained than D2; or
						- D1 and D2 are equally constrained	

CA 11	n/a	14.10.4	Para 3, bullet 3.3	te	Are there any situations where bullet 3.3 applies? It seems that bullets 3.1 and 3.2 already exhaust the cases listed in paragraph 1 above.		ACCEPTED WITH MODIFICATION The text was changed to make explicit that template-ids must be fully resolved in constraint-expressions when they name a concept.
CA5	N/A	14.10.4	Para 4.3	te	at the bottom of the example, functions q1() and q2() have to satisfy concept Q (with a variadic number of parameters in one case and with a single parameter in the other). I do not see concept Q in that example. An oversight?		ACCEPTED
US 39		[temp.constr .resolve] (14.10.4)	bullet (3.2)	ed	The phrase "a sequence wildcards" seems to be missing a word.	Insert "of" before "wildcards".	ACCEPTED

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