Document No: SC22/WG21 N2837
INCITS / PL22.16 09-0027
Date: 2009-01-30
Project: Programming Language C++
Reference: WG21 N2800: C++0X CD1
Reply to: Barry Hedquist
PL22.16 IR
beh@peren.com

C++0X, CD 1, National Body Comments

Attached is a complete set of National Body Comments submitted to JTC1 SC22 in response to the SC22 Letter Ballot for Committee Draft 1 of the revision of ISO/IEC 14882, aka C++0X.

Comments that were submitted without numbering were numbered manually in the exact order of the NB's official ballot response. The comments were then organized per the hierarchy of the balloted document, SC22 N4411 (WG21 N2800). No editing of any kind was done on any of the comments.

C++0X, CD 1, NB Comments	Date: 21 Feb 2009	Document: **SC22 N4411, ISO/IEC CD 14882**

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
FR 1	General Comment		ge	Interactions between several new features appear obscure, and very few examples are offered to guide understanding of the formal text on interaction between these new additions. We worry about the complexity of the programming model so created.		
US 1	1-16		ge/te	The active issues identified in WG21 N2803, C++ Standard Core Language Active Issues, must be addressed and appropriate action taken. http://www.open-std.org/jtc1/sc22/wg21/docs/cwg_active.html	Appropriate action would include making changes to the CD, identifying an issue as not requiring a change to the CD, or deferring the issue to a later point in time.	
CA-1			Ge	There are quite a number of defects for the current CD recorded in SC22/WG21-N2803 and N2806	Consider these comments and update ISO/IEC CD 14882 accordingly	
DE-1	1 through 16		ge/te	DE-1 Consider addressing a significant part of the unresolved core language issues presented in WG21 document N2791 "C++ Standard Core Language Active Issues, Revision 59", available at http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2791.html .		
CH 2	all		te	The issues on the issues lists shall be addressed before the standard becomes final.		
US 3	all		ed	Latin abbreviations are presented incorrectly.	Italicize all Latin abbreviations, append commas after each occurrence of *i.e.* and *e.g.*, and remove extraneous space after each such abbreviation.	
FR 3	1 [intro.scope]	2	ed	C++ is split at the end of line.		
US 4	1.1	2	ed	There is a bad line break in "C++".		

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 1	1.1	2	Ed	List of additional facilities over C has been extended with this standard, so should be mentioned in the introductory material.	Add the following to the list in 1.1p2: atomic operations concurrency alignment control user-defined literals attributes	
FR 4	1.2 [intro.refs]	1	ed	Is the lack of reference to ISO/CEI 9899/AC3:2007 voluntary?		
UK 2	1.2	1	Ed	We recommend taking the latest update to each listed standard, yet the C standard is quite deliberately held back to the 1990 version without comment.+	... not sure ...	
UK 3	1.3.1		Ed	The definition of an argument does not seem to cover many assumed use cases, and we believe that is not intentional.	Revise the definition of argument to answer question such as: Are lambda-captures arguments? Are type names in a throw-spec arguments? 'Argument' to casts, typeid, alignof, alignas, decltype and sizeof? why in x[arg] : arg is not an agrument, but the value forwarded to operator[]() is ? Does not apply to operators as call-points not bounded by parenthises ? Similar for copy initialization and conversion? what are Deduced template 'arguments'? what are 'default arguments'? can attributes have arguments? what about concepts, requires clauses and concept_map instantiations? What about user-defined literals where parens are not used?	
UK 4	1.3.3		Te	This definition is essentially worthless, as it says nothing about what distinguished a diagnostic message from other output messages provided by the implementation	... add something about the diagnostic message being a message issues by the implementation when translating a program that violates the rules of the standard. ...	
FR 5	1.3.4 [defns.dynamic.type]		te	"The dynamic type of an rvalue expression is its static type." Is this true with rvalue references?		
US 5	1.3.5		te	The wording is unclear as to whether it is the input or the implementation "that is not a well-formed program".	Reword to clarify that it is the input that is here considered not well-formed.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
FR 6	1.3.6 [defns.impl.defined]		ed	There is a page break between the title and the paragraph.		
FR 7	1.3.13 [defns.undefined]		ed	[intro.execution]/5 explicitly allows non causal undefined behaviour,	Adding it to the note outlying possible undefined behaviours.	
US 6	1.3.14		ge	Unspecified behavior does not clearly state whether or not undefined behavior is permitted. (The standard says that "usually, the range of possible behaviors is delineated", but what happens if the range is not delineated? Is a crash, or worse, allowed?)	Clearly state whether or not Unspecified behavior includes undefined behavior.	
FR 8	1.4 [intro.compliance]	8	ed	The paragraph as its stands seems to require that violations of the ODR (which make a program ill-formed) are required to be diagnosed if the program also uses an extension which defines some cases of ODR.		
UK 5	1.5		Ge	Missing checklist of implementation defined behaviour (see ISO/IEC TR 10176, 4.1.1p6)	Provide a new annex with the missing checklist	
UK 6	1.5		Ge	Missing annex describing potential incompatibility to previous edition of the standard (see ISO/IEC TR 10176, 4.1.1p9)	Provide a new annex with the missing documentation. See n2733(08-0243) for a starting point	
US 7	1.5	2	ed	There is no mention of Clause 17.	Include Clause 17 among the list of Clauses that specify the Standard Library.	
US 8	1.5	2	te	The paragraph omits to mention concepts and concept maps among its list of entities defined in the Standard Library.	Mention concepts and concept maps among the list of entities.	
US 9	1.6	1	ed	The syntax description does not account for lines that wrap.		
US 10	1.7	3	ed	The term thread is used before defined.	Reference 1.10 [intro.multithread].	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
US 11	1.7	¶ 3 last sent.	ed	The phrase “threads of execution” should be accompanied by a reference to [intro.multithread].	Insert the recommended reference.	
US 12	1.7	¶ 3 first sent.	te	A memory location is not an object as the sentence claims.	Clarify that a memory location “holds” an object rather than that it “is” an object.	
US 13	1.7	¶ 3 last sent.	te	It is unclear what is meant by memory locations that are "separate": are they distinct? non-overlapping? how much "separation" is needed?	Provide either a better definition of “separate” or reword (this and subsequent paragraphs) to avoid this term.	
US 14	1.7	¶ 4	te	The phrase "no matter what the sizes of the intervening bit-fields happen to be" contradicts the claim of separation "by a zero-length bit-field declaration".	Delete the “no matter…” phrase, or resolve the contradiction in a different way.	
US 15	1.7	¶ 5	te	A struct does not “contain” memory locations.	Reword so that a struct is “held in” one or more memory locations.	
US 16	1.9			The discussion of observable behavior in 1.9 is not consistent with the addition of threads to the language. Volatile reads and writes and other observable actions no longer occur in a single "sequence”.	Remove/replace various occurrences of "sequence" in 1.9.	
UK 8	1.9	5	Te	With parallel execution there is no longer the idea of a single execution sequence for a program. Instead, a program may be considered a set of exectution sequences.	Update first sentance as: A conforming implementation executing a well-formed program shall produce the same observable behavior as one of the possible SETS OF execution sequences of the corresponding instance of the abstract machine CONFORMING TO THE MEMORY MODEL (1.10) with the same program and the same input.	
UK 7	1.9	6	Te	Does the term 'sequence' imply all reads/writes through volatile memory much be serialized, and cannot occur in parallel on truly parallel hardware? Allow for multiple concurrent sequences where each sequence is constrained by this observable behaviour rule, and multiple sequences are constrained by the memory model and happens-before relationships defined in 1.10	Replace 'sequence' with 'sequences'.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
FR 9	1.9 [intro.execution]	16	ed	This example use int *v while the other examples seems to use notation like int* v.		
US 17	1.10	1	Ge	This definition of “thread” is poor, and assumes the user already knows what multi-threaded means (probably true!). In particular, it does not deal adequately with the concept that all threads share the same address space.	Replace first sentence of para 1 as follows: Under a hosted implementation, a C++ program can have more than one thread of execution (a.k.a. thread) running concurrently. Each thread is a single flow of control within a program. Anything whose address may be determined by a thread, including but not limited to static objects, storage obtained via new or by any dynamic allocator, directly addressable storage obtained through implementation-defined functions, and automatic variables, are accessible to all threads in the same program.	
UK 9	2.1	2, 4	Te	Undefined behaviour is a drastic way to silently ignore minor issues. The cases in this paragraph could be easily defined. In this case opt for conditionally supported behaviour, which mandates a diagnostic if the compiler is not prepared to handle the syntax consistently.	Replace undefined behaviour with conditionally supported behavior. Conditional behaviour may be implementation defined, although suggest there is a reasonable default in each case. For creating a universal-character name, splice text to create a universal-character. In the case of a file ending without a newline, treat as if the newline was implictly added, with an empty line to follow if the last character was a back-slash.	
UK 10	2.1	3	Te	Implementation defined seems unnecessarily burdensome for negligible gain. I am yet to see code that depended on whether non-empty sequences of whitespace were concatenated. Better left unspecified.	How the compiler treats non-empty sequences of whitespace should be left unspecified, rather than implementation-defined.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
FR 10	2.1 [lex.phases]/5 and 2.2 [lex.charset]/3		te	[defns.multibyte] "the extended character set." [lex.charset]/3 cited below implies that there is an extended character set per locale. [lex.phases]/5 "Each [...] universal-character-name [...] is converted to the corresponding member of the execution character set" [lex.charset]/3 "The values of the members of the execution character sets are implementation defined, and any additional members are locale-specific." Together they seem to imply that what is locale-specific is if a value is valid or not for the current locale, not the representation of a given universal character. This is not the behaviour of at least some compilers I've access to which are allowing different codes for the same abstract character in different locale. During phase 5, they are using an implementation defined char set.		
UK 11	2.3		Te	Trigraphs are a complicated solution to an old problem, that cause more problems than they solve in the modern environment. Unexpected trigraphs in string literals and occasionally in comments can be very confusing for the non-expert.	Deprecate the whole of 2.3 and move it to appendix D.	
UK 12	2.4, 2.8	2	Te	This undefined behaviour in token concatenation is worrying and we believe hard to justify. An implementation should either support this in a defined way, or issue a diagnosic. Documenting existing practice should not break existing implementations, although unconditionally requiring a diagnostic would lead to more portable programs.	Replace undefined behaviour with conditionally supported behaviour with implementation defined semantics.	
US 18	2.4	¶ 2	ed	The paragraph begins with an empty line.	Delete the empty line.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
FR 11	2.4 [lex.pptokens]	3	ed	There are spurious empty lines.		
FR 12	2.5 [lex.digraph] and 2.11 [lex.key]/2		te	The alternative representations are reserved as such even in attribute. Is that what is wanted?		
FI 2	2.5	Table 2	te	Add eq, for spelling out == in order to distinguish it from the assignment operator.	See eq-keyword.doc, eq-keyword.ppt	
UK 13	2.9	2	Ed	This text is confusing in isolation, as it implies pp-numbers do not have a value in translation phase 4 when evaluating #if preprocessor expressions.	Add a note with a cross-refernce to 16.1 that a pp-number may briefly acquire a value during translation phase 4 while evaluating #if expressions.	
UK 14	2.11	table 3	Ed	The table is nearly sorted, but not quite. It was sorted in previous versions of the standard.	Sort the table.	
JP 1	2.11	Table 3	ed	Keywords in the table are listed disorderly. Also, a part of a frame of the table is not drawn.	Sort it in alphabetical order. Complete the table frame.	
US 19	2.13.1	Table 5, rows “l or L” and “ll or LL”	te	The final entry in the last column (“unsigned long int”) is incorrect.	Replace the incorrect entries by “unsigned long long int”.	
US 20	2.13.1, 2.13.3		te	Long strings of digits in literals are a continuing problem in the production and maintenance of programs.	Adopt the 1983 technology of Ada and use underscores to separate digits. http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2007/n2281.html	
UK 15	2.13.2	2	Te	Inconsistency between definition of a multicharacter literal and a wide character literal containing multiple c-chars.	Define the term multicharacter wide literal for a wchar_t literal containing multiple elements, and specify its type is integer (or wider)	
UK 16	2.13.2	3	Ed	Not immediately clear why the question mark needs escaping. A note would help.	Add a note explaining that the ? character may need escaping to avoid accidentally creating a trigraph.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
JP 2	2.13.4	1st paragraph, 2nd line	ed	Typo, R"..." should be R"[...]"	Correct typo.	
JP 3	2.13.4	2nd paragraph	te	We think that the explanation of d-char-sequence is not enough.	Add the following. (1) Add the following to the explanation of d-char-sequence, more easily to understand. ...prefix is a raw string literal. The d-char-sequence is used as delimiter. The terminating d-char-sequence of ... (2) Add the following note that there are square brackets in r-char-sequence. [Note: char foo[] = R"*delimiter*[[a-z] specifies a range which matches any lowercase letter from "a" to "z".]*delimiter*"; the expression statement behaves exactly the same as char foo[]="[a-z] specifies a range which matches any lowercase letter from \"a\" to \"z\"."; - end note]	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
JP 4	2.13.4	3rd paragraph, 1st line of example	ed	Typo. Lack of a necessary backslash in the first line of the example as follows: const char *p = R"[a b c]"; should be const char *p = R"[a\ b c]";	Correct typo.	
US 21	2.13.4	¶ 3	ed	The paragraph, marked as a Note, contains an embedded example not marked as such.	Denote the code (and perhaps also its commentary) as an Example.	
US 22	2.13.4	¶ 3	te	The code does not have the effect predicted by its accompanying narrative.	Append a backslash to the first line of the code.	
JP 5	2.13.4	11th paragraph, Table 7	te	It is not explicit how to combine raw-string and non-raw-string.	Add rules containing raw-string in the table 7.	
FR 13	2.13.4 [lex.string]	3	ed	Shouldn't the assert be assert(std::strcmp(p, "a\nb\nc") == 0);		
UK 17	2.13.4	10	Te	It would be preferred for attempts to modify string literals to be diagnosable errors. This is not possible due to the deprecated implicit conversion to pointer to null-terminated character sequence of non-const characters. If this deprecated conversion were remove (see other comments) then string literals are always accessed through const types, and the compiler can enforce the no modification rule. The only exception would be using const_cast to cast away constness, but this is already covered under the const_cast rules so needs no further detail here.	(asssuming deprecated conversion to non-const array is removed or can be turned off) Strike the sentence on undefined behaviour.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 18	2.13.4		Te	The addition of static_assert (7p4) to the language raises the need to concatenate string representations of integral constant expressions (typically from a sizeof or alignof expression) with narrow string literals to provide an informative error message. There is no need to support arbitrary constant expressions and especially not floating point values or formatting flags. Likewise, the need is purely to support static_assert so only narrow string literal support is required, although generalizing to other literal types would be useful.	Define a syntax to support string-ization of integral constant expressions in a form eligible for string literal concatenation, 2.13.4p6. Suggested syntax: I" integral-constant-expression ". There is no raw variant, although it could combine with type specifier in the same way that the R prefix does, supporting u8I, uI, UI and LI.	
UK 19	2.13.4		Ed	The grammar for string literal is becoming unwieldy and could easily be refactored into the type optional specifier and the string contents.	Refactor string-literal grammar as: (note - current Drupal view loses formatting which is vital to clearly read the grammar) string-literal: string-literal-type-specifierOPT string-literal-body string-literal-type-specifier: one of u8 u U L string-literal-body: " s-char-sequenceOPT " R raw-string	
FR 14	3 [basic]	7	ed	"In general it is necessary to determine whether a name denotes one of these entities before parsing the program that contains it."	Would prefer "... before continuing to parse the program that contains it." or even "... to complete the parsing of the program that contains it." as some names denotes entities declared after the first occurrence.	
FR 15	3 [basic]	8	ed	/operator-function-id/, /conversion-function-id/, /template-id/ are followed by a space and then a "s" while usually such production names aren't followed by a space when put in plural (see /identifier/).		
UK 20	3		Ge	Chapter 3 ("Basic concepts") provides common definitions used in the rest of the document. Now that we have concepts as a primary feature, the title of this chapter can be confusing as it does not refer to the language feature but to definitions used in the document.	Change the title to "Basic definitions".	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 21	3	2	Ed	Concepts is now the name of a specific feature of the language, the term now risks confusion and ambiguity when used in the more general sense.	Rename the chapter Basic ???. THe note in p2 specifically needs similar rewording	
UK 22	3	6	Te	References are frequently considered variables, but this definition only applies to objects.	Add "or reference" after both uses of "object"	
UK 23	3.1	2	Ed	alias-declarations are not definitions and should be added to the list	Add alias-declaration after typedef declaration.	
UK 24	3.1	2	Te	The current words suggest the declaration of a static integral constant data member of a class cannot be a definition. Trying to fix this wording in-place will be verbose and risk raising more confusion than it solves, so suggest a footnote to call out the special case	Add a footnote attached to the static data membmer rule: *static data member delcarations of intergral type may also be definitions if a constant integral expression is provided for an initializer.	
UK 25	3.1	3	Ed	Example is misleading as implicitly defined default constructor uses default initialization, not value initialization, for non-static data members. In the case of std::String this makes no difference, but it makes a big difference for fundamental types and pointers.	Remove the : s() from the illustrated default constructor: struct C { std::string s; C() { } C(const C& x): s(x.s) { } C& operator=(const C& x) { s = x.s; return *this; } ~C() { } };	
UK 26	3.2	1	Te	THe one definition rule should cover references, and unless the term 'variable' is extended to cover references the list in this paragraph is incomplete.	Either include references in the definition of 'variable' (see earlier comment) or add reference to the list in this paragraph.	
UK 27	3.2	4	Ed	A class type must be complete when catching exceptions, even by reference or pointer. See 15.3.	Add "when used in an exception-handler (15.3)" to the list.	
FR 16	3.3 [Declarative regions and scopes.]		te	The scope of function parameters is defined, but what is the scope of template parameters?		
UK 28	3.3.1	3	Te	Class templates are not classes, so we should include this case.	ammend "class" to "class or class template"	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 29	3.3.10	3	Te	operators and conversion functions do not have names, yet are susceptible to 'name hiding' within a class - indeed we rely on this for the implicitly declared copy-assignment operator.	Add the additional phrase "The declaration of an operator or conversion function in a derived class (Clause 10) hides the declaration of an operator or conversion function of a base class of the same operator or type;"	
FR 17	3.5 [Program and linkage]		te	This section does not specify whether concept names have linkage. Do they or not? If concept names do not have linkage, then a note is appropriate, and that would be a bit surprising and curious. What is the rationale?		
UK 30	3.5	2	Te	This paragraph implies concepts have no linkage (do they need it?) and that the entities behind names without linkage cannot be used in other scopes. This maybe a bigger problem for concept maps?	Add a note to clarify that concepts don't need linkage.	
UK 31	3.5	4	Te	What is the linkage of names declared inside a namespace, in turn declared inside an anonymous namespace? It is not clear why such a namespace has no linkage, and there is no language suggesting its memmbers should lose linkage with it, which we assume is the intended consequence.	Clarify rules for namespaces inside nested namespaces, or remove the restriction.	
US 23	3.5	6	ed	Bad paragraph break.		
FR 18	3.5 [basic.link]	6	ed	The paragraph number is not aligned with the text.		
FR 19	3.6 [Start and termination]		te	This section completely ignores the real world and practical case of dynamically linked or loaded libraries. In current computing environments, they are ubiquitous and they cannot be ignored in practical C++ programs. The Standard should address this aspect.		

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 32	3.6.1	3	Te	Do we really want to allow: constexpr int main() { return 0; } as a valid program?	Add constexpr to the list of ill-formed things to annotate main	
US 24	3.6.1	4	te	std::quick_exit is not referenced.	Reference std::quick_exit as well as std::exit in saying that automatic objects are not destroyed. It should **not** do so in saying that calling std::quick_exit is undefined from within destructors for static or thread duration objects.	
US 25	3.6.3	¶ 2 last sent.	ed	The parenthesized phrase, introduced via "i.e." is in the nature of an example.	Change "i.e." to "e.g."	
JP 6	3.7.4.1	4th paragraph, 4th line	ed	Typo. Lack of a comma right after "(3.7.2)" in the sentence while there are commas after any other recitations like "(3.7.1)". It is just a unification matter. [Note: in particular, a global allocation function is not called to allocate storage for objects with static storage duration (3.7.1), for objects or references with thread storage duration (3.7.2) for objects of type std::type_info (5.2.8), or for the copy of an object thrown by a throw expression (15.1). -end note] should be [Note: in particular, a global allocation function is not called to allocate storage for objects with static storage duration (3.7.1), for objects or references with thread storage duration (3.7.2), for objects of type std::type_info (5.2.8), or for the copy of an object thrown by a throw expression (15.1). -end note]	Correct typo.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
DE-3	3.7.4.3		te	DE-3 It is unclear whether the following code has well-defined behavior; none of the bullets in the second paragraph seem to apply. int& i = *new int(5); delete &i;	Clarify that &i is considered a safely-derived pointer value.	
US 26	3.8	1 and 5	te	Use of object fields during destruction is excessively and erroneously constrained.	See the attached document "Issues with the C++ Standard" under Chapter 3 "Use of objects, especially from other threads, during destruction".	
US 27	3.9	¶ 9 first sent.	ed	There is a superfluous/extraneous “and”.	Delete “and” from the phrase “and std::nullptr_t”.	
FR 20	3.9 [Types]		te	The phrase 'effective type' is defined and used in a way that is incompatible with C99. Such a deliberate incompatible choice of terminology is both unfortunate and confusing, given past practice of the committee to maintain greater compatibility with C99. We strongly suggest that the phrase 'effective type' not be used in such an incompatible way.		
JP 7	3.9.2	3rd paragraph, 13th line	ed	over-aligned type was added as new notion. So it is preferable to add the link after that.	Add (3.11) after over-aligned type as the link. [Note: pointers to over-aligned types(3.11) have no special representation, but their range of valid values is restricted by the extended alignment requirement. This International Standard specifies only two ways of obtaining such a pointer: taking the address of a valid object with an over-aligned type(3.11), and using one of the runtime pointer alignment functions. An implementation may provide other means of obtaining a valid pointer value for an over-aligned type(3.11).—end note]	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
US 28	3.9.3	¶ 5 first sent.	ed	The closing braces of the first two sets are preceded by extraneous space.	Delete the extra spaces.	
DE 4	4.2	p2	te	DE-4 The deprecated conversion from string literals to pointer to non-const character types should be limited to those conversions and types of string literals that were already present in ISO/IEC 14882:2003, or the deprecated conversions should be removed entirely.	Consider applying the proposed resolution presented in core issue 693 in WG21 document N2714 “C++ Standard Core Language Active Issues, Revision 58“, available at http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2714.html ; or remove only the conversions to "pointer to char16_t", "pointer to char32_t" in 4.2 paragraph 2 and 15.1 paragraph 3.	
CH 1	4.9 and 5.2.9		te	With respect to the target type, pointer to members should behave like normal pointers (least surprise principle).	The standard should allow implicit conversions from ``pointer to member of T of type *cv* D'' to ``pointer to member of T of type *cv* B'', where D is of class type and B is a public base of D, It should allow explicit conversion the other way around.	
DE-5	4.11, 5.3.1, 5.5		te	DE-5 Ref-qualification has not been integrated with pointer-to-members.	Review implicit conversions (4.11), forming pointer-to-members (5.3.1), and dereferencing pointer-to-members (5.5) for type-safety concerns in the presence of ref-qualifiers on the member.	
UK 33	4.13	1	Te	We have: "No two signed integer types shall have the same rank ..." "the rank of char shall equal the rank of signed char" Can we therefore deduce that char may not be signed?	Replace the first sentence with "No two signed integer types shall have the same rank, even if they have the same representation, except that signed char shall have the same rank as char even if char is signed (3.9.1/1)."	
UK 34	4.13	1	Ed	6th bullet, "the rank of char" - first letter should be capitalised for consistency with the other bullets	The rank of char	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 36	5.1	1	Ed	Primary expressions are literals, names, names qualified by the scope resolution operator ::, and lambda expressions. The immediately following grammar flatly contradicts this - this and (e) are also lambda expressions.	Delete this paragraph.	
UK 37	5.1	11	Ed	Member function templates are not member functions, so should also be listed in the 3rd bullet	Add member function templates to the 3rd bullet	
UK 38	5.1	3	Te	this might be useful in a few more places than it is permitted, specifically in decltype expressions within a class. Two examples that would be ill-formed at class scope without changes: typedef decltype(*this) this_type; decltype([this]{ return this->memfun(); }) my_lambda;	... words to follow ...	
JP 8	5.1	7th paragraph, Syntax rules	te	In the current syntax definition, a scope operator(::) cannot be applied to decltype, but it should be. It would be useful in the case to obtain member type(nested-type) from an instance as follows: vector<int> v; decltype(v)::value_type i = 0; // int i = 0;	Add “decltype (expression) :: “ to nested-name-specifier syntax like below. nested-name-specifier: type-name :: namespace-name :: nested-name-specifier identifier :: nested-name-specifier templateopt simple-template-id :: nested-name-specifieropt concept-id :: decltype (expression) ::	

JP 9	5.1.1		te	It would be preferable that “&&” could be specified in a lambda expression to declare move capture.
				
				Here is an example from N2709.
				
				template<typename F>
				std::unique_future<typename std::result_of<F()>::type>
				spawn_task(F f){
				typedef typename std::result_of<F()>::type result_type;
					Add move capture in a lambda expression.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

				struct local_task {
				std::promise<result_type> promise;
				F func;
				local_task(local_task const& other)=delete;
				local_task(F func_):
				func(func_)
				{}
				
				local_task(local_task&& other):
				promise(std::move(other.promise)),
				f(std::move(other.f))
				{}
				
				void operator() {
				try
				{
				promise.set_value(f());
				}
				catch(...)
				{
				promise.set_exception(std::current_exception());
				}
				}
				};
				
				local_task task(std::move(f));
				
				std::unique_future<result_type>
				res(task.promise.get_future());
				std::thread(std::move(task));
				return res;
				}
				
				This can be rewritten simply as follows if move capture can be used in a lambda expression.		

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

				template<typename F>
				std::unique_future<typename std::result_of<F()>::type>
				spawn_task(F f){
				typedef typename std::result_of<F()>::type result_type;
				
				std::promise<result_type> promise;
				std::unique_future<result_type>
				res(promise.get_future());
				std::thread([&&promise, &&f]() {
				try
				{
				promise.set_value(f());
				}
				catch(...)
				{
				promise.set_exception(std::current_exception());
				}
				});
				return res;
				}		

JP 10	5.1.1		te	In the current syntax definition, a returned type of a function object cannot be obtained by using result_of from an unnamed function object generated by a lambda expression because it doesn't have result type.
				
				template <class F>
				void foo(F f)
				{
				typedef std::result_of<F()>::type result; // error
				}
				foo([]{});
				
				If "Callable" or "Predicate" concept is specified, a returned type can be obtained from a function object without result_type. But it is preferable to be able to obtain it with template.
					Add result_type to the syntax of an unnamed function object generated by a lambda expression.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
US 29	5.1.1		te	The standard does not state whether or not direct recursion of lambdas is possible.		
US 30	5.1.1		te	The standard does not clarify the meaning of this in lambdas. Does it mean this lambda, or this class within which the lambda is nested?		
US 31	5.1.1		te	The current wording does not specify how context capturing and name resolution take place when the inner lambda refers to the outer lambda's locals variables and parameters.		
UK 45	5.1.1	para 2	Te	Lambda is a language feature with an apparent dependency on <functional>. This increases dependency of language on library, and is inconsistent with the definition of freestanding in 17.6.2.4.	Change the text "a closure object behaves as a function object" to "a closure object is a built-in object which behaves as a function object"; and after "context.", insert " A closure object may be used without any need for <functional>." This makes clear what may already be implied, namely that lambdas can be used in freestanding implementations and don't increase dependency of language on library. (Marked as technical comment anyway because this clarity is technically important).	
US 32	5.1.1	3	ed	The final italic "this" in the paragraph should be a teletype "this".		
UK 39	5.1.1	11	Te	This paragraph lists all the special member functions for the class representing a lambda. But it omits the destructor, which is awkward.	Add "F has an implicitly-declared destructor".	
UK 40	5.1.1	12	Te	If one or more names in the effective capture set are preceded by &, the effect of invoking a closure object or a copy after the innermost block scope of the context of the lambda expression has been exited is undefined. That is too restrictive. The behaviour should be undefined iff the lifetime of any of the variables referenced has ended.	If one or more names in the effective capture set are preceded by &, the effect of invoking a closure object or a copy after the lifetime of any of the variables referenced has ended is undefined.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				This should be safe and legal; currently it has undefined behaviour: int i; reference_closure<void ()> f; if (blah) { f = [&i]() { }; } if (f) f();		
UK 41	5.1.1	12	Te	For argument dependant lookup (3.4.2) the associated namespaces for a class include its bases, and associated namespaces of its bases. Requiring the result of a lambda expression *to dervide from* std::reference_closure means that ADL will look in namespace std when the lambda capture is entirely by reference, which might have surprising results. Also, relying on the idea of implicitly slicing objects is uncomfortable.	Replace inheritance with implicit conversion.	
UK 42	5.1.1		Te	A lambda with an empty capture list has identical semantics to a regular function type. By requiring this mapping we get an efficient lambda type with a known API that is also compatible with existing operating system and C library functions.	Add a new paragraph: "A lambda expression with an empty capture set shall be convertible to pointer to function type R(P), where R is the return type and P is the parameter-type-list of the lambda expression." Additionally it might be good to (a) allow conversion to function reference (b) allow extern "C" function pointer types	
UK 43	5.1.1	12	Te	The note spells out the intent that objects from lambda-expressions with an effective capture list of references should be implemented as a pair of pointers. However, nothing in the rest of 5.1.1 lifts the requirement of to declare a reference member for each captured name, and a non-normative note is not enough to relax that.	... provvide exceptions in the right places ...	
UK 44	5.1.1	12	Te	There is a strong similarity between a [&]{} lambda capturing a stack frame, and a [this]{} lambda binding a member function to a class instance. The reference_closure requirement should be extended to the second case, although we need some syntax to create such an object that is distinct from the existing pointer-to-member syntax. This would be a cleaner alternative to the new std::mem_fn library component.	Extend reference_closure requirement to cover [this] lambdas. Consider a simple syntax for creating such bound expressions.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 46	5.1.1	para 12	Te	The requirement that a lambda meeting appropriate conditions be an object derived from reference_closure makes lambda the language feature dependent on <functional>, which increases dependency of the language on the library and bloats the definition of freestanding C++.	Replace text "is publicly derived from" with "shall be implemented in a manner indistinguishable from". This places an ABI constraint on reference closures such that compiler and library implementer have to do compatible things. But it cuts the dependency of lambda syntax on <functional>.	
DE-6	5.1.1, 20.7.18		te	DE-6 Some uses of lambda expressions refer to specializations of the unconstrained class template std::reference_closure (5.1.1). If the lambda expression appears in a constrained context and the return type or a parameter type for the lambda depend on a template parameter (see 14.10), such a use is ill-formed.	In 20.7.18, for the class template std::reference_closure, require Returnable for R and VariableType for each of the ArgTypes.	
DE-7	5.1.1	p10	ed	DE-7 The note at the end of paragraph 10 appears to be garbled.	Remove "or references" in the note.	
DE-8	5.1.1	p10	te	DE-8 The construction of the function call operator signature is missing specifications for the ref-qualifier and the attribute-specifier.	Add bullets that say that the ref-qualifier and the attribute-specifier are absent.	
US 33	5.1.1	11	Ge	There is no definition of “move constructor” or “move operation”	Since this is the first place the terms are used, a definition should either be added here, or a cross reference to one.	
DE-9	5.1.1		te	DE-9 There is not a single example of a lambda-expression in the standard. See also core issue 720 in WG21 document N2791 "C++ Standard Core Language Active Issues, Revision 59", available at http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2791.html .	Add a few well-chosen examples.	
UK 52	5.2	3	Ed	This paragraph seens out of place, assignment expressions are covered in 5.17	Move p3 to subsection 5.17	
UK 53	5.2.1		Te	The definition in p1 makes no allowance for overloaded operator[] that treats the expression as a simple function call, and does not support the interchangability of arguments. Howver p2 relies on this definition when	Insert a new p2 describing the changed semantics for overloaded operator[]. This should be a note to avoid introducing normative text that could potentially conflict with the later definiton of these	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				describing the use of brace-init-lists inside [].	semantics.	
UK 59	5.2.2	7	Te	When there is no parameter for a given argument, the argument is passed in such a way that the receiving function can obtain the value of the argument by invoking va_arg. That shouldn't apply to parameter packs. template <class ... Types> void f(Types ... pack); f(1, 2, 3);	Clarify that this sentence only applies where the ellipsis is used.	
UK 60	5.2.5	3	Ed	In the remainder of 5.2.5, cq represents either const or the absence of const vq represents either volatile or the absence of volatile.	Add "and" before vq	
UK 61	5.2.5	p1	Ed	Together with footnote 60 there may be confusion that the postfix expression is always evaluated - even when part of an unevaluated operand. We believe the standard does not require this, and a comment in the existing note would be a useful clarification.	Clarify in footnote 60 that this will not happen if the whole expression is an unevaluated operand.	
UK 62	5.2.5	4	Te	In the final bullet, what does 'not an lvalue' mean? Does it imply rvalue, or are there other possible meanings? Should clauses that trigger on rvalues pick up on this?	Replace 'not an lvalue' with 'is an rvalue'.	
DE-10	5.2.5		te	DE-10 If E1.E2 is referring to a non-static member function, the potential ref-qualification on E2 should be taken into account.	Adjust the presentation of the types involved as appropriate.	
UK 63	5.2.6	2	Ed	Paragraph 2 is missing its number.	Add one.	
UK 64	5.2.7	3	Ed	A new name R is introduced for use in paragraphs 3 and 4. But R is the same as T.	Replace R with T and replace "the required result type (which, for convenience, will be called R in this description)" with "T".	
UK 65	5.2.7	8	Te	In the first two bullets we have "the result is a pointer (an lvalue referring) to". But para 2 makes clear that a dynamic_cast of an rvalue references produces a rvalue. (Can an lvalue refer to anything anyway?)	Replace "an lvalue referring to" with "reference", twice.	
UK 66	5.2.8	1	Te	typeid may return "an implementation-defined class derived from std :: type_info". The derivation must be public.	an implementation-defined class publicly derived from std :: type_info	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 67	5.2.9	1, 2, 3	Te	Paragraph 1 specifies when the result of static_cast is an lvalue; repeating it is unnecessary.	In para 2, delete "It is an lvalue if the type cast to is an lvalue reference; otherwise, it is an rvalue." and "The result is an rvalue.". In para 3, delete "The result is an lvalue if T is an lvalue reference type (8.3.2), and an rvalue otherwise."	
UK 54	5.2.10	3, 6	Te	Para 3: "The mapping performed by reinterpret_cast is implementation-defined.". Para 6: "... the result of such a pointer conversion is unspecified." Which is it?	In para 6, replace unspecified with implementation-defined. Alternatively, delete paragraph 3, since individual cases are labelled appropriately.	
UK 55	5.2.10	2	Ed	dynamic_cast and reinterpret_cast crossreference 5.2.11 without creating an extra note. The second half of the note is unrelated to the crossrefernce, and would serve as well in normative text.	Strike the note about definition of casting away constness, preserve the cross-reference. The second sentance on reintrepret_cast to its own type should move out of the note into the normative text.	
UK 56	5.2.10	5	Ed	The notion of safely derived pointers means this conversion may not be as safe in the revised standard as the original. It would be good to call attention to the changed semantics with a note.	Add: [Note: the result of such a conversion will not be a safely-derived pointer value (3.7.4.3) -- end note]	
UK 57	5.2.10	8	Ed	Conditionally supported behaviour gives a wide range or permission, so clarify relationship between safely-derived object pointers and function pointers in a note.	Add: [Note: In such cases, the implementation shall also define whether a safely-derived object pointer cast to a function pointer can be safely cast back -- end note]	
UK 58	5.2.11	9	Te	Casting from an lvalue of type T1 to an lvalue of type T2 using a reference cast casts away constness if a cast from an rvalue of type “pointer to T1” to the type “pointer to T2” casts away constness. That doesn't cover rvalue references.	Replace lvalue with "lvalue or rvalue" twice.	
US 34	5.3	1	ed	The list of unary operator should be in teletype font.		
UK 68	5.3.1	2-9	Te	All the unary operands other than * return rvalues - but this is not stated.	Add a paragraph 1a "The following unary operators all produce results that are rvalues."	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 69	5.3.1	2	Te	If we cannot bind references/take address of functions in concept_maps, does that mean we cannot use generic bind in constrained templates? Launch threads with expressions found via concept map lookup? Hit problems creating std::function objects? Does the problem only occur if we use qualified lookup to explicitly name a concept map? Does it only kick in if we rely on the implicit function implementation provided by a concept_map, so some types will work and others won't for the same algorithm?!	... unknown ...	
UK 70	5.3.3	1	Te	The sizeof operator shall not be applied to ... an enumeration type before all its enumerators have been declared We should allow enum E : int; sizeof(E).	Change "an enumeration type" to "an enumeration type whose underlying type is not fixed".	
UK 71	5.3.4	2	Te	The type of an allocated object wih the type specifier auto is determined by the rules of copy initialization, but the initialization applied will be direct initialization. This would affect classes which declare their copy constructor explicit, for instance. For consistency, use the same form of initiailization for the deduction as the new expression.	Replace T x = e; with T x(e);	
UK 72	5.3.4	7	Te	The library headers have been carefully structured to limit the dependencies between core language and specific headers. The exception thrown should be catchable by a handler for a type lised in <exception> header in cluase 18. This might be accomplished by moving length_error into the <exception> header, but its dependency on logic_error with its std::string constructors suggest this is not a good idea. Prefer to pick an existing exception instead.	Throw std::bad_alloc instead of std::length_error.	
UK 73	5.3.4	6	Ed	A class type with conversion operator can only be used if the conversion type is constexpr and the class is a literal type. Adding the single word 'literal' before class type will clarify this.	Add 'literal' before 'class type'	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 74	5.3.4	8	Ed	operators, like constructors and destructors, do not have names. However, in certain circumstances they can be treated as if they had a name, but usually the stanadard is very clear not to actually describe their name as a distinct property.	Change "the allocation function's name is operator new" to "the allocation function is named operator new" and similarly for operator delete.	
UK 35	5.3.4	9	Ed	Missing period in middle of paragraph between "in the scope of T" and "If this lookup fails"	Add a period between "in the scope of T" and "If this lookup fails"	
UK 75	5.3.5	8	Ed	A paragraph strarting with [Note... is easily skipped when reading, missing the normative text at the end.	Swap order of the note and normative text.	
FR 21	5.3.6 [Alignof		te	Should not the type of alignof-expression be of type std::max_align_t?		
US 35	5.8	2 and 3	ed	There is curious spacing in the expressions "E1 <<E2" and "E1 >>E2". This is a formatting change since previous versions of the Standard.		
UK 47	5.14 / 5.15	2	Ed	Why are the descriptions of order of evaluation of expressions and side effects different between && and || operators. The interaction with the memory model should be identical, so identical words should be used to avoid accidential inconsistencies in interpretation.	Pick one form of wording as 'the best' and apply it in both places.	
UK 48	5.18	1	Ed	The defining feature of the comma operator is the guaranteed sequencing of two expressions. This guarantee is lost when presented with an overloaded operator, and this change is subtle enough to call attention to it.	Add: [Note: There are no guarantees on the order of value computation for an overloaded comma operator -- end note]	
UK 49	5.19	2	Te	Is an implementation permitted to reject this? constexpr int f() { return f(); } int a[f()]; AFAICT it is well-formed; f() seems to satisfy all the rules to make it a constant expression. I would hate compilation to become a potentially non-terminating experience.	Add an escape clause to allow the implementation to reject excessively deep nesting of constexpr function evaluations. (This can possibly be a note, since it is arguable that this point is handled by the general rule on resource limits in 1.4/2. A sufficiently smart compiler could use tail recursion above, meaning that it would never run out of	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					memory given this program though.)	
UK 50	5.19	2	Te	The following should be valid: enum E { foo = 4}; const E c = foo; int a[c]; But currently it is not - c is not an lvalue of effective integral type (4th bullet). (See also 7.1.6.1/2)	Change "effective integral type" to "effective integral or enumeration type" in the 4th bullet, 1st sub-bullet.	
UK 51	5.19	2	Te	typeid expressions can never be constant, whether or not the operand is a polymorphic class type. The result of the expression is a reference, and the typeinfo class that the reference refers to is polymorphic, with a virtual destructor - it can never be a literal type.	Strike the words "whose operand is of a polymorphic class type" on the bullet for typeid expressions.	
UK 76	6.3		Ed	Do we really need two different terms that say the same thing?	Pick either 'block' or 'compound statement' as the preferred term and use it consistently throughout the standard.	
FR 22	6.4.2 [The switch statement]		te	The constant-expression in case constant-expression should be allowed to be of any constant expression of literal type for which a constexpr comparison operator (operator< and operator==) is in scope. Now that constant expressions of other integral types are evaluated at compile time, the restriction for case-labels is at best artificial.		
UK 77	6.5	5	Ed	The terms i/o operation, synchronize operation and atomic operation have very specific meanings within the standard. The paragraph would be much easier to understand with the terms crossreferenced.	Profide a cross-reference for the terms: i/o operation, synchronize operation and atomic operation	

JP 11	6.5.4	1st paragraph, 5th line	ed	There is no _RangeT type in the equivalent code to “range-base for” statement. It existed in N2049.	Add a typedef for _RangeT in the example as follows:
					
					{
					typedef decltype(expression) _RangeT;	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					auto && __range = (expression);
					for (auto __begin = std::Range<_RangeT>::
					begin(__range),
					__end = std::Range<_RangeT>::
					end(__range);
					__begin != __end;
					++__begin)
					{
					for-range-declaration = *__begin;
					statement
					}
					}	

UK 78	6.5.4	2	Te	Including the header <iterator_concepts> is far too unwieldy to enable an important and (expected to be) frequently used syntax.	Merge <iterator_concepts> into <concepts> and change 6.5.4p2 to refer to <concepts>, or make the Range concept fundamental along with the other support concepts in 14.9.4 and strike any reference to including a header.	
UK 79	6.5.4		Te	The definition of for (for-range-declaration : expression) statement is expanded in terms which require a Range concept, and the program is ill-formed if <iterator_concepts> isn't included. For users, iterating through old-fashioned arrays, this is a sledge-hammer to crack a nut and compares poorly with other languages. It's also not possible to implement this without adversely impacting the freestanding definition in 17.6.2.4.	When expression is an array a of length N whose length is known at compile time, expand range-for as 'for (... p=a, p!=a+N, p++) ...' without requiring the Range concept or <iterator_concepts>. Also, when expression is an initializer_list, expand range-for similarly without requiring <iterator_concepts>.	
DE-11	6.9	p1	te	DE-11 A sentence in paragraph 1 reads: "Outside of a constrained context, the late-checked block has no effect." This, at face value, specifies that the *compound-statement* of such a late-checked block is never executed, which appears to be unintended.	State that such a late-checked block has the same meaning as if the late_check keyword were absent.	
UK 80	7	1	Ed	Many of the sections and major subsections open with a sentence summarising the content. I'm not sure this is	Strike the first sentence.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				necessary; this isn't a tutorial. The problem with these summaries is that because they omit much of the detail, they tend to be inaccurate. This may not matter, but I feel the document would be improved by omitting them. There's a prime example here: "Declarations specify how names are to be interpreted." Not true for static_assert, an asm declaration nor an anonymous bit field.		
UK 81	7	4	Te	String literal concatenation happens in phase 6, before parsing, so it is legal and useful to use it for the string literal in a static_assert. It would be useful to add a note mentioning this.	Add a note: Multiple adjacent string literals may be used instead of a single /string-literal/; see [lex.phases].	
UK 82	7	2	Te	Paragraph 2 talks about declarations that can have nested declarations within them. It doesn't mention scoped enumerations - but according to 7.2/11, "Each scoped enumerator is declared in the scope of the enumeration."	Add "scoped enumeration" to the list in the second sentence.	
UK 83	7.1	2	Te	The longest sequence of decl-specifiers that could possibly be a type name is taken as the decl-specifier-seq of a declaration. But many sequences of decl-specifiers cannot possibly be a type name - eg the sequence "friend int", or "typedef int".	Not sure. I understand the rule, just not how to say it.	
UK 84	7.1	1	Te	The grammar includes alignment-specifier as a production for decl-specifier, but there is no production for alignment-specifier. I suspect this is a holdover from before alignment was handled as an attribute.	Delete the production (including the duplicate in A6)	
FI 3	7.1	[dcl.spec.auto]	te	While it's considered too late for this standard revision, consider loosening the restrictions for auto specifier and making it more a mirror of a deduced template function parameter.	See restricted-auto.ppt	
UK	7.1.1	1	Ed	... the init-declarator-list of the declaration shall not be	Replace "global" with "namespace scope".	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
85				empty (except for global anonymous unions, which shall be declared static). Global here means "declared at namespace scope". (cf 9.5/3 "Anonymous unions declared in a named namespace or in the global namespace shall be declared static.").		
UK 86	7.1.1	2,3	Te	The register keyword serves very little function, offering no more than a hint that a note says is typically ignored. It should be deprecated in this version of the standard, freeing the reserved name up for use in a future standard, much like auto has been re-used this time around for being similarly useless.	Deprecate current usage of the register keyword.	
UK 87	7.1.1	1, 4, 5	Te	Why require two keywords, where one on its own becomes ill-formed? thread_local should imply 'static' in this case, and the combination of keywords should be banned rather than required. This would also eliminate the one of two exceptions documented in 7.1.1p1.	Drop requirement to combine static keyword with thread_local at block-scope inside a function definition.	
US 36	7.1.1	4	te	The permission to use thread_local static data members is missing.	Add the static members as a permitted use.	
FR 23	7.1.5 [constexpr]		te	'constexpr' functions should be allowed to take const reference parameters, as long as their uses are in a context where a constant expression may be required. For example, the following should be allowed template<typename T, int N> int size(const T(&)[N]) { return N; } int a[] = { 41,42,43,44 }; enum { v = size(a) };		
JP 12	7.1.5		te	It should be allowed to define constexpr recursively. There is an explanation in N2235, Generalized Constant Expressions—Revision 5, as follows.	Allow constexpr recursion.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				We (still) prohibit recursion in all its form in constant expressions. That is not strictly necessary because an implementation limit on recursion depth in constant expression evaluation would save us from the possibility of the compiler recursing forever. However, until we see a convincing use case for recursion, we don't propose to allow it. Then, here are the use cases where allowing recursion for constexpr is very useful. Range of problem to be handled with constexpr would become extended. For example, user defined type (e.g. Complex type) could be placed in ROM area. But with current specification, a function defined with constexpr cannot be called recursively. As a side effect is not allowed in compile-time, it cannot be implemented to repeat anything without recursion. Although it could be implemented without recursion like func0, func1, func2 in an example below, it is not elegant solution. constexpr double func0(double x) { /* ... */} constexpr double func1(double x) { /* call for func0 */ } constexpr double func2(double x) { /* call for func1 */ } /* ... */ - Compile-time and runtime As constexpr can be also evaluated both in compile-time and runtime, we need to discuss about both cases. Runtime evaluation is just to execute it. If you eliminate constexpr keyword, it is executable as of now. Any modern compiler may optimize tail recursion easily. Compile-time evaluation is the same thing as template		

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

				recursion. It is necessary to support floating point operation, but it is already possible to calculate it in compile-time, so it's ok.
				
				- Sample
				Here is an example to calculate a square root using constexpr recursively.
				
				/*constexpr*/ double SqrtHelper(double x, double a, int n)
				{
				return n == 0 ? a : SqrtHelper(x, (x / a + a) / 2.0, n - 1);
				}
				
				/*constexpr*/ double Sqrt(double x)
				{
				return SqrtHelper(x, x, 20);
				}
				
				/*constexpr*/ double root2 = Sqrt(2.0); // 1.41421...		

US 37	7.1.6.1	1	ed	There is a "Note: 3.9.3 describes how cv-qualifiers affect object and function types." So far as I can see, 3.9.3 CV-qualifiers only describes cv-qualifiers for objects, cv-qualifiers for (member) functions being described in 8.3.5 Functions.		
UK 89	7.1.6.1	2	Te	The two normative sentences in this paragraph appear to duplicate text elsewhere - but they aren't exact duplicates, which introduces uncertainty. 1. "An object declared in namespace scope with a const-qualified type has internal linkage unless it is explicitly declared extern or unless it was previously declared to have external linkage.". This nearly repeats 7.1.1/7: "Objects declared const and not explicitly declared extern have internal linkage." The former seems to allow more wiggle room - can an object be "previously declared to have external	Make the normative text in this section into one or more notes, with cross references, and correct the referenced text if necessary.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				linkage" without having been "explicitly declared extern"? 2. "A variable of non-volatile const-qualified integral or enumeration type initialized by an integral constant expression can be used in integral constant expressions (5.19)." This nearly duplicates 5.19/2, bullet 4, 1st sub-bullet - "[... an integaral constant expression can use] an lvalue of effective integral type that refers to a non-volatile const variable or static data member initialized with constant expressions". The latter does not allow for lvalues of enumeration type (neither scoped not unscoped enumerations are integral types - 3.9.1/7, and note 44). This seems to be a flaw in 5.19/2.		
UK 90	7.1.6.2	para 1 and table 9	Ed	The grammar in paragraph one makes "nested-name-specifier template simple-template-id" a simple-type-specifier, but unlike all the others it is omitted from table 9.	Add a row to table 9 mentioning simple-template-id and punting to clause 14 (cf decltype(expression)).	
UK 91	7.1.6.2	4	Te	5.1/5 says "[A] parenthesized expression can be used in exactly the same contexts as those where the enclosed expression can be used, and with the same meaning, except as otherwise indicated." When the first bullet point of this paragraph, describing the type denoted by decltype(e), says "if e is an id-expression ... decltype(e) is the type of the entity named by e", 5.1/5 is not excluded, which would imply that decltype((e)) was also the type of e. But the intention appears that it should be caught by the third bullet and treated as an lvalue expression, so decltype((e)) should be a reference to the type of e. Conversely, the second bullet point says "(parentheses around e are ignored)", which is redundant because of 5.1/5.	Insert "unparenthised" in the first bullet point - "if e is an *unparenthised* id-expression ...". In the second bullet point, move "(parentheses around e are ignored)" into a note.	
UK 92	7.1.6.3	2	Ed	The note correctly indicates that, if T is a template type parameter, then "friend class T;" is ill-formed. It might be worth pointing out at the same time that the alternative "friend T;" is now allowed - see 11.4/3.	Either strike the note or add reference to 11.4/3 and/or mention of "friend T;".	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 93	7.1.6.3	Grammar before para 1	Ed	In the third production, "enum ::opt nested-name-specifieropt identifier", enum should not be in italics; its referring to the enum keyword.	Change to keyword font	
UK 94	7.1.6.4	1	Ed	The auto type-specifier signifies that the type of an object being declared shall be deduced from its initializer or specified explicitly at the end of a function declarator. A function declarator does not declare an object.	The auto type-specifier signifies that the type of an object being declared shall be deduced from its initializer or that the return type of a function is specified explicitly at the end of a function declarator.	
UK 95	7.1.6.4	4	Te	(See also c++std-core-13583) This paragraph allows auto "in the type-specifier-seq in a new-type-id (5.3.4)" (and nowhere else not listed). Specifically, it isn't allowed in a type-id in a new-expression. That allows "new auto (42)", but not "new (auto)(42)". However, 5.3.4/2 suggests the latter should be allowed "If the auto type-specifier appears in the type-specifier-seq of a new-type-id or type-id of a new-expression ...". The inconsistency should be resolved, ideally in favour of allowing both forms.	Change "in a new-type-id" to "in a new-type-id or type-id in a new-expression".	
FR 24	7.1.6.4 [auto specifier]		te	Now that 'auto' is finally used in its most obvious sense to state `deduce the type of this variable from initializer', it should also be allowed in template parameter declarations, as in template<auto n> struct X { /* … */ }; X<903> x; X<&Widget::callback> y; instead of the current, often verbose and cumbersome template<typename T, T n> struct X { /* … */ }; X<int,903> x;		

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				X<void (Widget::*)(),&Widget::callback> y; We understand that 'auto' is used in 14.1/18 in a different way (for constrained template), but that usable appears very strange syntax, unnatural and does not fit well with the usage in this section.		
US 38	7.2	1	ed	The discussion of attribute specifiers should be a separate paragraph.		
US 39	7.2	2	te	The paragraph says in part "An opaque-enum-declaration declaring an unscoped enumeration shall not omit the enum-base." This statement implies that the base may be omitted for scoped enumerations, which is somewhat inconsistent with paragraph 3 and somewhat consistent with paragraph 5.	As this implication leaves no representation, it should be either affirmed here or the statement should be expanded. Perhaps a note is warranted.	
JP 13	7.2	paragraph 3	ed	In the description for an unscoped enumeration, enum-base in redeclaration must be the same underlying type as in the 1st declaration, but it is not described explicitly, while it is referred that all enum-bases in redeclarations must specify the same underlying type.	Replace the description, "same underlying type", with "same as underlying type of (previous) declaration."	
UK 96	7.2	7	Te	enum E { }; What are the values of E? It has neither a smallest nor largest enumerator, so paragraph 7 doesn't help. (Paragraph 6 indicates that the underlying type is as if E had a single enumerator with value 0, but that does not define the values of E.)	Add a second sentence to paragraph 7 (before "Otherwise"): "If the enumerator-list is empty, 0 is the only value of the enumeration."	
UK 97	7.2	9	Ed	Missing punctuation after "blue" in: "The possible values of an object of type color are red, yellow, green, blue these values can be converted ..."	Add a semicolon: "The possible values of an object of type color are red, yellow, green, blue; these values can be converted ..."	
UK 98	7.2	5	Te	It would be useful to be able to determine the underlying type of an arbitrary enumeration type. This would allow safe casting to an integral type (especially needed for scoped enums, which do not promote), and would allow	Add a TransformationTrait to 20.5.6 that returns the underlying type of an enumeration type.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				use of numeric_limits. In general it makes generic programming with enumerations easier.		
UK 99	7.2	3	Te	It is unclear whether an enumeration type is complete after an opaque-enum-declaration. This paragraph only says so in a note, and the general rule in 3.9/5 ("Incompletely-defined object types ... are incomplete types") is unclear in this situation.	Move "an enumeration declared by an opaque-enum-declaration ... is a complete type" from the note to normative text.	

JP 14	7.3.1		te	The description of the behavior when a member that was defined with same name in other namespace was referred.
				- It seems that the behavior of the following case is not defined. So we think that it is necessary to define that.
				namespace Q {
				inline namespace V {
				int g;
				}
				int g;
				}
				Q::g =1;// ill-fromed, Q::V::g =1;, or Q::g = 1;?
				- Add that the following case is ill-formed to more easily to understand.
				namespace Q {
				inline namespace V1{
				int g;
				}
				inline namespace V2{
				int g;
				}
				}
				Q::g =1;//ill-formed
					Add the description of the behavior when a member that was defined with same name in other namespace was referred.	

UK 100	7.3.3	10 and 13	Ed	Para 10 says "A using-declaration is a declaration and can therefore be used repeatedly where (and only where) multiple declarations are allowed." Para 13 says "Since a	Delete para 10, moving its example into para 13.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				using-declaration is a declaration, the restrictions on declarations of the same name in the same declarative region (3.3) also apply to using-declarations." These appear to be saying exactly the same thing.		
UK 101	7.3.3	20	Te	If a using-declaration uses the keyword typename and specifies a dependent name (14.6.2), the name introduced by the using-declaration is treated as a typedef-name (7.1.3). That doesn't specify at all what the effect of using typename with a non-dependent name is. Is it allowed? What about outside any template? What if the name isn't a type? (14.6/4 doesn't cover this, I think.)	Allow typename for non-dependent names iff they refer to a type.	
DE-12	7.3.3	p15	te	DE-12 Overriding and hiding of member functions named in using-declarations should consider ref-qualifiers, because they are part of the function type.		
FR 25	7.3.3 [The using declaration]	Paragraph 21	te	The syntax for concept map alias is unnecessarily both confused and verbose.	We strongly suggest simplifications, e.g. using N1::C<int>; that fits well with existing constructs. The syntactic complexity is too high for a new feature presumably designed to support sound programming.	
UK 102	7.3.4	6	Ed	This paragraph says "If name lookup finds a declaration for a name in two different namespaces, and the declarations do not declare the same entity and do not declare functions, the use of the name is ill-formed." But the example uses declaration of functions, so is not covered by this paragraph.	Move the example to paragraph 7, and/or replace it with an appropriate example.	
US 40	7.6		te	The list of attributes is missing an attribute to indicate that a function with a throw() (throws nothing) clause need not have the unexpected() catch clause generated. This attribute was a motivating example for the attribute syntax, and its omission is surprising.	Add the attribute.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
US 41	7.6		te	A common problem is unintentionally declaring a new virtual member function instead of overriding a base virtual member function.	An attribute stating intent to override would enable better diagnostics.	
FR 26	7.6 [Attributes]		ed	Are they part of object types or not? The section does not appear to indicate that clearly.		
FI 1	7.6		te	Add override-attribute for functions in order to avoid mistakes when overriding functions.	See override-attribute.doc, override-attribute.ppt	
FR 27	7.6.1		te	This section specifies that no name lookup is performed on any identifier contained in an attribute-token. This in particular implies that, for example, it is impossible to define a template class parameterized by its alignment. That restriction is unacceptable. The original alignment proposal made that useful construct possible. Furthermore paragraph 7.6.1/2 appears contradictory with the rest of that section -- since no name lookup is performed, how a 'type-id'is determined?		
UK 103	7.6.1		Te	Attributes should support pack expansion. For example, this would be extremely useful with the align attribute, directly supporting the (removed) functionality of aligned_union. NOte that aligned_union was removed as varaiant-unions were considered a complete replacement - however this is not true for variadic templates. Adding this support to attributes would remove the remaining need, and support similar attributes in the future.	Add: attribute... to the grammar for attribute-list Add to list in 14.5.3p4: "In an attribute-list(7.6.1); the pattern is an attribute."	
UK 104	7.6.1	1	Ed	It is helpful for each subclause to contain a short paragraph introducing its intent an purpose. 7.6 has such a paragraph, but it is nested under a more specific subsection.	7.6.1p1 should move up one level to become 7.6p1. There grammar should remain under 7.6.1	
UK 105	7.6.1	1	Te	Allowing only one level of namespaces in attributes seems unnecessarily limiting.	To: attribute-scoped-token: attribute-namespace :: identifier add attribute-namespace :: attribute-scoped-token	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 106	7.6.2	1	Ed	Extensive use of alignment and related terms without cross reference.	Add cross-reference to 3.11.	
JP 15	7.6.2		ed	An abbreviation of 7.6.2 should be “[decl.attr.align]” instead of “[dcl.align]”. Section name “[dcl.align]” is not consistent with others, because others in 7.6 are the form of “dcl.attr.*”. In N2761, the section name of 7.1.7 had been changed from “[dcl.align]” to “[dcl.attr.align]”, but in N2800 it was reverted to “[dcl.align]” along with a change of section number, 7.1.7 to 7.6.2.	Change "[dcl.align]" of 7.6.2 to "[decl.attr.align]".	
UK 107	7.6.3		Ed	While undefined behaviour might be the best we can guarantee, it would be helpful to encourage implementations to diagnose function definitions that might execute a return.	Adda a [Note : implementations are encouraged to issue a diagnostic where the definition of a function marked [[noreturn]] might execute a return statement -- end note]	
UK 108	7.6.4	2	Te	It is unclear why no diagnostic is required for an easily detectable violation. It is even more surprising that the associated footnote mandates behaviour for an ill-formed program.	Strike "no diagnostic required" and the associated footnote.	
US 42	7.6.4		te	The meaning of the [[final]] attribute applied to classes is inconsistent with other languages and not desirable in its own right.	Modify the semantics of [[final]] applied to classes. See the attached paper "Issues with the C++ Standard" under Chapter 7 "Meaning of [[final]] attribute applied to classes".	
UK 109	7.6.5	4	Ed	The example code refers in comments to "Compilation unit" A and B. The term should be "Translation unit" (2/1)	Replace "Compilation" with "Translation" in two places	
UK 110	7.6.5	4	Te	The code in the example (compilation unit A) has: "foo_head[i].load(memory_order_consume)". foo_head[i] is of type foo *, so it does not have a load member.	Change the type of foo_head to atomic<foo *>[].	
US	8		te	With the introduction of late-specified return types for	Some simplification is needed.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
43				functions and lambda expressions, we now have three different syntaxes for declaring functions. The -> late declaration is used in two. The auto keyword is used in one, but also used differently in variable definitions.		
UK 111	8.3.5	13	Ed	Example missing closing bracket in template<typename... T> void f(T (* ...t)(int, int);	Add closing bracket like this: template<typename... T> void f(T (* ...t)(int, int));	
US 44	8.3.5	13	ed	In the Example, "template void f(T (* ...t)(int, int);" is missing a close parenthesis.	It presumably should read: "template void f(T (* ...t))(int, int);".	

US 45	8.3.5	13	te	At present, function parameter packs can only occur at the end of a parameter-declaration-list. This restriction unnecessarily prohibits uses of function parameter packs in cases where template argument deduction isn't needed, e.g.,
				
				template<class... T> struct X { };
				template<class... T1, class... T2>
				struct X<pair<T1, T2>...> {
				void f(T1..., T2...);
				};
				
				More importantly, this restriction is inconsistent with the way pack expansions are handled. For example, this template is well-formed (but X<T..., int> is a non-deduced context):
				
				template<class... T> void f(X<T..., int>);
				
				Therefore, the restriction that limits function parameter packs to the end of the parameter-declaration-list should be removed. Instead, function parameter packs not at the end of the parameter-declaration-list should be considered non-deduced contexts.
					In 8.3.5p13, remove the sentence "A function parameter pack, if present, shall occur at the end of the parameter-declaration-list."
					
					In 14.8.2.1p1, replace the phrase "For a function parameter pack" with "For a function parameter pack that occurs at the end of a *parameter-declaration-list*".
					
					Replace the note text "A function parameter pack can only occur at the end of a parameter-declaration-list (8.3.5)." with "A function parameter pack that does not occur at the end of a parameter-declaration-list is a non-deduced context."
					
					In 14.8.2.5p5, add a new bullet: "A function parameter pack that does not occur at the end of its parameter-declaration-list."
					
					In 14.8.2.5p10, replace "If the parameter-declaration corresponding to Pi is a function parameter pack" with "If the parameter-declaration corresponding to Pi is a function parameter pack and Pi occurs at the end of the parameter-declaration-list".	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					Replace the note text “A function parameter pack can only occur at the end of a parameter-declaration-list (8.3.5).” with “A function parameter pack that does not occur at the end of a parameter-declaration-list is a non-deduced context.”	
DE-13	8.4	p2	te	DE-13 The second paragraph, quoting the grammar for the declarator of a function declaration, is not considering late-specified return types and attributes.	Properly quote the grammar from 8.3.5.	
JP 16	8.5	15th paragraph, 1st line	ed	Typo, duplicated "in" "The initialization that occurs in in the forms"	Remove one.	

US 46	8.5.3		te	The ability for an rvalue reference to bind to an lvalue opens a type-safety hole that becomes very dangerous with concepts. For example, consider vector’s push_back operation:
				requires MoveConstructible<T> void push_back(T&&);
				requires CopyConstructible<T> void push_back(const T&);
				
				For a copy-constructible T (which is also move-constructible), push_back does the right thing. However, if T is something that is move-constructible (e.g., unique_ptr<int>), the second overload is removed from considered (it is effectively SFINAE’d away), so only the first overload remains. Therefore, one could accidentally call push_back with an lvalue of type T, and push_back would silently move from the lvalue. The same problem occurs without concepts (albeit less frequently).
					Prohibit rvalue references from binding to lvalues.
					
					Unfortunately this change will break some current use cases of rvalue reference including the use of rvalue streams, and of the forward function itself. To resolve this we may want to consider three types of references:
					
					The current reference.
					A non-const reference that only binds to rvalues.
					A non-const reference that will bind to both lvalues and rvalues.
					
					Still another solution would be to adopt the “deleted function” solution for all functions. This solution is described in comment for 12.1, 12.4, 12.8, but restricted to special functions in that comment. (See US NN).	

US 49	8.5.4	6	ed	In the Example, the comments could be improved.	See the attached paper "Issues with the C++ Standard" under "Editorial Issues" and "8.5.4/6".	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 112	9	4-9	Ge	We now have concepts that should (or should not?) map to the terms described in Clause 9 - these should be at least referenced.	Add appropriate forward references to 14.9.4	
UK 113	9.4.2	3	Ed	Mis-applied edit from the paper n2756	The term 'constant-initializer' should have been struck out when replaced by brace-or-equal-initializer. There are two occurrences in this paragraph	
US 50	12.1, 12.4, 12.8		te	Implicitly-declared default constructors, destructors, copy constructors, and copy assignment operators are deleted when their definitions would be ill-formed. However, unlike with overloading and template argument deduction, access control is performed as part of the check for making one of these special function deleted. This inconsistency should be removed. This change would sacrifice some backward compatibility in favor of consistency. With the current wording, checking that the following class 'A' is CopyConstructible would proceed without error (it is not CopyConstructible): class A { A(const A&); }; With the proposed change, testing whether A is CopyConstructible would produce a diagnostic. To fix the problem, the user would have to use a deleted function: class A { A(const A&) = delete; };	In 12.1p5, remove the phrase " or inaccessible from the implicitly-declared default constructor". In 12.4p3, remove the phrase "or a destructor that is inaccessible from the implicitly-declared destructor," and the phrase "or a destructor that is inaccessible from the implicitly-declared destructor". In 12.8p5, remove the phrase " or inaccessible from the implicitly-declared copy constructor" from the two places it occurs. In 12.8p10, remove the phrase "or inaccessible from the implicitly-declared copy assignment operator" from the two places it occurs.	
FR 28	12.6.1 [Explicit initialization]		te	This section, in particular the example with `g' appears contradictory with the syntax for uniform initialization.		
US 51	12.6.2	2	ed	The discussion of delegating constructors should be in its own paragraph.		
UK 114	12.6.2	1	Te	Despite all the attempts to unify initialization syntax, it is still not possible to copy-initialize base classes or non-static data members, which means the explicit keyword cannot have a bearing during evaluation of a constructor.	Ammend the grammar for mem-initializer: mem-initializer-id =OPT braced-init-list Extend p3 to allow for Copy Initialization if the optional = is present: 3 The expression-list or braced-init-list in	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				A minimal addition to the grammar, allowing an optional = between the mem-initializer-id and braced-init-list would allow the user to choose between copy and direct initialization	a mem-initializer is used to initialize the base class or non-static data member subobject denoted by the mem-initializer-id according to the initialization rules of 8.5 for direct-initialization, OR COPY-INITIALIZATION IF THE OPTIONAL = IS PRESENT BETWEEN THE MEM-INITIALIZER-ID and the BRACED-INIT-LIST. [Example:...	
US 52	13.5.8	¶ 5	ed	A word is misspelled.	Change “shal” to “shall”.	
UK 115	14	6-11	Ge	Exported templates were a great idea that is generally understood to have failed. In the decade since the standard was adopted, only one implementation has appeared. No current vendors appear interested in creating another. We tentatively suggest this makes the feature ripe for deprecation. Our main concern with deprecation is that it might turn out that exported constrained templates become an important compile-time optimization, as the constraints would be checked once in the exported definition and not in each translation unit consuming the exported declarations.	Consider deprecating exported templates, but no action yet. Examine interaction with constrained templates, and see if other more appropriate mechanism will support compile-time optimization.	
UK 116	14	6-11	Te	Is it possible to export a concept map template? The current wording suggests it is possible, but it is not entirely clear what it would mean.	Either prohibit exporting concept map templates, or more directly address what it means to export a concept map.	
UK 117	14	2	Ge	It would be nice to allow template alias within a function scope, and possibly a scoped concept map. As these affect name lookup and resolution, rather than defining new callable code, they are not seen to present the same problems that prevented class and function templates in the past.	Allow template aliases to be declared inside a function scope, and consider scoped concept maps.	
UK 118	14	6-11	Ed	Exported templates are a complicated feature with surprisingly little text. To make this important text more visible, split it off into its own subclause [temp.export]	Create a new subclause [temp.export] containing 14p6-11. Move 14p12 ahead of this subclause.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 119	14	4	Te	Does a concept map have linkage? Reading this paragraph and 3.5 suggests a concept map template has external linkage, but not a 'regular' concept map. Believe this is an oversight that the linkage words were not updated to provide an exception, rather than linkage of concept maps is intended.	Add an exception that concept map templates have no linkage, or add concept maps to the list of entities with linkage in 3.5	
UK 120	14.1	9	Ed	As this is the first time the phrase “parameter pack” appears in Ch 14 I would like to see the section 8.3.5 referenced here (as well as in 14.1p17).	Insert “(8.3.5)” after “parameter pack”	
UK 121	14.1	18	Ed	The example (that follows the normative text) has no begin example marker	Prefix the example code with "[Example:"	
FR 29	14.3 [Template arguments]		te	Constant expressions of any literal type should be allowed as template arguments.		

US 53	14.5.1	5	te	If the requirements of a constrained member that is a copy constructor, copy assignment operator, or destructor are not satisfied, then that user-declared special function will not exist. It appears that, in this case, the special function will then be *implicitly* defined, which is likely to either (a) fail to compile or (b) produce a function with the wrong semantics. For example:
				template<ObjectType T> class vector {
				T* first, last, end;
				public:
				requires CopyConstructible<T> vector(const vector&);
				};
				
				If instantiated with a type that is not CopyConstructible, vector will get an implicitly-defined copy constructor that performs a copy of the pointers.
					Add to 14.5.1p5:
					If the constrained member is a copy constructor (12.8), destructor (12.4), or copy assignment operator and its template requirements are not satisfied, then a copy constructor, destructor, or copy assignment operator, respectively, with the same signature as the constrained member (after substituting the class template’s template arguments for its template parameters) will be declared as a deleted function (8.4).	

UK 122	14.5.3	4	Te	Variadic templates should be supported in axioms. There are axioms in the library that rely on this feature, such as the FrontEmplacement axiom in	Add clarification in p2 that function parameter packs can be used to declare axioms, much like p1 clarifies they can be used to declare concepts	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				FrontEmplacementContainer (23.1.6.1p10)	as well as templates.	
FR 30	14.5.7 [Template aliases]		te	When are two template alias names equivalent? E.g. given template<template<class> class> struct X { }; template<typename,typename> struct Y { }; template<typename T> using Z1 = Y<int,T>; template<typename T> using Z2 = Y<int,T>; Are the types X<Z1> and X<Z2> equivalent? We would suggest yes (since Z1<T> and Z2<T> are the same for all T), but we do not see any wording to that effect.		
JP 17	14.7.2	2nd paragraph, 15th line	ed	Typo. if that namespace is inline, any namespace from its enclosing namespace set. should be if that namespace is inline, any namespace forming its enclosing namespace set.	Replace "from" with "forming"	
DE-14	14.7.3	p1	te	DE-14 The bulleted list neither addresses "member function template of a class" nor "member class template of a class".	Add the respective bullets.	
JP 18	14.7.3	2nd paragraph, 2nd line	ed	Typo, any namespace from its enclosing namespace set should be	Replace "from" with "forming"	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				any namespace forming its enclosing namespace set		
JP 19	14.8.2	6th paragraph, 1st line	ed	Typo, duplicated "is" "At certain points in the template argument deduction process it is is necessary"	Remove one	
US 54	14.9 [concept], 14.10 [temp.constrained]		ge	Concepts is of course the largest new feature in C++0x (in terms of new text inserted into the wording), and already we have found some significant defects with it. So far nothing devastating has been found, but more time is needed to shake more bugs out.	I propose no specific change here.	
US 55	14.9.1	¶ 6	ed	The paragraph number is in the wrong place, causing a grammar rule to be indented more than its fellows.	Move the paragraph number so as to follow the grammar rules, thus numbering the single sentence, “The body of a concept … .”	
US 56	14.9.1	¶ 6	ed	The sentence contains two references to 14.9.1.3 [concept.req].	Change the second such reference (at the end of the sentence) to 14.9.1.4 [concept.axiom].	
US 57	14.9.1.4	¶ 3	ed	A word is misplaced, changing the intended meaning.	Change “only find … if” to “find … only if”.	
US 58	14.9.1.4	¶ 3	ed	The listed phrases are not grammatically parallel.	Insert “in” before “one” so as to obtain “... in the concept, in one of its less refined concepts, or in an associated requirement.”	
US 59	14.9.1.4		te	Axioms are under-specified and provide little benefit to programmers, so they should be removed from the working paper. The optimizations permitted by axioms (see 14.9.1.4p4-5) are not compulsory (and, therefore, programmers cannot rely on them) and the semantics expressed by axioms cannot be verified by any implementation. The resulting specification has lead to great confusion (see the reflector thread “Are floating point types Regular?” starting with c++std-lib-22717). Given the level of confusion and the inability to verify the correctness of axioms, it is likely that many axioms	Remove clause 14.9.1.4 [concept.axiom] In 2.11p1, remove “axiom” from the list of keywords. In 14.5.8p7, remove “, or if the resulting concept map fails to satisfy the axioms of the corresponding concept” In 14.9.1p6, remove axiom-definition from the list of grammar productions for concept-member-specifier. Remove “, and axioms” from the final	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				written by programmers (including those specified in the candidate draft) will be incorrect.	sentence, and instead “and” prior to “associated requirements” in the final sentence. Remove paragraph 14 of clause 14.9.2. In 14.10.1p6, remove the sentence, “When the concept-instance-alias-def appears in the optional requires-clause of an axiom-definition (14.9.1.4), the potential scope of the identifier begins at its point of declaration and terminates at the end of the axiom-definition.” In clauses 20.2.5, 20.2.8, 23.1.6.1, 23.1.6.2, and 24.1.4, remove the axiom-definitions and replace them with paragraphs (denoted Requires, Postconditions, or Effects, as appropriate) that express the intended semantics of the concepts from which the axiom-definitions were removed. In 24.1.4p2, replace the word “axiom” with “condition.”	
FR 31	14.9.1.4 [Axioms]		te	This section states that an axiom-definition defines a new semantics axiom but is unusually vague as to what those semantics might be. The use of the '==' and '!=' with completely new semantics, unrelated to anything we have seen before in C++ is both unwise and confusing, especially if the types involved in the expressions happen to have operator== and operator!= defined. We strongly suggest use of different tokens, e.g. ⇔, and opposed to this obscure usage/overload. The description is very vague. How many times is an implementation permitted to replace one expression by another one when they have side effects?		

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
DE-15	14.9.1.4		te	DE-15 There is no implementation experience for axioms. Use of axioms is an area of active scientific research. It is likely that syntax changes will become necessary to make good use of axioms; having the syntax space already crowded is unhelpful. Axioms ought to be useful in concepts applicable to floating-point types (such as EqualityComparable), but IEEE floating-point types have special values such as NaN violating the axioms.	Remove section 14.9.1.4 and any reference to axioms in the rest of the proposed standard other than the keyword reservation in section 2.11.	
UK 123	14.9.1.4		Te	auto concepts and axioms are incompatible. An axiom defines the semantics of an operaton or set of operations that describes the run time behaviour. A concept describes purely syntactic requirements at compile time. Where an auto concept will match anything that meets the syntax requirements, there is no way to know if the axioms will be met or not, and no way to opt out via some kind of negative concept map.	Add a paragraph making axioms ill-formed inside an auto concept.	
UK 124	14.9.1.4	6	Ed	Spelling mistake, double-e in were.	weere -> were	
UK 125	14.9.1.4	2	Te	The implicit equality comparison operator available to axioms has no semantic. It is not clear what expressing the condition if(a == b) { conditional-axiom } would mean if a and b are not truly EqualityComparable. We suspect the intent of the 'implicit defefinition' is to support declaring the equivalence of statements, a context where the operator will not actually be evaluated.	Define the semantics of the implicitly declared comparison operators, or restrict their usage to declaring equivalence between statements.	
UK 126	14.9.4	41	Ed	This paragraph contains the only definition of the underlying_type member - but it's a note, so not normative.	Move the second sentence to the Requires clause in paragraph 42.	
UK 127	14.9.4		Ed	Provide a diagram clearly showing refinement relationship between the different support concepts. Several were created during development of this clause and they were very helpful.	Provide the diagram.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 128	14.9.4	4	Ed	It is surprising for many people that non-copyable move-only types can be used with a return statement, and so Returnable does not always imply CopyConstructible.	A non-normative note: [Note: 'move only' types that are constructible from rvalue references may be Returnable, but not CopyConstructible(20.1.8) - end note]	
JP 20	14.9.4	2nd paragraph	te	Trivially copyable type was added in “3.9 Types”, so we think that it is necessary to add concept to trivially copyable type like “TriviallyCopyableType”.	Add TriviallyCopyableType that is trivially copyable type as concept.	
UK 129	14.10.1, 20.1.2		Te	It should be possible to support boolean constant expressions as requirements without resorting to defining the True concept in the library. Boolean expressions are very likely to be constraints when deadline with non-type template parameters and variadic templates, and constraints in these cases should feel just as natural as constraints on the type system.	Remove the True concept and library subclause 20.1.2. Provide support in 14.10.1 for boolean constant expressions as constraints. This may involve overloading the true keyword to disambiguate but ideally would not.	
US 60	14.10.1	1	te	The use of && as the separator for a list of requirements has shown itself to be a serious teachability problem. The mental model behind ‘&&’ treats concepts as simple predicates, which ignores the role of concepts in type-checking templates. The more programmers read into the ‘&&’ (and especially try to fake || with && and !), the harder it is for them to understand the role of concept maps. Simply changing the separator to ‘,’ would eliminate a significant source of confusion.	Replace requirement-list: requirement-list ... [opt] && requirement requirement ... [opt] with requirement-list requirement-list ...[opt] , requirement requirement ... [opt] In 14.5.4p6, replace the first sentence with: The instantiation of an expansion produces a comma-separated list E1, E2, ..., EN, where N is the number of elements in the pack expansion parameters.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 130	15.1	4	Te	With the new crrent_exception API it is possible to capture a reference to an exception that will outlive its last active handler. That is in conflict with the sentance "When the last remaining active handler for the exception exits by any means other than throw; the temporary object is destroyed and the implementation may deallocate the memory for the temporary object;"	Update sentence to allow for exceptions held in excpetion_ptr objects.	
UK 131	15.3	3	Te	A handler catching its parameter by rvalue-reference is syntactically valid, but will never be activated.	Disallow handlers catching by rvalue-reference.	
UK 132	15.3	16	Te	There are obscure cases whrere a copy constructor is not usually the best match to copy-initialize an object, e.g. A converting constructor template taking arguments by non-const reference. A footnote explaining such cases would be helpful, or the sentance could be rewritten using copy-initialization instead of directly calling a copy constructor.	Rewite using copy-initialization rather than directly invoking the copy constructor	
UK 133	15.4	2	Te	Template aliases have the same semantics as a typedef so should also be disallowed	add "or alias-declaration" after "shall not appear in a typedef declaration".	
UK 134	15.4	6	Ed	The sentance "An exception-specification can also include the class std::bad_exception (18.7.2.1)." is redundant.	Either strike the quoted sentance, or add a note explaining why it is worth calling special attention to this class.	
UK 135	15.4	8	Te	Unclear if std::unexpected is called before or after the function arguments have been destroyed	Clarify the sequence of calling unexpected with respect to interesting objects, such as function arguments or partially constructed bases and members when called from a constructor or destructor	
UK 136	15.4		Ge	Exception specifications have proven close to worthless in practice, while adding a measurable overhead to programs. The feature should be deprecated. The one exception to the rule is the empty throw specification which could serve a legitimate optimizing role if the requirement to call the runtime unexpected mechanism was relaxed in this case.	Move 15.4 and the parts of 15.5 that refer to it to Appendix D. Replace 15.4 with a simpler specification for empty throw specifications, where the std::unexpected call is conditionally supported allowing vendors to choose between optimizing and providing runtime checks. Ideally require vendors to provide a mode where the runtime	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					checks are always disabled.	
UK 137	15.5		Ed	There is no mention of the current_exception API which can extend the lifetime of an exception object. There should at least be a forward reference to the library clause 18.7.5	Add another paragraph outlining 18.7.5 and the ability of an exception_ptr to extend the lifetime of an exception object	
UK 138	15.5.1	1	Ed	The third bullet is redundant with the first, as it is a subset of the same conditions.	Merge the third bullet into the first bullet as a note or example.	
UK 139	15.5.1	1	Te	According to the first bullet it is perfectly alright for a library function to exit by throwing an exception during stack unwinding, This is clearly not true.	Strike the word 'user' from the first bullet point.	
UK 140	15.5.2	2	Ed	The detailed specification can fool people into thinking an exception will automatically be translated into bad_exception, where the default behaviour of std::unexcepted is to immediately call std::terminate();	Add a note highlighting the default behaviour of std::unexpected if the user does not supply a hander-function	
UK 141	15.6		Ed	This whole subclause is redundant due to 15.1p5 and 15.3p17	Strike 15.6	
UK 142	16.3.5	3	Ed	This paragraph opens with "[Note" but has no corresponding "end note]"	Add "end note]"	
UK 143	16.3.5	7	Ed	Example uses #define t(x,y.z) x ## y ## z	Change "x,y.z" to "x,y,z"	
US 2	17-30		ge/te	The active issues identified in WG21 N2806, C++ Standard Library Active Issues, must be addressed and appropriate action taken. http://www.open-std.org/jtc1/sc22/wg21/docs/lwg-active.html	Appropriate action would include making changes to the CD, identifying an issue as not requiring a change to the CD, or deferring the issue to a later point in time.	
FR 2	General Comment	Library	ge	The adoption of the library `constexpr' proposal was not reflected in the draft, despite formal WG21 committee vote.	FR 2	General Comment

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
US 61	17 onward		te	The concepts core language feature is applied to only some of the Standard Library clauses, and even then not always consistently.	Review all clauses of the Standard Library, and consistently apply concept technology wherever possible and appropriate. The proposed wording in WG21 N2781 exemplifies the necessary level of detail.	
CA-2	17 Library		Ge	“Concepts” are a significant new addition to the language, but are not exploited uniformly in the library as documented in CD 14882.	Fix the standard library so that “Concepts” are used appropriately in the library.	
US 62	17-30		ge	Provide concepts and requirements clauses for all standard library templates		
US 63	17-30		te	The behavior of the library in the presence of threads is incompletely specified. For example, if thread 1 assigns to X, then writes data to file f, which is read by thread 2, and then accesses variable X, is thread 2 guaranteed to be able to see the value assigned to X by thread 1? In other words, does the write of the data "happen before" the read? Another example: does simultaneous access using operator at() to different characters in the same non-const string really introduce a data race?		
DE-2	17 through 30		te	DE-2 Marking a constructor with "explicit" has semantics even for a constructor with zero or several parameters: Such a constructor cannot be used with list-initialization in a copy-initialization context, see 13.3.1.7. The standard library apparently has not been reviewed for marking non-single-parameter constructors as "explicit".	Consider marking zero-parameter and multi-parameter constructors "explicit" in classes that have at least one constructor marked "explicit" and that do not have an initializer-list constructor.	
JP 21	17 Library 21.2, 21.4, 27.2, 27.6, 27.7, 27.8.1,		te	Support of char16_t/char32_t is insufficient. The basic_xxx classes of <iostream>, <fstream>, <regex>, etc. does not have typedefs for char16_t/char32_t. Functions such as stoi, to_string in ‘21.4 Numeric Conversion’ does not support char16_t/char32_t types.	Add commented lines corresponding to char16_t, char32_t. 21.2 paragraph1	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

	28.4				namespace std {
					...
					
					// 21.4: numeric conversions
					...
					
					int stoi(const u16string& str, size_t *idx = 0, int base = 10);
					long stol(const u16string& str, size_t *idx = 0, int base = 10);
					unsigned long stoul(const u16string& str, size_t *idx = 0, int base = 10);
					long long stoll(const u16string& str, size_t *idx = 0, int base = 10);
					unsigned long long stoull(const u16string& str, size_t *idx = 0, int base = 10);
					float stof(const u16string& str, size_t *idx = 0);
					double stod(const u16string& str, size_t *idx = 0);
					long double stold(const u16string& str, size_t *idx = 0);
					u16string to_u16string(long long val);
					u16string to_u16string(unsigned long long val);
					u16string to_u16string(long double val);
					
					int stoi(const u32string& str, size_t *idx = 0, int base = 10);
					long stol(const u32string& str, size_t *idx = 0, int base = 10);
					unsigned long stoul(const u32string& str, size_t *idx = 0, int base = 10);
					long long stoll(const u32string& str, size_t *idx = 0, int base = 10);
					unsigned long long stoull(const u32string& str, size_t *idx = 0, int base = 10);
					float stof(const u32string& str, size_t *idx = 0);
					double stod(const u32string& str, size_t *idx = 0);
					long double stold(const u32string& str, size_t *idx	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					= 0);
					u32string to_u32string(long long val);
					u32string to_u32string(unsigned long long val);
					u32string to_u32string(long double val);
					}
					
					27.2
					
					namespace std {
					...
					typedef basic_ios<char> ios;
					typedef basic_ios<wchar_t> wios;
					typedef basic_ios<char16_t> u16ios;
					typedef basic_ios<char32_t> u32ios;
					
					...
					typedef basic_ifstream<wchar_t> wifstream;
					typedef basic_ofstream<wchar_t> wofstream;
					typedef basic_fstream<wchar_t> wfstream;
					
					typedef basic_streambuf<char16_t>
					u16streambuf;
					typedef basic_istream<char16_t> u16istream;
					typedef basic_ostream<char16_t> u16ostream;
					typedef basic_iostream<char16_t> u16iostream;
					
					typedef basic_stringbuf<char16_t>
					u16stringbuf;
					typedef basic_istringstream<char16_t>
					u16istringstream;
					typedef basic_ostringstream<char16_t>
					u16ostringstream;
					typedef basic_stringstream<char16_t>
					u16stringstream;
					typedef basic_filebuf<char16_t> u16filebuf;
					
					typedef basic_ifstream<char16_t> u16ifstream;	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					typedef basic_ofstream<char16_t> u16ofstream; typedef basic_fstream<char16_t> u16fstream; typedef basic_streambuf<char32_t> u32streambuf; typedef basic_istream<char32_t> u32istream; typedef basic_ostream<char32_t> u32ostream; typedef basic_iostream<char32_t> u32iostream; typedef basic_stringbuf<char32_t> u32stringbuf; typedef basic_istringstream<char32_t> u32istringstream; typedef basic_ostringstream<char32_t> u32ostringstream; typedef basic_stringstream<char32_t> u32stringstream; typedef basic_filebuf<char32_t> u32filebuf; typedef basic_ifstream<char32_t> u32ifstream; typedef basic_ofstream<char32_t> u32ofstream; typedef basic_fstream<char32_t> u32fstream; ... template <class state> class fpos; typedef fpos<char_traits<char>::state_type> streampos; typedef fpos<char_traits<wchar_t>::state_type> wstreampos; typedef fpos<char_traits<char16_t>::state_type> u16streampos; typedef fpos<char_traits<char32_t>::state_type> u32streampos; } 27.6	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					namespace std {
					template <class charT, class traits = char_traits<charT> >
					class basic_istream;
					typedef basic_istream<char> istream;
					typedef basic_istream<wchar_t> wistream;
					typedef basic_istream<char16_t> u16istream;
					typedef basic_istream<char32_t> u32istream;
					
					template <class charT, class traits = char_traits<charT> >
					class basic_iostream;
					typedef basic_iostream<char> iostream;
					typedef basic_iostream<wchar_t> wiostream;
					typedef basic_iostream<char16_t> u16iostream;
					typedef basic_iostream<char32_t> u32iostream;
					}
					
					namespace std {
					template <class charT, class traits = char_traits<charT> >
					class basic_ostream;
					typedef basic_ostream<char> ostream;
					typedef basic_ostream<wchar_t> wostream;
					typedef basic_ostream<char16_t> u16ostream;
					typedef basic_ostream<char32_t> u32ostream;
					}
					
					27.7 paragraph 1
					
					namespace std {
					template <class charT, class traits = char_traits<charT>,
					class Allocator = allocator<charT> >
					class basic_stringbuf;
					
					typedef basic_stringbuf<char> stringbuf;	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					typedef basic_stringbuf<wchar_t> wstringbuf; typedef basic_stringbuf<char16_t> u16stringbuf; typedef basic_stringbuf<char32_t> u32stringbuf; template <class charT, class traits = char_traits<charT>, class Allocator = allocator<charT> > class basic_istringstream; typedef basic_istringstream<char> istringstream; typedef basic_istringstream<wchar_t> wistringstream; typedef basic_istringstream<char16_t> u16istringstream; typedef basic_istringstream<char32_t> u32istringstream; template <class charT, class traits = char_traits<charT>, class Allocator = allocator<charT> > class basic_ostringstream; typedef basic_ostringstream<char> ostringstream; typedef basic_ostringstream<wchar_t> wostringstream; typedef basic_ostringstream<char16_t> u16ostringstream; typedef basic_ostringstream<char32_t> u32ostringstream; template <class charT, class traits = char_traits<charT>, class Allocator = allocator<charT> > class basic_stringstream;	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					typedef basic_stringstream<char> stringstream;
					typedef basic_stringstream<wchar_t> wstringstream;
					typedef basic_stringstream<char16_t> u16stringstream;
					typedef basic_stringstream<char32_t> u32stringstream;
					}
					
					27.8.1 paragraph 1
					
					namespace std {
					template <class charT, class traits = char_traits<charT> >
					class basic_filebuf;
					typedef basic_filebuf<char> filebuf;
					typedef basic_filebuf<wchar_t> wfilebuf;
					typedef basic_filebuf<char16_t> u16filebuf;
					typedef basic_filebuf<char32_t> u32filebuf;
					
					template <class charT, class traits = char_traits<charT> >
					class basic_ifstream;
					typedef basic_ifstream<char> ifstream;
					typedef basic_ifstream<wchar_t> wifstream;
					typedef basic_ifstream<char16_t> u16ifstream;
					typedef basic_ifstream<char32_t> u32ifstream;
					
					template <class charT, class traits = char_traits<charT> >
					class basic_ofstream;
					typedef basic_ofstream<char> ofstream;
					typedef basic_ofstream<wchar_t> wofstream;
					typedef basic_ofstream<char16_t> u16ofstream;
					typedef basic_ofstream<char32_t> u32ofstream;	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					template <class charT, class traits = char_traits<charT> > class basic_fstream;
					typedef basic_fstream<char> fstream;
					typedef basic_fstream<wchar_t> wfstream;
					typedef basic_fstream<char16_t> u16fstream;
					typedef basic_fstream<char32_t> u32fstream;
					}
					
					28.4
					
					namespace std {
					...
					typedef basic_regex<char> regex;
					typedef basic_regex<wchar_t> wregex;
					typedef basic_regex<char16_t> u16regex;
					typedef basic_regex<char32_t> u32regex;
					
					...
					typedef sub_match<const char*> csub_match;
					typedef sub_match<const wchar_t*> wcsub_match;
					typedef sub_match<const char16_t*> u16csub_match;
					typedef sub_match<const char32_t*> u16csub_match;
					typedef sub_match<string::const_iterator> ssub_match;
					typedef sub_match<wstring::const_iterator> wssub_match;
					typedef sub_match<u16string::const_iterator> u16ssub_match;
					typedef sub_match<u32string::const_iterator> u32ssub_match;
					
					...	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					typedef match_results<const char*> cmatch; typedef match_results<const wchar_t*> wcmatch; typedef match_results<const char16_t*> u16cmatch; typedef match_results<const char32_t*> u32cmatch; typedef match_results<string::const_iterator> smatch; typedef match_results<wstring::const_iterator> wsmatch; typedef match_results<u16string::const_iterator> u16smatch; typedef match_results<u32string::const_iterator> u32smatch; ... typedef regex_iterator<const char*> cregex_iterator; typedef regex_iterator<const wchar_t*> wcregex_iterator; typedef regex_iterator<const cha16r_t*> u16cregex_iterator; typedef regex_iterator<const char32_t*> u32cregex_iterator; typedef regex_iterator<string::const_iterator> sregex_iterator; typedef regex_iterator<wstring::const_iterator> wsregex_iterator; typedef regex_iterator<u16string::const_iterator> u16sregex_iterator; typedef regex_iterator<u32string::const_iterator> u32sregex_iterator; ... typedef regex_token_iterator<const char*>	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					cregex_token_iterator; typedef regex_token_iterator<const wchar_t*> wcregex_token_iterator; typedef regex_token_iterator<const char16_t*> u16cregex_token_iterator; typedef regex_token_iterator<const char32_t*> u32cregex_token_iterator; typedef regex_token_iterator<string::const_iterator> sregex_token_iterator; typedef regex_token_iterator<wstring::const_iterator> wsregex_token_iterator; typedef regex_token_iterator<u16string::const_iterator> u16sregex_token_iterator; typedef regex_token_iterator<u32string::const_iterator> u32sregex_token_iterator; }	
UK 144	17.1	2	Ed	List of contents of library should be extened to cover new clauses	Add "regular expressions, atomic operations and threads"	
UK 145	17.1	6	Ed	Summary of numeric facilities should mention random numbers	Add random number framework to the list of library facilities	
UK 146	17.1		Ed	Add a summary paragraph for regular expressions	Add a summary paragraph for regular expressions	
UK 147	17.1		Ed	Add a summary paragraph for threads	Add a summary paragraph for threads	
UK 148	17.2	Table 12	Ed	Table 12 is mentioned in and relates to section 17.2, but has been pushed down to appear directly after the title of section 17.3 which is rather unfortunate/confusing for the reader.	Make sure tables are rendered in the section to which they relate.	
UK 149	17.3		Ed	For consistency with narrow-oriented and wide-oriented streams, we should add terms for streams of Unicode	Define Utf16-oriented stream classes and Uft32-oriented stream classes for streams of	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				character sequences	char16_t/char32_t values.	
UK 150	17.3		Ed	The addition of move semantics to the language means that many library APIs leave an object in a safely-destructible state, where no other operations can safely be performed unless it is assigned a new value. Library presentation would be simplified and made more precise if we introduce a term for this state. By analogy with singular iterators suggest the term 'singular object' or 'the object is in a singular state'.	Define 'singular state' such that an object with a singular state can only be assigned to or safely destroyed. Assiging a new value typically removes the singular state. Note that objects with a singular state may not be safely copied, so you cannot put an object into a singular state by copying another object in a singular state. Use this new term in the postcondition of all library APIs that move from an rvalue reference. It might also be used to simplify the definition of singular iterator to an iterator value with a singular state.	
UK 151	17.3.1		Ed	Missing crossreference to 17.3.17 [defns.repositional.stream]	Add cross-reference in the existing empty brackets	
UK 152	17.3.12		Te	Object state is using a definition of object (instance of a class) from outside the standard, rather than the 'region of storage' definiton in 1.8p1	Clarify terms and usage	
UK 153	17.3.17		Te	If a repositional stream can only seek to a position previously encountered, then an arbitrary-positional-stream cannot satisfy this definition, as cross-referenced in 17.3.17	Strike the word 'only'. A note might be added to reinforce the intent	
UK 154	17.3.20		Ed	Missing definition of a stable partition algorithm	Add definition from 25.2.12p7	
UK 155	17.3.3		Ed	Add clause 28 to list that use this definition of character	Add clause 28 to list that use this definition of character	
UK 156	17.3.4		Ed	Add regular expressions to set of templates using character container type	Add regular expressions to set of templates using character container type	
UK 157	17.5.2.2	3	Ed	Add concepts to the ordered list of presentation	Add concepts into the sequence	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 158	17.5.2.2	3	Ed	templates are neither classes nor functions	Replace 'classes' and 'functions' with 'classes and class templates' and 'functions and function templates'	
UK 159	17.5.2.4	Footnote 152	Ed	This informative footnote was relevant in 1998, not 2008. The term 'existing vendors' may imply something different now	Strike the footnote, or replace 'existing' with 'original' or similar	
UK 160	17.5.2.4	3	Ed	requires is now a keyword with a specific meaning related to concepts, and its use in library specifcation may be confusing. Generally the Requires clause is used to make requirements on the caller, not the library, so typically providing runtime pre-conditions. Suggest a new name to refflect that. Note that Clause 30 already seems to be written to this convention.	Replace 'Requires' with 'Preconditions'	
UK 161	17.5.2.4	4	Ed	This paragraph is redundant as the definition of the term 'handler function' is already provided in 17.3. Are we in danger of proving two definitions of the same terms? Which is the 'controlling' definition?	Strike 17.5.2.4p4	
UK 162	17.5.2.4	3	Ed	Clause 30 makes use of a 'Synchronization' semantic element, that frequently appears either between Effects: and Postconditions:, or between Returns: and Throws:	Add 'Synchronization' to the list either between Effects: and Postconditions:, or between Returns: and Throws:.	
UK 163	17.5.2.4	3	Te	Many functions are defined as "Effects: Equivalent to a...", which seems to also define the preconditions, effects, etc. But this is not made clear.	Introduce an explicit "Equivalent to", which defines all of the properties of the function.	
UK 164	17.5.3.2.1	1	Ed	This phrasing predates concepts. While this kind of description is still used, the examples provided are now all concepts, and should be replaced with appropriate examples	Use better names for the examples. Ideally totally replace the need by constraining all templates in library, so that real concepts are all that is needed. Note if retained that CopyConstructible is mis-spelled.	
UK 165	17.5.3.2.2, 17.5.3.2.3		Te	constraints on bitmask and enumation types were supposed to be tightened up as part of the motivation for	Adopt wording in line with the motivation described in N2235	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				the constexpr feature - see paper n2235 for deails		
UK 166	17.5.3.2.4.1, 17.5.3.3		Ed	List of library clauses should go up to 30, not 27	Replace initial refernce to ch27 with ch30	
UK 167	17.5.3.4 Private members		Ed	Comment marker in wrong place.	Change: // streambuf* sb; exposition only to streambuf* sb; // exposition only To reflect actual usage.	
UK 168	17.6.2.2	2	Te	We should make it clear (either by note or normatively) that namespace std may contain inline namespaces, and that entities specified to be defined in std may in fact be defined in one of these inline namespaces. (If we're going to use them for versioning, eg when TR2 comes along, we're going to need that.)	Replace "namespace std or namespaces nested within namespace std" with "namespace std or namespaces nested within namespace std or inline namespaces nested directly or indirectly within namespace std"	
UK 169	17.6.2.2		Te	This phrasing contradicts later freedom to implement the C standard library portions in the global namespace as well as std. (17.6.2.3p4)	Resolve conflict in either place	
UK 170	17.6.2.3		Te	One of goals of C++0x is to make language easier to teach and for 'incidental' programmers. The fine-grained headers of the C++ library are valuable in large scale systems for managing dependencies and optimising build times, but overcomplicated for simple development and tutorials. Add additional headers to support the whole library through a single include statement.	Add a new header <std> that has the effect of including everything in tables 13 and 14, except <iosfwd> and <cassert>. Add an additional header <fwd> that adds all declarations from <std> but no definitions.	
UK 171	17.6.2.4	3	Ed	Does freestanding implementation require a full implementation of all listed headers? The reference to abort, at_exit and exit is confusing. Is a conforming implementation allow to deliver partial forms of these headers? If so which ones? Are empty versions of everything but <cstdlib> conforming?	Either strike the references to abort, at_exit and exit, or clarify which headers only require partial support.	
UK 172	17.6.2.4	3	Te	No reference to new functions quick_exit and at_quick_exit	Add reference to quick_exit and at_quick_exit	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 173	17.6.2.4	table 15	Te	<initializer_list> is missing from headers required in freestanding implementations.	Add 18.8, initializer lists, <initializer_list>, to the end of the table.	
JP 23	17.6.2.4	2nd paragraph, Table 15	te	There is a freestanding implementation including <type_traits>, <array>, <ratio>, lately added to Table 13, C++ library headers. Programmers think them useful and hope that these headers are also added to Table 15, C++ headers for freestanding implementations, that shows the set of headers which a freestanding implementation shall include at least.	Add <type_traits>, <array>, <ration> to Table 15.	
UK 174	17.6.3.2	3	Ed	The phrasing is mildly ambiguous when using the word 'it' to refer back to the header - an unfotunate reading might confuse it with the translate unit, which is the subject of the surrounding clause.	Replace 'the first reference to any of the entities declared in that header by the translation unit' with 'the first reference to any of the entities that header declares in the translation unit'	
UK 175	17.6.4.2.1	2	Te	Local types can now be used to instantiate templates, but don't have external linkage	Remove the reference to external linkage	
UK 176	17.6.4.3.3	Footnote 175	Ed	Reference to namespace ::std should be 17.6.4.2	Change referfence from 17.6.4.3 to 17.6.4.2	
UK 177	17.6.4.3.4	3	Ed	Sentence is redundant as double underscores are reserved in all contexts by 17.6.4.3.3	Strike the sentence	
UK 178	17.6.4.8	2	Ed	The last sentence of the third bullet "Operations on such types can report a failure by throwing an exception unless otherwise specified" is redundant as behaviour is already undefined.	Strike the sentence	
UK 179	17.6.4.8	2	Te	According to the 4th bullet there is a problem if "if any replacement function or handler function or destructor operation throws an exception". There should be no problem throwing exceptions so long as they are caught within the function.	Replace the word 'throws' with 'propogates'	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
JP 22	17.6.5.7	4th paragraph, 1st line	ed	The statement below describes relation among two or more threads using words “between threads”: [Note: This means, for example, that implementations can’t use a static object for internal purposes without synchronization because it could cause a data race even in programs that do not explicitly share objects between threads. —end note] In such case, “among” is preferred instead of “between”.	Change "between threads" to "among threads". There are same cases in 17.6.1 paragraph 2, 17.6.5.7 paragraph 6, 30.1 paragraph 1, 30.3.1 paragraph 1 also.	
UK 180	17.6.5.10	1, 4	Te	It should not be possible to strengthen the exception specification for virtual functions as this could break user code. Note this is not a problem in practice as there are no virtual functions with exception specifications in the current library, other than empty throw specifications which it is not possible to strengthen.	Add restriction that exception specification of virtual functions cannot be tightened.	
UK 181	17.6.5.10	Footnote 186	Te	This footnote is wrong. C library functions do not have any exception specification, but might be treated as if they had an empty throw specification	Clarify that this note does not mean the functions are genuinely declared with the specification, but are treated as-if.	
UK 182	17.6.5.10	Footnote 188	Te	It is very helpful to assume all exceptions thrown by the standard library derive from std::exception. The 'encouragement' of this note should be made normative.	Make this footnote normative	
UK 184	18 -> 30		Ed	The new alias-declaration syntax is generally easier to read than a typedef declaration. This is especially true for complex types like function pointers.	Replace all typedef declarations in the standard library with alias-declarations, except in the standard C library.	
JP 24	18	2nd paragraph, Table 16	ed	Subclauses are listed in Table 16 as: "18.6 Type identification …" "18.8 Initializer lists …" "18.7 Exception handling …".	Sort them in the increasing order "18.6 Type identification …" "18.7 Exception handling …". "18.8 Initializer lists …"	
JP 25	18.1	6th paragraph ,	ed	max_align_t is described in 18.1, so add 3.11 Alignment as the reference.	Add "3.11, Alignment" to SEE ALSO.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
		last line, SEE ALSO				
FR 32	18.2.1 [Numeric limits]		te	The definition of numeric_limits<> as requiring a regular type is both conceptually wrong and operationally illogical. As we pointed before, this mistake needs to be corrected. For example, the template can be left unconstrained. In fact this reflects a much more general problem with concept_maps/axioms and their interpretations. It appears that the current text heavily leans toward experimental academic type theory.	We suggest that a more pragmatic approach, in the spirit of C and C++, be taken so that calls to constrained function templates are interpreted as assertions on *values*, not necessarily semantics assertions on the carrier type.	
DE-16	18.2.1		te	DE-16 The class template numeric_limits should not specify the Regular concept requirement for its template parameter, because it contains functions returning NaN values for floating-point types; these values violate the semantics of EqualityComparable. See also library issue 902 in WG21 document N2794 "C++ Standard Library Active Issues List (Revision R60)", available at http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2794.html .	Specify a concept requirement with fewer constraints as appropriate, for example SemiRegular.	
JP 26	18.2.1.1		te	numeric_limits does not use concept.	Correct as follows. template<class T> class numeric_limits<const T>; template<class T> class numeric_limits<volatile T>; template<class T> class numeric_limits<const volatile T>; should be template<Regular T> class numeric_limits<const T>; template<Regular T> class numeric_limits<volatile T>;	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					template<Regular T> class numeric_limits<const volatile T>;	
DE-17	18.2.6		te	DE-17 The class type_index should be removed; it provides no additional functionality beyond providing appropriate concept maps.	Specify concept maps for "const type_info *" as required by the ordered and unordered containers and remove the class type_index.	
UK 185	18.3.1	2	Ed	There is no header <stdint>, it should either be <stdint.h> or <cstdint>	Replace <stdint> with <cstdint>	
DE-18	18.4		te	DE-18 The proposed C++ standard makes a considerable number of existing programs that have well-defined behavior according to ISO/IEC 14882:2003 have undefined behavior without a diagnostic hint to the programmer at all. This applies to the presence of threads and to pointer safety (the latter was introduced to support garbage collection). In order to avoid requiring a full code review for user code, facilities should be present that allow the compile-time detection of the advanced features of the proposed C++ standard. It is expected that C++ implementations will provide a means (for example, a command-line switch) to turn off either or both of threads and garbage collection support, turning potentially undefined programs into well-defined ones. *Note:* This issue is contributing significantly to Germany's overall "no" vote.	Consider applying the changes proposed in WG21 document N2693 "Requirements on programs and backwards compatibility", available at http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2693.html .	
UK 186	18.4	Footnote 221	Ed	What is the purpose of this comment? The standard stream objects (cin, cerr etc.) have a peculiar lifetime that extends beyond the program. They may never be destroyed so will not be responsible for flushing buffers at the stated time.	Remove the footnote	
UK 187	18.4	9	Te	The term "thread safe" is not defined nor used in this context anywhere else in the standard.	Clarify the intended meaing of "thread safe"	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 188	18.4	12	Te	The function _Exit does not appear to be defined in this standard. Should it be added to the table of functions included-by-reference to the C standard?	Depends on where _Exit comes from	
UK 189	18.4, 18.7		Te	The addition of the [[noreturn]] attribute to the language will be an important aid for static analysis tools.	The following functions should be declared in C++ with the [[noreturn]] attribute: abort exit quick_exit terminate unexpected rethrow_exception throw_with_nested	
JP 27	18.4, 18.9, 18.7.2.2, 18.7.3.1		te	There are Standard library functions that never return to the caller. They are explained so in the Standard but not declared explicitly.	Consider to add the attribute [[noreturn]] to such functions, 15.5.2 unexpected 18.4: abort(), exit(), quick_exit, 18.7.2.2: unexpected_handler, 18.7.3.1: terminate_handler, 18.7.6 rethrow_nested 18.7.6 throw_with_nested 18.9: longjmp.	
UK 190	18.5.1	various	Te	It is not entirely clear how the current specification acts in the presence of a garbage collected implementation.	All deallocation functions taking a pointer parameter should have a Precondition : ptr is a safely-derived pointer value.	
UK 191	18.5.1.1	4	Ed	According to the second bullet, behaviour becomes undefined (for lack of a specification) if the user has not yet called set_new_handler.	Rephrase the second bullet in terms of a new handler being installed, and update any definition of handler function necessary to be sure the term 'installed' is defined.	
UK 192	18.5.1.2		Te	The declared signature is not compatible with the current requirement to throw std::length_error. It is too late to weaken the exception specification, so the only other change is to preserve new (improved) behaviour is to throw std::bad_alloc, or something derived from std::bad_alloc.	Fix 5.3.4p7 by required std::bad_alloc rather than std::length_error	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 193	18.5.2.2	2	Te	quick_exit has been added as a new valid way to terminate a program in a well defined way	Change 3rd bullet: call either abort(), exit() or quick_exit();	
UK 194	18.6		Te	The inclusion of type_index and hash<type_index> in <typeinfo> brings dependencies into <typeinfo> which are inconsistent with the definition of freestanding C++ in 17.6.2.4.	Move type_index and hash<type_index> out of <typeinfo> and into a new header, <typeindex>.	
JP 28	18.6, 18.7, 19.1		te	Errors reported by Exception classes are of types char or std::string only. For example, std::exception is declared with char, std::string types, therefore types wchar_t/wstring, char16_t/u16string, or char32_t/u32string can not be used.	Consider other types.	

JP 29	18.7.6		te	throw_with_nested does not use concept.	Correct as follows.
					template<class T> void throw_with_nested(T&& t); // [[noreturn]]
					
					should be
					
					template<CopyConstructible T> void throw_with_nested(T&& t); // [[noreturn]]	

JP 30	18.7.6		te	To handle nested exceptions strictly, error information of tree structure will be required though, the nested_exception does not support tree structure. It is insufficient as error handling.	Consider nested_exception to support tree structure.	
JP 31	18.7.6		te	It is difficult to understand in which case nested_exception is applied.	Consider to add a sample program which rethrows exception.	
UK 195	18.8		Te	The class definition of std::initializer_list contains concept-maps to Range which should be out of the class, and in <iterator_concepts> instead. Otherwise, it's not possible to use initializer_lists in a freestanding C++	Delete the two concept maps from std::initializer_list.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				implementation.		
UK 196	18.8.3		Te	Concept maps for initializer_list to Range should not be in language support headers, but instead in iterator concepts.	Remove section 18.8.3 and put it in 24.1.8.1 instead, so that the concept_maps from initializer_list to Range are specified under Range instead of under initializer lists; also, so that they're implemented in <iterator_concepts> instead of <initializer_list>.	
UK 197	19		Te	All the exception classes in this clause take a std::string argument by const reference. They should all be overloaded to accept std::string by rvalue rerefence for an efficient move as well.	Provide a constructor for every exception class in clause 19 accepting a std::string by rvalue reference, with the semantics that the passed string may be moved.	
JP 32	19.1		te	Messages returned by the member function what() of standard exception classes seem difficult to judge. For example, following messages are returned by what() of std::bad_alloc of existing implementations: Compiler: Message returned by what() --- Borland C++ 5.6.4: no named exception thrown Visual C++ 8.0: bad allocation Code Warrior 8.0: exception g++ 3.4.4: St9exception It is difficult to recognize what exception was thrown when using those compilers except Visual C++.	Consider to add footnote that recommends what() returns message easy to recognize what exception was thrown.	
US 64	19.3	1	Ge	" See also: ISO C 7.1.4, 7.2, Amendment 1 4.3." It is unclear why this cross reference is here. Amendment 1 was to C89, not C99.	Delete this cross reference. If necessary, expand the main text to include the relevant referenced text	
US 65	20		te	Scoped allocators and allocator propagation traits add a small amount of utility at the cost of a great deal of	Sketch of proposed resolution: Eliminate scoped allocators, replace allocator propagation traits with	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				machinery. The machinery is user visible, and it extends to library components that don't have any obvious connection to allocators, including basic concepts and simple components like pair and tuple.	a simple uniform rule (e.g. always propagate on copy and move), remove all mention of allocators from components that don't explicitly allocate memory (e.g. pair), and adjust container interfaces to reflect this simplification. Components that I propose eliminating include HasAllocatorType?, is_scoped_allocator, allocator_propagation_map, scoped_allocator_adaptor, and ConstructibleAsElement?.	
UK 198	20		Ed	The organization of clause 20 could be improved to better group related items, making the standard easier to navigate.	20.6.7, 20.6.8, 20.6.9 and 20.6.10 should be grouped under a section called "operator wrappers" or similar. The goal of all 4 subsections combined is to provide a functor for every operator in the language. 20.6.17 class template hash should numerically appear immediately after the operator wrappers, as they are functors that are used in similar ways 20.6.11, 20.6.12, 20.6.13, 20.6.14, 20.6.15 are strongly related to 20.6.3, and to an extent 20.6.2. These should all come under a subheading of "function adapters" 20.7.1, 20.7.3, 20.7.4, 20.7.5, 20.7.6, 20.7.7 and 20.7.10 should all be grouped as subclauses under [20.7.2 Allocators] [20.7.12 unique_ptr] should be a sub-section of [20.7.13 smart pointers] [20.7.13 smart pointers] is important enough to have a high level bullet after [20.7 memory], suggest renumbering as [20.8 smart pointers] [20.7.13.7 Pointer safety] is nothing to do with smart pointers and should become its own subclause [20.7.14 Pointer safety] [20.9 date and time functions] should be moved under [20.8 time utilities] and retitled [20.8.6 C Library] (as per memory management/C Library) [20.6.18 reference_closure] is fundamentally a language support feature and should move to clause 18, see separate comment [20.6.5	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					reference_wrapper] should be simplified and moved into [2.2 utility components], see separate comment [20.6.4 result_of] should be reorganised as a type trait - see separate comment Tuples and pairs are closely related so merge tuple and pair into the same subclause - see more general comment on this	
UK 199	20.1.1, 20.1.2	2	Te	The requirement that programs do not supply concept_maps should probably be users do not supply their own concept_map specializations. THe program will almost certainly supply concept_maps - the standard itself supplies a specialization for RvalueOf? references. Note that the term _program_ is defined in 3.5p1 and makes no account of the standard library being treated differently to user written code.	Replace the term 'program' with 'user'.	
UK 200	20.1.4		Te	All standard library use expects Predicates to be CopyConstructible, and this should be recognised easily without reatedly stating on every use-case.	Either require CopyConstructible<F> as part of Predicate, or create a refined concept, StdPredicate, used throughout the library that requires CopyConstructible as well as Callable. Consider making (Std)Predicate SemiRegular.	
UK 201	20.1.5		Te	The Consistency axiom for LessThanComparable will not compile.	Add a requires clause to the Consistency axiomL requires HasLessEquals<T> && HasGreaterEquals<T>, or split the Consistency axiom into two so that 'basic' consistency can be asserted regardless of the <=/>= requirement.	
JP 33	20.1.5		te	LessThanComparable and EqualityComparable don't correspond to NaN.	Apply concept_map to these concepts at FloatingPointType	
US 66	20.1.10		te	Application of the "Regular" concept to floating-point types appears to be controversial (see long discussion on std-lib reflector).	State that the “Regular” concept does not apply to floating-point types.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
JP 34	20.2	1st paragraph, 4th line	ed	Though N2672 pointed at adding "#include<initializer_list>", it isn't reflected.	add followings #include <initializer_list> // for concept_map	
US 67	20.2.1	¶ 5 first sent.	ed	Some connective words are missing.	Insert “corresponding to” before “an lvalue reference type.”	
JP 35	20.2.3	6th paragraph, 1st line	ed	Typo, "stdforward" should be "std::forward"	Correct typo.	
UK 202	20.2.4		Ed	The references to pair in the tuple-like access to pair functions qualify pair with std::pair even though they are in a namespace std block.	Remove the std:: qualification from these references to pair.	
US 68	20.2.12	IntegralLike	te/ed	The code defining the context is syntactically incorrect.	Insert a comma in two places: at the end of the third line of refinements, and at the end of the fourth line of refinements.	
UK 203	20.3.2	1-4	Ed	The ratio_xyz types have a misplaced '}'. For example: template <class R1, class R2> struct ratio_add { typedef see below} type; ;	Move the '}' to after the typedef: template <class R1, class R2> struct ratio_add { typedef see below type; };	
JP 36	20.4.2.1	19th paragraph, 6th line	ed	Typo. "it it" should be "it is"	Correct typo.	
UK 204	20.5	Table 41	Te	It is not possible to create a variant union based on a parameter pack expansion, e.g. to implement a classic discriminated union template.	Restore aligned_union template that was removed by LWG issue 856.	
US 69	20.5		ed	This section, dealing with tuple<>, should be in the same section as the similar utility pair<>.	Restructure Clause 20 so as to bring these similar components together.	
UK 205	20.5.3		Te	integral_constant objects should be usable in integral-constant-expressions. The addition to the language of literal types and the enhanced rules for constant expressions make this possible.	Add a constexpr conversion operator to class tempalate integral_constant: constexpr operator value_type() { return value; }	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 206	20.5.5	para 4	Te	Currently the std says: "In order to instantiate the template is_convertible<From, To>, the following code shall be well formed:" But the code shown is the requirement for the result of is_convertible to be a true_type, not a precondition on whether the template can be instantiated.	Change: "In order to instantiate the template is_convertible<From, To>, the following code shall be well formed:" To: "The template is_convertible<From, To> inherits either directly or indirectly from true_type if the following code is well formed:"	
UK 207	20.5.6.1	Table 36	Ed	suffix "::type" is missing from the some of the examples.	Change: Example:remove_const<const volatile int>::type evaluates to volatile int, whereas remove_const<const int*> is const int*. —end example To: Example:remove_const<const volatile int>::type evaluates to volatile int, whereas remove_const<const int*>::type is const int*. —end example And change: Example:remove_volatile<const volatile int>::type evaluates to const int, whereas remove_volatile<volatile int*> is volatile int*. —end example To: Example:remove_volatile<const volatile int>::type evaluates to const int, whereas remove_volatile<volatile int*>::type is volatile int*. —end example And change: Example:remove_cv<const volatile int>::type evaluates to int, whereas remove_cv<const volatile int*> is const volatile int*. —end example To: Example:remove_cv<const volatile int>::type evaluates to int, whereas remove_cv<const volatile int*>::type is const volatile int*. —end example	
JP 37	20.5.7	Table 41	ed	Typo. There isn't a period at the end of enable_if's comments. If B is true, the member typedef type shall equal T; otherwise, there shall be no member typedef type	Add ”.”	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				should be If B is true, the member typedef type shall equal T; otherwise, there shall be no member typedef type.		
US 70	20.6		te	Specifications now expressed via narrative text are more accurately and clearly expressed via executable code.	Wherever concepts are available that directly match this section's type traits, express the traits in terms of the concepts instead of via narrative text. Where the type traits do not quite match the corresponding concepts, bring the two into alignment so as to avoid two nearly-identical notions.	
US 71	20.6.7	Table 51, last row, column 3	ed	The grammar is incorrect.	Change "conversion are" to "conversion is".	

JP 38	20.6.12.1.3		te	add the move requirement for bind's return type.
				
				For example, assume following th1 and th2,
				
				void f(vector<int> v) { }
				
				vector<int> v{ ... };
				thread th1([v]{ f(v); });
				thread th2(bind(f, v));
				
				When function object are set to thread, v is moved to th1's lambda expression in a Move Constructor of lambda expression becuase th1's lambda expression has a Move Constructor. But bind of th2's return type doesn't have the requirement of Move, so it may not moved but copied. Add the requirement of move to get rid of this useless copy.
				And also, add the MoveConstructible as well as CopyConstructible.
					Add the following requirements.
					"it has a public move constructor that performs a member-wise move."
					Add the MoveConstructible.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

JP 39	20.6.16.2		te	There are no requires corresponding to F of std::function.	Correct as follows.
					template<class F, Allocator A>
					function(allocator_arg_t, const A&, F);
					template<class F, Allocator A>
					function(allocator_arg_t, const A&, F&&);
					
					should be
					
					template<class F, Allocator A>
					requires CopyConstructible<F> && Callable<F, ArgTypes...>
					&& Convertible<Callable<F, ArgTypes...>::result_type, R>
					function(allocator_arg_t, const A&, F);
					template<class F, Allocator A>
					requires CopyConstructible<F> && Callable<F, ArgTypes...>
					&& Convertible<Callable<F, ArgTypes...>::result_type, R>
					function(allocator_arg_t, const A&, F&&);	

JP 40	20.6.16.2		ed	Thougn it's "Allocator Aloc" at other places, it's "Allocator A" only std::function constructor template parameter.	Correct as follows.
					template<class F, Allocator A>
					function(allocator_arg_t, const A&, F);
					template<class F, Allocator A>
					function(allocator_arg_t, const A&, F&&);
					
					should be
					
					template<class F, Allocator Alloc>
					function(allocator_arg_t, const Alloc &, F);
					template<class F, Allocator Alloc>
					function(allocator_arg_t, const Alloc &, F&&);	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

JP 41	20.6.16.2		te	There are no requires corresponding to R and Args of UsesAllocator.	Correct as follows.
					template <class R, class... Args>
					concept_map UsesAllocator<function<R(Args...)>, Alloc> {
					typedef Alloc allocator_type;
					}
					
					should be
					
					template <Returnable R, CopyConstructible... Args>
					concept_map UsesAllocator<function<R(Args...)>, Alloc> {
					typedef Alloc allocator_type;
					}	

JP 42	20.6.16.2		ed	The requires are wrong.
				
				R require Returnable, and ArgTypes requires CopyConstructible by a definition of function, then it's a mistake to designate followings by MoveConstructible.
					Correct as follows.
					
					template <MoveConstructible R, MoveConstructible... ArgTypes>
					bool operator==(const function<R(ArgTypes...)>&, nullptr_t);
					template <MoveConstructible R, MoveConstructible... ArgTypes>
					bool operator==(nullptr_t, const function<R(ArgTypes...)>&);
					template <MoveConstructible R, MoveConstructible... ArgTypes>
					bool operator!=(const function<R(ArgTypes...)>&, nullptr_t);
					template <MoveConstructible R, MoveConstructible... ArgTypes>
					bool operator!=(nullptr_t, const function<R(ArgTypes...)>&);
					template <MoveConstructible R, MoveConstructible... ArgTypes>
					void swap(function<R(ArgTypes...)>&, function<R(ArgTypes...)>&);	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					should be
					
					template <Returnable R, CopyConstructible... ArgTypes>
					bool operator==(const function<R(ArgTypes...)>&, nullptr_t);
					template <Returnable R, CopyConstructible... ArgTypes>
					bool operator==(nullptr_t, const function<R(ArgTypes...)>&);
					template <Returnable R, CopyConstructible... ArgTypes>
					bool operator!=(const function<R(ArgTypes...)>&, nullptr_t);
					template <Returnable R, CopyConstructible... ArgTypes>
					bool operator!=(nullptr_t, const function<R(ArgTypes...)>&);
					template <Returnable R, CopyConstructible... ArgTypes>
					void swap(function<R(ArgTypes...)>&, function<R(ArgTypes...)>&);	

UK 208	20.6.17	1	Te	std::hash should be implemented for much more of the standard library. In particular for pair, tuple and all the standard containers.	.	
UK 209	20.7		Te	Smart pointers cannot be used in constrained templates	Provide constraints for smart pointers	
UK 213	20.7.6		Te	std::allocator should be constrained to simplify its use on constrained contexts. This library component models allocation from free store via the new operator so choose constraints to match. The Allocator concept allows for a wider variety of allocators that users may choose to	The primary allocator template should be constrained to require ObjectType<T> and FreeStoreAllocatable<T>. Further operations to be constrained as required.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				supply if their allocation model does not require operator new, without impacting the requirements of this template.		
UK 214	20.7.8		Te	raw_storage_iterator needs constraining as an iterator adaptor to be safely used in constrained templates	Constrain the raw_storage_iterator template	
UK 210	20.7.11		Te	Specialized algorithms for memory managenment need requirements to be easily usable in constrained templates	Provide constraints for all algorithms in 20.7.11	
DE-20	20.7.12		ed	DE-20 The section heading and the first sentence use the term "template function", which is undefined.	Replace "template function" by "function template".	
US 72	20.7.12		te	bind should support move-only functors and bound arguments.		
DE-21	20.7.12.1.3		te	DE-21 The specification for bind claims twice that "the values and types for the bound arguments v1, v2, ..., vN are determined as specified below". No such specification appears to exist.	Add the missing specification in the same section, or add a cross-reference indicating the section where the specification appears.	
UK 211	20.7.12.2.3	11	Te	The nullptr_t type was introduced to resolve the null pointer literal problem. It should be used for the assignemnt operator, as with the constructor and elsewhere through the library.	Change signature here and in the synopsis to: unique_ptr& operator=(nullptr_t); Strike the sentance an note before the Effects clause.	
UK 212	20.7.13.7		Te	The pointer-safety API is nothing to do with smart pointers, so does not belong in 20.7.13. In fact it is a set of language support features are really belongs in clause 18, with the contents declared in a header that deals with language-support of memory management.	Move this specification to 18.5. Move the declarations into the header <new>.	
DE-22	20.7.16.2		te	DE-22 The conditions for deriving from std::unary_function and std::binary_function are unclear: The condition would also be satisfied if ArgTypes were std::vector<T1>, because it (arguably) "contains" T1.	Consider stating the conditions in normative prose instead of in comments in the class definition. Use "consists of" instead of "contains". Consider using "if and only if" instead of "iff".	
US	20.7.18		te	The std::reference_closure template is redundant with	Remove 20.7.18 [func.referenceclosure].	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
73				std::function and should be removed. std::reference_closure is a premature optimization that provides a limited subset of the functionality of std::function intended to improve performance in a narrow use case. However, the "parallel application performance" benchmark used to motivate the inclusion of std::reference_closure was flawed in several ways: (3) it failed to enable a common optimization in std::function (implemented by all vendors), exacting a large and unrealistic penalty for copying std::function instances, and (4) it failed to account for parallel scheduler overhead or realistically-sized work units, both of which would dominate the costs measured by the benchmark in any realistic application.	Remove 5.1.1 paragraph 12.	
US 74	20.8		te	Scoped allocators represent a poor trade-off for standardization, since (1) scoped-allocator--aware containers can be implemented outside the C++ standard library but used with its algorithms, (2) scoped allocators only benefit a tiny proportion of the C++ community (since few C++ programmers even use today's allocators), and (3) all C++ users, especially the vast majority of the C++ community that won't ever use scoped allocators are forced to cope with the interface complexity introduced by scoped allocators. In essence, the larger community will suffer to support a very small subset of the community who can already implement their own data structures outside of the standard library. Therefore, scoped allocators should be removed from the working paper. Some evidence of the complexity introduced by scoped allocators: 20.3.3, 20.5: large increase in the number of pair and tuple constructors 23: confusing "AllocatableElement" requirements	Remove support for scoped allocators from the working paper. This includes at least the following changes: Remove 20.8.3 [allocator.element.concepts] Remove 20.8.7 [allocator.adaptor] Remove 20.8.10 [construct.element] In Clause 23: replace requirements naming the AllocatableElement concept with requirements naming CopyConstructible, MoveConstructible, DefaultConstructible, or Constructible, as appropriate.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				throughout.		
US 74	20.8.2.2	(a) synopsis (b) after ¶ 14	te/ed	A concept name is twice misspelled.	Change “Hasconstructor” to “HasConstructor” (twice).	
US 75	20.8.2.2		te	Allocator concepts are incomplete	See paper: http://www.halpernwightsoftware.com/WG21/n2810_allocator_defects.pdf	

JP 43	20.8.3		ed	Typo.
				"alloc" should be "Alloc"
					Correct as follows.
					
					auto concept UsesAllocator<typename T, typename Alloc> {
					requires Allocator<alloc>;
					typename allocator_type = T::allocator_type;
					
					should be
					
					auto concept UsesAllocator<typename T, typename Alloc> {
					requires Allocator<Alloc>;
					typename allocator_type = T::allocator_type;	

UK 215	20.8.3.3	6,8	Ed	Extra pair of {}, presumably some formatting code gone awry : duration& operator-{-}();	Remove the {} or fix formatting	
US 77	20.8.4		te	Allocator-specific move and copy behavior for containers (N2525) complicates a little-used and already-complicated portion of the standard library (allocators), and breaks the conceptual model of move-constructor and move-assignment operations on standard containers being efficient operations. The extensions for allocator-specific move and copy behavior should be removed from the working paper. With the introduction of rvalue references, we are	Remove 20.8.4. Remove 20.8.5. Remove all references to the facilities in 20.8.4 and 20.8.5 from clause 23.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				teaching programmers that moving from a standard container (e.g., a vector<string>) is an efficient, constant-time operation. The introduction of N2525 removed that guarantee; depending on the behavior of four different traits (20.8.4), the complexity of copy and move operations can be constant or linear time. This level of customization greatly increases the complexity of standard containers, and benefits only a tiny fraction of the C++ community.		
US 78	20.8.12, 20.8.13.2		te	There is presently no way to convert directly from a shared_ptr to a unique_ptr.	Add an interface that performs the conversion. See the attached, Issues with the C++ Standard" paper under Chapter 20, "Conversion from shared_ptr to unique_ptr".	

US 79	20.8.12.2.1		te	[unique.ptr.single.ctor]/5 no longer requires for D not to be a pointer type. This restriction needs to be put back in. Otherwise we have a run time failure that could have been caught at compile time:
				
				unique_ptr<int, void(*)(void*)> p(malloc(sizeof(int))); // should not compile
				
				unique_ptr<int, void(*)(void*)> p(malloc(sizeof(int)), free); // ok		

JP 44	20.8.13.6		te	The 1st parameter p and 2nd parameter v is now shared_ptr<T> *. It should be shared_ptr<T>&, or if these are shared_ptr<T>* then add the "p shall not be a null pointer" at the requires.	Change shared_ptr<T>& or add the "p shall not be a null pionter" at the requires.	
JP 45	20.9		te	Rep, Period, Clock and Duration don't correspond to concept. template <class Rep, class Period = ratio<1>> class duration; template <class Clock, class Duration = typename	Make concept for Rep, Period, Clock and Duration, and fix 20.9 and wait_until and wait_for's template parameter at 30.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				Clock::duration> class time_point;		
US 80	20.9.2.1	Heading	ed	The section heading does not describe the contents.	Change the heading "is_floating_point" to "treat_as_floating_point". Optionally adjust the section's label similarly.	

US 81	20.9.3		te	chrono::duration is missing the modulous operator for both member and non-member arithmetic. This operator is useful for finding the position of a duration within a bounded time frame. Having it be built in to duration is safer than requiring the client to extract and operate directly on the duration's representation as the latter will not enforce the correct units of the operation.
				
				Ex:
				
				milliseconds ms(...);
				microseconds us(...);
				
				ms % us; // microseconds
				us % ms; // microseconds
				ms % 5; // milliseconds
				5 % ms; // does not compile
					Add:
					
					template <class Rep, class Period = ratio<1>>
					class duration {
					public:
					...
					duration& operator%(const rep&);
					duration& operator%(const duration&);
					..
					};
					
					template <class Rep1, class Period,
					class Rep2>
					duration<typename common_type<
					Rep1, Rep2>::type, Period>
					operator%(const duration<Rep1, Period>& d,
					const Rep2& s);
					
					template <class Rep1, class Period1,
					class Rep2, class Period2>
					typename common_type<duration<Rep1,
					Period1>, duration<Rep2, Period2>>::type
					operator%(const duration<Rep1, Period1>& lhs,
					const duration<Rep2, Period2>& rhs);	

US 82	20.9.5.3	after ¶ 1	ed	The code synopsis has a minor alignment error.	Align "rep" with the other symbols defined in this synopsis.	
UK 216	21		Te	All the containers use concepts for their iterator usage, exect for basic_string. This needs fixing.	Use concepts for iterator template parameters throughout the chapter.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
JP 46	21.2, 21.3		te	The basic_string does not use concept.	The "class Alloc" is changed to "Allocator Alloc". The "class InputIterator" is changed to "InputIterator Iter". // 21.3, basic_string: template<class charT, class traits = char_traits<charT>, Allocator Alloc = allocator<charT> > class basic_string; template<class charT, class traits, Allocator Alloc> basic_string<charT,traits,Alloc> operator+(const basic_string<charT,traits,Alloc>& lhs, const basic_string<charT,traits,Alloc>& rhs); template<class charT, class traits, Allocator Alloc> basic_string<charT,traits,Alloc>&& operator+(basic_string<charT,traits,Alloc>&& lhs, const basic_string<charT,traits,Alloc>& rhs); template<class charT, class traits, Allocator Alloc> basic_string<charT,traits,Alloc>&& operator+(const basic_string<charT,traits,Alloc>& lhs, basic_string<charT,traits,Alloc>&& rhs); template<class charT, class traits, Allocator Alloc> basic_string<charT,traits,Alloc>&& operator+(basic_string<charT,traits,Alloc>&& lhs, basic_string<charT,traits,Alloc>&& rhs); template<class charT, class traits, Allocator Alloc>	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					basic_string<charT,traits,Alloc> operator+(const charT* lhs, const basic_string<charT,traits,Alloc>& rhs); template<class charT, class traits, Allocator Alloc> basic_string<charT,traits,Alloc>&& operator+(const charT* lhs, basic_string<charT,traits,Alloc>&& rhs); template<class charT, class traits, Allocator Alloc> basic_string<charT,traits,Alloc> operator+(charT lhs, const basic_string<charT,traits,Alloc>& rhs); template<class charT, class traits, Allocator Alloc> basic_string<charT,traits,Alloc>&& operator+(charT lhs, basic_string<charT,traits,Alloc>&& rhs); template<class charT, class traits, Allocator Alloc> basic_string<charT,traits,Alloc> operator+(const basic_string<charT,traits,Alloc>& lhs, const charT* rhs); template<class charT, class traits, Allocator Alloc> basic_string<charT,traits,Alloc>&& operator+(basic_string<charT,traits,Alloc>&& lhs, const charT* rhs); template<class charT, class traits, Allocator Alloc> basic_string<charT,traits,Alloc> operator+(const basic_string<charT,traits,Alloc>& lhs, charT rhs);	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					template<class charT, class traits, Allocator Alloc> basic_string<charT,traits,Alloc>&& operator+(basic_string<charT,traits,Alloc>&& lhs, charT rhs); template<class charT, class traits, Allocator Alloc> bool operator==(const basic_string<charT,traits,Alloc>& lhs, const basic_string<charT,traits,Alloc>& rhs); template<class charT, class traits, Allocator Alloc> bool operator==(const charT* lhs, const basic_string<charT,traits,Alloc>& rhs); template<class charT, class traits, Allocator Alloc> bool operator==(const basic_string<charT,traits,Alloc>& lhs, const charT* rhs); template<class charT, class traits, Allocator Alloc> bool operator!=(const basic_string<charT,traits,Alloc>& lhs, const basic_string<charT,traits,Alloc>& rhs); template<class charT, class traits, Allocator Alloc> bool operator!=(const charT* lhs, const basic_string<charT,traits,Alloc>& rhs); template<class charT, class traits, Allocator Alloc> bool operator!=(const basic_string<charT,traits,Alloc>& lhs, const charT* rhs);	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					template<class charT, class traits, Allocator Alloc> bool operator< (const basic_string<charT,traits,Alloc>& lhs, const basic_string<charT,traits,Alloc>& rhs); template<class charT, class traits, Allocator Alloc> bool operator< (const basic_string<charT,traits,Alloc>& lhs, const charT* rhs); template<class charT, class traits, Allocator Alloc> bool operator< (const charT* lhs, const basic_string<charT,traits,Alloc>& rhs); template<class charT, class traits, Allocator Alloc> bool operator> (const basic_string<charT,traits,Alloc>& lhs, const basic_string<charT,traits,Alloc>& rhs); template<class charT, class traits, Allocator Alloc> bool operator> (const basic_string<charT,traits,Alloc>& lhs, const charT* rhs); template<class charT, class traits, Allocator Alloc> bool operator> (const charT* lhs, const basic_string<charT,traits,Alloc>& rhs); template<class charT, class traits, Allocator Alloc> bool operator<=(const basic_string<charT,traits,Alloc>& lhs, const basic_string<charT,traits,Alloc>& rhs);	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					template<class charT, class traits, Allocator Alloc>
					  bool operator<=(const basic_string<charT,traits,Alloc>& lhs,
					  const charT* rhs);
					
					template<class charT, class traits, Allocator Alloc>
					  bool operator<=(const charT* lhs,
					  const basic_string<charT,traits,Alloc>& rhs);
					
					template<class charT, class traits, Allocator Alloc>
					  bool operator>=(const basic_string<charT,traits,Alloc>& lhs,
					  const basic_string<charT,traits,Alloc>& rhs);
					
					template<class charT, class traits, Allocator Alloc>
					  bool operator>=(const basic_string<charT,traits,Alloc>& lhs,
					  const charT* rhs);
					
					template<class charT, class traits, Allocator Alloc>
					  bool operator>=(const charT* lhs,
					  const basic_string<charT,traits,Alloc>& rhs);
					
					// 21.3.8.8: swap
					template<class charT, class traits, Allocator Alloc>
					  void swap(basic_string<charT,traits,Alloc>& lhs,
					  basic_string<charT,traits,Alloc>& rhs);
					
					template<class charT, class traits, Allocator Alloc>
					  void swap(basic_string<charT,traits,Alloc>&& lhs,
					  basic_string<charT,traits,Alloc>& rhs);	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					template<class charT, class traits, Allocator Alloc>
					  void swap(basic_string<charT,traits,Alloc>& lhs,
					  basic_string<charT,traits,Alloc>&& rhs);
					
					// 21.3.8.9: inserters and extractors
					template<class charT, class traits, Allocator Alloc>
					  basic_istream<charT,traits>&
					  operator>>(basic_istream<charT,traits>&& is,
					  basic_string<charT,traits,Alloc>& str);
					
					template<class charT, class traits, Allocator Alloc>
					  basic_ostream<charT, traits>&
					  operator<<(basic_ostream<charT, traits>&& os,
					  const basic_string<charT,traits,Alloc>&
					str);
					
					template<class charT, class traits, Allocator Alloc>
					  basic_istream<charT,traits>&
					  getline(basic_istream<charT,traits>&& is,
					  basic_string<charT,traits,Alloc>& str,
					  charT delim);
					
					template<class charT, class traits, Allocator Alloc>
					  basic_istream<charT,traits>&
					  getline(basic_istream<charT,traits>&& is,
					  basic_string<charT,traits,Alloc>& str);
					
					// 21.3 Class template basic_string
					namespace std {
					  template<class charT, class traits =
					char_traits<charT>,
					  Allocator Alloc = allocator<charT> >
					  class basic_string {
					public:
					  // types:
					  typedef traits traits_type;	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					typedef typename traits::char_type value_type;
					typedef Alloc allocator_type;
					typedef typename Alloc::size_type size_type;
					typedef typename Alloc::difference_type difference_type;
					typedef typename Alloc::reference reference;
					typedef typename Alloc::const_reference const_reference;
					typedef typename Alloc::pointer pointer;
					typedef typename Alloc::const_pointer const_pointer;
					typedef implementation-defined iterator; // See 23.1
					typedef implementation-defined const_iterator; // See 23.1
					typedef std::reverse_iterator<iterator> reverse_iterator;
					typedef std::reverse_iterator<const_iterator> const_reverse_iterator;
					static const size_type npos = -1;
					
					// 21.3.2 construct/copy/destroy:
					explicit basic_string(const Alloc& a = Alloc());
					basic_string(const basic_string& str);
					basic_string(basic_string&& str);
					basic_string(const basic_string& str, size_type pos, size_type n = npos,
					const Alloc& a = Alloc());
					basic_string(const charT* s,
					size_type n, const Alloc& a = Alloc());
					basic_string(const charT* s, const Alloc& a = Alloc());
					basic_string(size_type n, charT c, const Alloc& a = Alloc());	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					template<InputIterator Iter>
					basic_string(Iter begin, Iter end,
					const Alloc& a = Alloc());
					basic_string(initializer_list<charT>, const Alloc& = Alloc());
					basic_string(const basic_string&, const Alloc&);
					basic_string(basic_string&&, const Alloc&);
					~basic_string();
					basic_string& operator=(const basic_string& str);
					basic_string& operator=(basic_string&& str);
					basic_string& operator=(const charT* s);
					basic_string& operator=(charT c);
					basic_string& operator=(initializer_list<charT>);
					
					// 21.3.3 iterators:
					...
					
					// 21.3.4 capacity:
					...
					
					// 21.3.5 element access:
					...
					
					// 21.3.6 modifiers:
					...
					
					basic_string& append(const basic_string& str);
					basic_string& append(const basic_string& str, size_type pos,
					size_type n);
					basic_string& append(const charT* s, size_type n);
					basic_string& append(const charT* s);
					basic_string& append(size_type n, charT c);
					template<InputIterator Iter>
					basic_string& append(Iter first, Iter last);
					basic_string& append(initializer_list<charT>);	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					void push_back(charT c);
					
					basic_string& assign(const basic_string& str);
					basic_string& assign(basic_string&& str);
					basic_string& assign(const basic_string& str,
					size_type pos,
					size_type n);
					basic_string& assign(const charT* s, size_type
					n);
					basic_string& assign(const charT* s);
					basic_string& assign(size_type n, charT c);
					template<InputIterator Iter>
					basic_string& assign(Iter first, Iter last);
					basic_string& assign(initializer_list<charT>);
					
					basic_string& insert(size_type pos1, const
					basic_string& str);
					basic_string& insert(size_type pos1, const
					basic_string& str,
					size_type pos2, size_type n);
					basic_string& insert(size_type pos, const charT*
					s, size_type n);
					basic_string& insert(size_type pos, const charT*
					s);
					basic_string& insert(size_type pos, size_type n,
					charT c);
					iterator insert(const_iterator p, charT c);
					void insert(const_iterator p, size_type n, charT
					c);
					template<InputIterator Iter>
					void insert(const_iterator p, Iter first, Iter last);
					void insert(const_iterator p,
					initializer_list<charT>);
					
					...
					basic_string& replace(size_type pos1, size_type
					n1,	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					const basic_string& str);
					basic_string& replace(size_type pos1, size_type n1,
					const basic_string& str,
					size_type pos2, size_type n2);
					basic_string& replace(size_type pos, size_type n1, const charT* s,
					size_type n2);
					basic_string& replace(size_type pos, size_type n1, const charT* s);
					basic_string& replace(size_type pos, size_type n1, size_type n2,
					charT c);
					basic_string& replace(iterator i1, iterator i2,
					const basic_string& str);
					basic_string& replace(iterator i1, iterator i2, const charT* s,
					size_type n);
					basic_string& replace(iterator i1, iterator i2, const charT* s);
					basic_string& replace(iterator i1, iterator i2,
					size_type n, charT c);
					template<InputIterator Iter>
					basic_string& replace(iterator i1, iterator i2,
					Iter j1, Iter j2);
					basic_string& replace(iterator, iterator, initializer_list<charT>);
					
					...
					
					// 21.3.7 string operations:
					...
					
					template <class charT, class traits, Allocator Alloc>
					struct constructible_with_allocator_suffix<
					basic_string<charT, traits, Alloc> > : true_type {	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					};	
JP 47	21.3		ed	Typo. Missing ”>” template <class charT, class traits, Allocator Alloc should be template <class charT, class traits, Allocator Alloc>	Correct typo.	
JP 48	21.3		te	char_traits does not use concept. For example, create CharTraits concept and change as follows: template<class charT, CharTraits traits = char_traits<charT>, class Allocator = allocator<charT> > class basic_string {	Add a concept for char_traits.	
UK 217	21.3		Ed	basic_string refers to constructible_with_allocator_suffix, which I thought was removed?	Remove the lines: template <class charT, class traits, class Alloc struct constructible_with_allocator_suffix< basic_string<charT, traits, Alloc> > : true_type { };	
UK 218	21.3.1	3	Te	The identity "&*(s.begin() + n) == &*s.begin() + n" relies on operator& doing the "right thing", which (AFAICS) there is no requirement for. See my comment under clauses "23.2.1, 23.2.6" (p1 in both cases) - this is the same issue, but I've created a separate comment for basic_string because it is in a different chapter.	See my recommendations for "23.2.1, 23.2.6".	
UK 219	21.3.6.6 [string.replace]	11	Ed	Effects refers to "whose first begin() - i1 elements" However i1 is greater than begin()...	Effects refers to "whose first i1 - begin() elements"	
UK 220	21.3.8.9		Te	The operator<< for basic_string takes the output stream by r-value reference. This is different from the same	Use the same reference type for the all the library types. This should be the r-value reference. There	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				operator for error_code (19.4.2.6), bitset (20.2.6.3), shared_ptr (20.7.13.2.8), complex (26.3.6) and sub_match (28.4)	are other places in the standard where TR1, and new classes, did not receive an 'R-value' update.	
FR 33	22.1.1 [locale]	3	ed	ios_base::iostate err = 0; iostate is a bitmask type and so could be an enumeration. Probably using goodbit is the solution.		
JP 49	22.1.3.2.2		te	codecvt does not use concept. For example, create CodeConvert concept and change as follows. template<CodeConvert Codecvt, class Elem = wchar_t> class wstring_convert {	Add a concept for codecvt.	
JP 50	22.1.3.2.2		te	Add custom allocator parameter to wstring_convert, since we cannot allocate memory for strings from a custom allocator.	Correct as follows. template<class Codecvt, class Elem = wchar_t> class wstring_convert { public: typedef std::basic_string<char> byte_string; typedef std::basic_string<Elem> wide_string; should be template<class Codecvt, class Elem = wchar_t, Allocator WideAllocator = allocator<Elem>, Allocator ByteAllocator = allocator<char>> class wstring_convert { public: typedef std::basic_string<char, char_traits<char>, ByteAllocator> byte_string; typedef std::basic_string<Elem, char_traits<Elem>, WideAllocator> wide_string;	
FI 4	22.2.1.4.1 22.2.1.4.2		ed	to_end and to_limit are both used. Only one is needed.	Change to_limit to to_end.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
FI 5	22.2.1.4.2	#3	ed	[Note: As a result of operations on state, it can return ok or partial and set next == from and to_next != to. —end note] "next" should be "from_next." Also, the sentence applies to all the examples, including do_in and do_out. Reasoning: When reading one element of multibyte source data such as UTF-8, it is possible that from_next is incremented, to_next is unaltered, state is altered and return value is partial. When reading one element of wide character data, the output can be several multibyte characters, so it is possible that from_next is unaltered, to_next is advanced, state is altered and return value is partial.	[Note: As a result of operations on state, do_in and do_out can return ok or partial and set from_next == from and/or to_next != to. —end note]	
FI 6	22.2.1.5 See also 22.2.1.4 (1,2,3)		te	codecvt_byname is only specified to work with locale names. There is no built-in means to find a codecvt with a character set's name. A locale and a character set are not the same. If the user has data which is encoded in a certain character set and she wants to create a codecvt which can convert from that character set to another one, she must iterate through locales to find one, or use external means (iconv, ICU's uconv). Specifying a locale with the character set is not a suitable solution, since there is no built-in mapping from character sets to locales. It is only possible to inquire the character set once the locale is known.	There should be a built-in means to find a codecvt with a pair of character set names.	
FI 7	22.2.1.4	1,2,3	ed	The word "codeset" is used, whereas the word "character set" is used elsewhere in the text. The words are intended to convey the same concept, so only one should be used (or always both together).	Change "codeset" to "character set."	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
JP 51	22.2.5.1.1	7th paragraph, 1st line	ed	A parameter `end' should be `fmtend'. get() function had two `end' parameters at N2321. iter_type get (iter_type s, iter_type end, ios_base& f, ios_base::iostate& err, tm* t, const char_type* fmt, const char_type *end) const; The function prototype of get() has been corrected at N2800, but a Requires statement still refers `end' parameter.	Correct as follows. Requires: [fmt,end) shall be a valid range. should be Requires: [fmt,fmtend) shall be a valid range.	
JP 52	22.2.5.1, 22.2.5.2, 22.2.6.1		te	InputIterator does not use concept.	Correct as follows. 22.2.5.1 template <class charT, class InputIterator = istreambuf_iterator<charT> > class time_get : public locale::facet, public time_base { public: typedef charT char_type; typedef InputIterator iter_type; should be template <class charT, InputIterator InputIter = istreambuf_iterator<charT> > class time_get : public locale::facet, public time_base { public: typedef charT char_type; typedef InputIter iter_type; 22.2.5.2 template <class charT, class InputIterator = istreambuf_iterator<charT> >	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					class time_get_byname : public time_get<charT, InputIterator> {
					public:
					typedef time_base::dateorder dateorder;
					typedef InputIterator iter_type;
					
					should be
					
					template <class charT, InputIterator InputIter = istreambuf_iterator<charT> >
					class time_get_byname : public time_get<charT, InputIter> {
					public:
					typedef time_base::dateorder dateorder;
					typedef InputIter iter_type;
					
					22.2.6.1
					
					template <class charT,
					class InputIterator = istreambuf_iterator<charT> >
					class money_get : public locale::facet {
					public:
					typedef charT char_type;
					typedef InputIterator iter_type;
					
					should be
					
					template <class charT,
					InputIterator InputIter = istreambuf_iterator<charT> >
					class money_get : public locale::facet {
					public:
					typedef charT char_type;
					typedef InputIter iter_type;	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

JP 53	22.2.5.3 , 22.2.5.4		te	OutputIterator does not use concept.	Correct as follows.
					
					22.2.5.3
					
					template <class charT, class OutputIterator = ostreambuf_iterator<charT> >
					class time_put : public locale::facet {
					public:
					typedef charT char_type;
					typedef OutputIterator iter_type;
					
					should be
					
					template <class charT, OutputIterator OutputIter = ostreambuf_iterator<charT> >
					class time_put : public locale::facet {
					public:
					typedef charT char_type;
					typedef OutputIter iter_type;
					
					22.2.5.4
					
					template <class charT, class OutputIterator = ostreambuf_iterator<charT> >
					class time_put_byname : public time_put<charT, OutputIterator>
					{
					public:
					typedef charT char_type;
					typedef OutputIterator iter_type;
					
					should be
					
					template <class charT, OutputIterator OutputIter = ostreambuf_iterator<charT> >
					class time_put_byname : public time_put<charT,	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					OutputIter> { public: typedef charT char_type; typedef OutputIter iter_type;	
JP 54	23	2nd paragraph, Table 79	ed	There is not <forward_list> in Table 79.	Add <forward_list> between <deque> and <list>.	
UK 221	23	Table 79	Ed	The table is missing the new <forward_list> header.	Add <forward_list> to the table for sequence containers. Alternative (technical) solution might be to merge <forward_list> into <list>.	
UK 222	23		Te	It is not clear what purpose the Requirement tables serve in the Containers clause. Are they the definition of a library Container? Or simply a conventient shorthand to factor common semantics into a single place, simplifying the description of each subsequent container? This becomes an issue for 'containers' like array, which does not meet the default-construct-to-empty requirement, or forward_list which does not support the size operation. Are these components no longer containers? Does that mean the remaining requirements don't apply? Or are these contradictions that need fixing, despite being a clear design decision?	Clarify all the tables in 23.1 are there as a convenience for documentation, rather than a strict set of requirements. Containers should be allowed to relax specific requirements if they call attention to them in their documentation. The introductory text for array should be expanded to mention a default constructed array is not empty, and forward_list introduction should mention it does not provide the required size operation as it cannot be implemented efficiently.	
JP 55	23.1.1	3rd paragraph, 4th line	ed	It seems that “the MinimalAllocator concep” is the typo of “the MinimalAllocator concept”.	Change to … models the MinimalAllocator concept.	
UK 223	23.1.1	3	Te	The library does not define the MinimalAllocator or ScopedAllocator concepts, these were part of an earlier Allocators proposal that was rejected.	Remove the references to MinimalAllocator and ScopedAllocator, or add definitions for these concepts to clause 20.7.	
UK 224	23.1.1	8	Te	This paragraph implicitly requires all containers in clause 23 to support allocators, which std::array does not.	Add an 'unless otherwise specified' rider somewhere in p8, or move the whole array container from clause 23 [containers] to clause 20	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					[utilies] to accompany bitset, pair and tuple.	
UK 225	23.1.1	Table 81	Ed	Inconsistent words used to say the same thing. Table 80 describes iterator/const_iterator typedef as returning an "iterator type whose value type is T". Table 81 expresses the same idea as an "iterator type pointing to T". Express identical ideas with the same words to avoid accidentally introducing subtlety and confusion	Change return types for X::(const)_reverse_iterator to say "iterator type whose value type is (const) T".	
UK 226	23.1.1	10	Te	<array> must be added to this list. In particular it doesn't satisfy: - no swap() function invalidates any references, pointers, or iterators referring to the elements of the containers being swapped. and probably doesn't satisfy: — no swap() function throws an exception.	If <array> remains a container, this will have to also reference array, which will then have to say which of these points it satisfies.	
UK 227	23.1.1	Table 80	Ed	The post-condition for a = rv uses the word “construction” when it means “assignment”	Replace the word “construction” with the word “assignment”	
UK 228	23.1.1	3	Ed	Line 4 contains a spelling mistake in the fragment "MinimalAllocator concep."	Replace "concep" with "concept"	
UK 229	23.1.1	3	Ed	The fragment "A container may directly call constructors" is not technically correct as constructors are not callable.	Replace "A container may directly call constructors and destructors for its stored objects" with something similar to "A container may directly construct its stored objects and call destructors for its stored objects"	
UK 230	23.1.2	1	Te	“implementations shall consider the following functions to be const” - what does this mean? I don't understand what it means by implementations considering the functions to be const – surely they are either declared const or not?	Clarify what is meant and what requirements an implementation must satisfy.	
JP 56	23.1.3	12th paragraph, Table 84	ed	`array’ is unstated in Table 84 - Optional sequence container operations.	Add `array’ to Container field for the following Expression. a.front() a.back() a[n]	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					a.at(n)	
UK 231	23.1.3	9-11	Te	These paragraphs are redundant now that Concepts define what it means to be an Iterator and guide overload resolution accordingly.	Strike 23.1.3p9-11. Make sure std::basic_string has constraints similar to std::vector to meet this old guarantee.	
UK 232	23.1.3	Table 84	Te	match_results may follow the requirements but is not listed a general purpose library container.	Remove reference to match_results against a[n] operation	
UK 233	23.1.3	Table 84	Te	Add references to the new containers.	Add reference to array to the rows for: a.front(), a. back(), a[n] a.at(n). Add reference to forward_list to the rows for: a.front(), a.emplace_front(args), a.push_front(t), a.push_front(rv), a.pop_front(). Add reference to basic_string to the row for: a.at(n).	
UK 234	23.1.3	Table 84	Te	Ther reference to iterator in semantics for back should also allow for const_iterator when called on a const-qualified container. This would be ugly to specify in the 03 standard, but is quite easy with the addition of auto in this new standard.	Replace iterator with auto in semantics for back: { auto tmp = a.end(); --tmp; return *tmp; }	
UK 235	23.1.3	1	Ed	“The library provides three basic kinds of sequence containers: vector, list, and deque” - text appears to be out of date re addition of array and forward_list	Change the text to read: “The library provides five basic kinds of sequence containers: array, deque, forward_list, list and vector”.	
UK 236	23.1.3	2	Ed	[I've moved (1) into a separate comment because I believe it is editorial in the simple sense, whereas (2) and (3) are not so straight forward] (2) “vector is the type of sequence container that should be used by default” -- As I understand it vector was considered first port of call because the things it has in common with the native array make programmers (especially those new to the container library) feel like they are on familiar territory. However, we now have the array container, so I think this should be recommended as the first port of call. (3) This paragraph is actually giving guidance on the use of the containers and should not be normative text	Remove this paragraph	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 237	23.1.3	2	Ed	vector, list, and deque offer the programmer different complexity trade-offs and should be used accordingly - this ignores array and forward_list	Modify the text to read: "array, deque, forward_list, list and vector offer the programmer different complexity trade-offs and should be used accordingly"	
UK 238	23.1.4	6	Te	Leaving it unspecified whether or not iterator and const_iterator are the same type is dangerous, as user code may or may not violate the One Definition Rule by providing overloads for both types. It is probably too late to specify a single behaviour, but implementors should document what to expect. Observing that problems can be avoided by users restricting themselves to using const_iterator, add a note to that effect.	Change 'unspecified' to 'implementation defined'. Add "[Note: As itererator and const_iterator have identical semantics in this case, and iterator is convertible to const_iterator, users can avoid violating the One Definition Rule by always using const_iterator in their function parameter lists -- end note]	
UK 239	23.1.4	85	Te	It is not possible to take a move-only key out of an unordered container, such as (multi)set or (multi)map, or the new hashed containers.	Add below a.erase(q), a.extract(q), with the following notation: a.extract(q), Return type pair<key, iterator> Extracts the element pointed to by q and erases it from the set. Returns a pair containing the value pointed to by q and an iterator pointing to the element immediately following q prior to the element being erased. If no such element exists,returns a.end().	
UK 240	23.1.6.1	12	Te	The axiom EmplacePushEquivalence should be asserting the stronger contract that emplace and insert return the same iterator value, not just iterators that dereference to the same value. This is a similar statement that is easier to express and should be equivalent - the idea that insert and emplace might return iterator values that do not compare equal but point to the same element should fail somewhere in the iterator concepts. Also, this axiom should be renamed to reflect its connection with insert, rather than push_front/push_back,	Remove the * to deference the returned iterator either side of the == in the EmplacePushEquivalence axiom, rename the axiom EmplacementInsertionEquivalence : requires InsertionContainer<C> && Constructible<value_type, Args...> axiom EmplacementInsertionEquivalence(C c, const_iterator position, Args... args) { emplace(c, position, args...) == insert(c, position, value_type(args...)); }	
JP 57	23.1.6.3	1st paragraph,	ed	Typo, duplicated "to" "to to model insertion container concepts."	Remove one.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
		4th line				
UK 241	23.2.1		Te	std::array does not have an allocator, so need to document an exception to the requirements of 23.1.1p3	add exception to 23.1.1p3	
UK 242	23.2.1	3	Ed	std:: qualification no longer needed for reverse_iterator.	remove std:: qualification from std::reverse_iterator<iterator> and std::reverse_iterator<const_iterator>	
UK 243	23.2.1	3	Te	Most containers, and types in general have 3 swaps: swap(T&, T&) swap(T&&, T&) swap(T&, T&&) But array only has swap(T&, T&).	add the other two swaps.	
UK 244	23.2.1, 23.2.6	1	Te	The validity of the expression &a[n] == &a[0] + n is contingent on operator& doing the “right thing” (as captured by the CopyConstructible requirements in table 30 in C++2003). However this constraint has been lost in the Concepts of C++0x. This applies to vector and array (it actually applies to string also, but that's a different chapter, so I'll file a separate comment there and cross-reference).	Define a ContiguousStrorage and apply it to vector,array and string. The Concept (supplied by Alisdair M) looks like this: Concept< typename C > ContiguousStrorage { requires Container<C>; typename value_type = C::value_type; value_type * data(C); axiom Contiguous { C c; true = equal_ranges(data(c), data(c) + size(c), begin(c)); } };	
UK 245	23.2.3	2	Te	The predicate types used in special member function of forward_list should be CopyConstructible, as per the algorithms of the same name. Note: an alternate solution would be to require these callable concepts to be CopyConstructible in clause 20, which would simplify the library specification in general. See earlier comment for details, that would render this one redundant.	Add CopyConstructible requirement to the following signatures: template <Predicate<auto, T> Pred> requires CopyConstructible<Pred> void remove_if(Pred pred); template <EquivalenceRelation<auto, T> BinaryPredicate> requires CopyConstructible<BinaryPredicate> void unique(BinaryPredicate binary_pred); template <StrictWeakOrder<auto, T> Compare> requires CopyConstructible<Compare> void merge(forward_list<T,Alloc>&& x, Compare comp); template <StrictWeakOrder<auto, T> Compare> requires CopyConstructible<Compare> void sort(Compare comp);	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
JP 58	23.2.3.2	1st line before 1st paragraph	ed	Unnecessary "{" exists before a word iterator like "{iterator before_begin()".	Remove "{"	
JP 59	23.2.4.4		ed	Types of the third and forth parameter of splice() are iterator at 23.2.4.4, though types of them are const_iterator at 23.2.4. (They are both const_iterator on N2350)	Correct as follows. void splice(const_iterator position, list<T,Allocator>&& x, iterator i); void splice(const_iterator position, list<T,Allocator>&& x, iterator first, iterator last); should be void splice(const_iterator position, list<T,Allocator>&& x, const_iterator i); void splice(const_iterator position, list<T,Allocator>&& x, const_iterator first, const_iterator last);	
US 83	23.2.6.2	7	ed	"shrink_to_fint" should be "shrink_to_fit".		
UK 246	23.3.2.2		Ed	The content of this sub-clause is purely trying to describe in words the effect of the requires clauses on these operations, now that we have Concepts. As such, the desctiption is more confusing than the signature itself. The semantic for these functions is adequately covered in the requirements tables in 23.1.4.	Strike 23.3.2.2 entirely. (but do NOT strike these signatures from the class template definition!)	
UK 247	24.1		Ge	Iterator concepts are not extensive enough to merit a whole new header, and should be merged into <concpts>. This is particularly important for supporting the new for loop syntax which requires access to the Range concept. The required header to enable this syntax shoud have a simple name, like <concepts>, rather than something awkward to type like	Move the concepts of <iterator_concepts> into the <concepts> header. We take no position on moving the text from Clause 24 to Clause 20 though.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				<iterator_concepts>.		
UK 248	24.1	6	Ed	The text "so for any iterator type there is an iterator value that points past the last element of a corresponding container" is slightly misleading. Iterators can refer into generalised ranges and sequences, not just containers. A broader term than 'container' should be used.	Replace the reference to container with a more appropriate concept	
UK 250	24.1.1		Te	A default implementation should be supplied for the post-increment operator to simplify implementation of iterators by users.	Copy the Effects clause into the concept description as the default implementation. Assumes a default value for postincrement_result	
UK 251	24.1.1	3	Te	The post-increment operator is dangerous for a general InputIterator. The multi-pass guarantees that make it meaningful are defined as part of the ForwardIterator refinement. Any change will affect only constrained templates that have not yet been written, so should not break existing user iterators which remain free to add these operations. This change will also affect the generalised OutputIterator, although there is no percieved need for the post-increment operator in this case either.	Move declaration of postincrement operator and postincrement_result type from Interator concept to the ForwardIterator concept	
UK 252	24.1.2	3	Ed	istream_iterator is not a class, but a class template	Change 'class' to 'class template' in the note.	
UK 253	24.1.3	1	Ed	First sentance does not make gramatical sense, Seems to be missing the words 'if it' by comparison with similar sentance in other subsections	Add the words 'if it' : "X satisfies the requirements of an output iterator IF IT meets the syntactic and semantic requirements"	
UK 254	24.1.3	5	Te	This postcondition for pre-increment operator should be written as an axiom	Move the postcondition into the concept definition as an axiom	
UK 255	24.1.4	4	Te	This postcondition for pre-increment operator should be written as an axiom	Move the postcondition into the concept definition as an axiom	
UK 256	24.1.5	3, 4, 5	Te	The relationship between pre- and post- decrement should be expressed as an axiom.	Move the text specification of pre/post-conditions and behaviour into an axiom within the BidirectionalIterator concept	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 257	24.1.5		Te	There is a reasonable default for postdecrement_result type, which is X. X is required to be regular, therefore CopyConstructible and a valid ResultType. Together with the next comment this simplifies user defined iterator implentations	Add the default : typename postincrement_result = X;	
UK 258	24.1.5		Te	A default implementation should be supplied for the post-decrement operator to simplify implementation of iterators by users.	Copy the Effects clause into the concept description as the default implementation. Assumes a default value for postincrement_result	
UK 259	24.1.5		Te	postdecrement_result is effectively returning a copy of the original iterator value, so should have similar constraints, rather than just HasDereference. If Concepts do not support this circularity of definition suggest that concepts feature may want a little more work	Add the requirement: requires Iterator< postdecrement_result >;	
UK 260	24.1.5	6	Te	The effects clause for post-decrement iterator should be available as an axiom and a default implementation, where the compiler can make better use of it.	Move the Effects clause into the BidirectionalIterator concept definition as an axiom, and as the default implementation for the operation.	
UK 249	24.1.6	2	Te	The semantic for operator+= should also be provided as a default implementation to simplify implementation of user-defined iterators	Copy the text from the effects clause into the RandomAccessIterator concept as the default implementaiton.	
UK 261	24.1.6		Te	To simplify user defined random access iterator types, the subscript_reference type should default to reference	typename subscript_reference = reference;	
UK 262	24.1.6	3, 4	Te	Effects and post-conditions for operator+ are more useful if expressed as axioms, and supplied as default implementations.	Move the effects and Postcondition definitions into the RandomAccessIterator concept and copy the code in the specification as the default implementation of these operations.	
UK 263	24.1.6	5	Te	This requirement on operator-= would be better expressed as a default implementation in the concept, with a matching axiom	Move the specification for operator-= from the returns clause into an axiom and default implementation within the RandomAccessIterator	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					concept	
UK 264	24.1.6	6	Te	Effects clauses are better expressed as axioms where possible.	Move code in operator- effects clause into RandomAccessIterator concept as both a default implementation and an axiom	
UK 265	24.1.6	8	Te	This effects clause is nonesense. It looks more like an axiom stating equivalence, and certainly an effects clause cannot change the state of two arguments passed by const reference	Strike the Effects clause	
UK 266	24.1.6	9	Te	This sentance should be provided as a default definition, along with a matching axiom	Move the Returns clause into the spefication for RandomAccessIterator operator- as a default implementation. Move the Effects clause as the matching axiom.	
UK 267	24.1.6	24.1.6	Te	The code in the Requires clause for RandomAccessIterator operator[] would be better expressed as an axiom.	Rewrite the Requires clause as an axiom in the RandomAccessIterator concept	
UK 268	24.1.6	12	Ed	This note is potentialy confusing as __far enters the syntax as a clear language extension, but the note treats it as a regular part of the grammar. It might be better expressed using attributes in the current wording.	Clean up the note to either avoid using language extension, or spell out this is a constraint to support extensions.	

JP 60	24.1.8	1st paragraph	te	Capability of an iterator is too much restricted by concept.
				
				Concept of std::Range is defined as:
				
				concept Range<typename T> {
				InputIterator iterator;
				iterator begin(T&);
				iterator end(T&);
				}
				
				So the following code generates an error.
					Add InputRange, OutputRange, ForwardRange, BidirectionalRange and RandomAccessRange.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

				template <std::Range Rng>
				void sort(Rng& r)
				{
				// error! Rng::iterator does not satisfy requirements of a random
				// access iterator.
				std::sort(begin(r), end(r));
				}
				
				std::vector<int> v; // vector::iterator is a random access iterator.
				sort(v);
				
				This is because the concept of an iterator of std::Range is InputIterator. For this reason, a random access iterator loses its capability when passed to a std::Range parameter.
				
				To be able to work the above code, we need to write as follows:
				
				template <std::Range T>
				requires std::RandomAccessIterator<T::iterator> &&
				std::ShuffleIterator<T::iterator> &&
				std::LessThanComparable<T::iterator::value_type>
				void sort(T& r)
				{
				sort(begin(r), end(r));
				}
				
				std::vector<int> v;
				sort(v);
				
				It needs quiet a few amount of codes like this only to recover random access capability from InputIterator concept.		

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

				We can write the following valid code with Boost.Range, which is implemented with using C++03 SFINAE.
				
				template <class Range>
				void sort(Range& r)
				{
				std::sort(boost::begin(r), boost::end(r));
				}
				
				std::vector<int> v;
				sort(v); // OK
				
				One of the motivation to introduce concepts are supporting template programming techniques by language directly to eliminate hacky techniques such as tag-dispatch, SFINAE and Type Traints. But SFINAE will be kept using because it needs quite a few amount of codes without using SFAINAE.		

UK 269	24.3	3	Ed	'decrements for negative n' seems to imply a negative number of decrement operations, which is odd.	Need simple, clearer wording	
UK 270	24.3	4	Te	The reachability constraint in p5 means that a negavite result, implying decrements operations in p4, is not possible	Split the two overloads into separate descriptions, where reachability is permitted to be in either direction for RandomAccessIterator	
UK 271	24.3	6,7	Te	next/prev return an incremented iterator without changing the value of the original iterator. However, even this may invalidate an InputIterator. A ForwardIterator is required to guarantee the 'multipass' property.	Replace InputIterator constraint with FOrwardIterator in next and prev function templates.	
UK 272	24.4		Te	reverse_iterator and move_iterator use different formulations for their comparison operations. move_iterator merely requires the minimal set of operations, < and ==, from its underlying iterator and synthesizes all oprations from these two. reverse_iterator	Rephrase the reverse_iterator comparison operations using only operators < and ==, as per the move_iterator specification.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				relies on the undelying iterator to support all comparision operations directly. In practice, move_iterator can only be implemented this way as it must support iterator types that are merely InputIterators, and so SemiRegular and not Regular. However, reserse_iterator has an existing specification and any change of semantic could change behaviour of conforming programs today - although an iterator that yields different results for (a > b) than (b < a) may fall foul of some semantic consistency requirements, even if the syntax is met.		
UK 274	24.4, 24.5		Ed	The subclauses for standard iterator adaptors could be better organised. There are essentially 3 kinds of iterator wrappers provided, stream iterators adapt streams and get their own subsection. insert iterators adapt containers, and get their own subsection but it is inserted into the middle of 24.4 Predifined iterators. reverse_iterator and move_iterator adpat other iterators, but their presentation is split by insert iterators	Promote 24.4.2 [insert.iterators] up one level to 24.6. Emphasize that insert iterators adapt containers Retarget 24.4 [predef.iterators] as iterator adapters for iterator templates that wrap other iterators.	
UK 275	24.4.1.1		Te	The constructor template taking a single Iterator argument will be selected for Copy Initialization instead of the non-template constructor taking a single Iterator argument selected by Direct Initialization.	The reverse_iterator template constructor taking a single Iterator argument should be explicit.	
UK 276	24.4.1.1		Ed	It is odd to have a mix of declaration stlyes for operator+ overloads. Prefer if either all are member functions, or all are 'free' functions.	Make the member operators taking a difference_type argument non-member operators	
UK 277	24.4.1.2.1	1	Te	The default constructor default-initializes current, rather than value-initializes. This means that when Iterator corresponds to a trivial type, the current member is left un-initialized, even when the user explictly requests value intialization! At this point, it is not safe to perform any operations on the reverse_iterator other than assign it a new value or destroy it. Note that this does correspond to the basic definition of a singular iterator.	i/ Specify value initialization rather than default intialization or ii/ specify = default; rather than spell out the semantic. This will at least allow users to select value initialization and copy the iterator value. or iii/ Add a note to the description emphasizing the singular nature of a value-initialized reserve iterator.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 278	24.4.1.2.1	3	Te	There is an inconsistency between the constructor taking an iterator and the constructor template taking a reverse_iterator where one passes by value, and the other by const reference. The by-value form is preferred as it allows for efficient moving when copy elision fails to kick in.	Change the const reverse_iterator<U> & parameter to pass-by-value	
UK 279	24.4.1.2.12, 24.4.3.2.12		Te	The reason the return type became unspecified is LWG issue 386. This reasoning no longer applies as there are at least two ways to get the right return type with the new language facilities added since the previous standard.	Specify the return type using either decltype or the Iter concept map	
UK 280	24.4.1.2.4		Ed	The presence of the second iterator value is surprising for many readers who underestimate the size of a reverse_iterator object. Adding the exposition only member that is required by the semantic will reduce confusion.	Add reverse_iterator expsoition only member tmp as a comment to class declaration, as a private member	
UK 281	24.4.1.2.5		Te	The current specification for return value will always be a true pointer type, but reverse_iterator supports proxy iterators where the pointer type may be some kind of 'smart pointer'	Replace the existing returns specification with a copy of the operator* specification that returns this->tmp.operator->	
UK 282	24.4.2.1, 24.4.2.2.2, 24.4.2.3, 24.4.2.4.2, 24.4.2.5, 24.4.2.6.2	n/a	Te	Insert iterators of move-only types will move from lvalues	Add an additional constrained overload for operator= that requires !CopyConstructible<Cont::value_type> and mark it =delete.	
UK 283	24.4.2.5, 24.4.2.6.4		Te	postincrement operator overloads traditionally return by value - insert_iterator is declared as return by reference. The change is harmless in this case, but either front/back_insert_iterator should take the matching change for consistency, or this function should return-by-value	change operator++(int) overload to return by value, not reference. Affects both class summary and operator definition in p	
JP	24.4.3.2.1	2nd	ed	Typo.	Add "i"	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
61		paragraph, 1st line		"intializing" should be "initializing"		
UK 284	24.5		Te	The stream iterators need constraining with concepts/requrires clauses.	Provide constraints	
UK 285	24.5.1	1,2	Ed	Much of the content of p1 and the whole of p2 is a redundant redefinition of InputIterator. It should be simplified	Strike p2. Simplify p1 and add a cross-reference to the definition of InputIterator concept.	
UK 286	24.5.1	3	Ed	To the casual reader it is not clear if it is intended to be able to assign to istream_iterator objects. Specifying the copy constructor but relying on the implicit copy-assign operator is confusing.	Either provide a similar definition to the copy-assign operator as for the copy constructor, or strike the copy constructor	
UK 287	24.5.1.1	2	Te	It is not clear what the intial state of an istream_iterator should be. Is _value_ initialized by reading the stream, or default/value initialized? If it is initialized by reading the stream, what happens if the initialization is deferred until first dereference, when ideally the iterator value should have been that of an end-of-stream iterator which is not safely dereferencable?	Specify _value_ is initialized by reading the stream, or the iterator takes on the end-of-stream value if the stream is empty	
UK 288	24.5.1.1	3	Ed	The provided specification is vacuous, offering no useful information.	Either strike the specification of the copy constructor, or simply replace it with an = default definition.	
UK 289	24.5.1.2	6	Ed	It is very hard to pick up the correct specification for istream_iterator::operator== as the complete specification requires reading two quite disconnected paragraphs, 24.5.1p3, and 24.5.1.2p6. Reading just the specifaction of the operation itself suggests that end-of-stream iterators are indistinguishable from 'valid' stream iterators, which is a very dangerous misreading.	Merge 24.5.1p3, equality comparison of end-of-stream-iterators, into 24.5.1.2p6, the specification of the equality operator for istream_iterator.	
UK 290	24.5.2	1	Te	The character type of a string delimiter for an ostream_iterator should be const charT *, the type of the elements, not char *, a narrow character string.	Replace char * with const charT *	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 291	24.5.2.2	2	Te	ostream_iterator postincrement operator returns by reference rather than by value. This may be a small efficiency gain, but it is otherwise unconventional. Prefer return-by-value.	ostream_iterator operator++(int);	
FR 34	24.5.3 [istreambuf.iterator]		ed	There are two public sections, and the content of the second one is indented with respect to the first. I don't it should be.		
UK 292	24.5.3	1	Ed	Prefer the use of the new nullptr constant to the zero literal when using a null pointer in text.	Change istreambuf_iterator(0) to istreambuf_iterator(nullptr)	
UK 293	24.5.3	2,3,4	Ed	The listed paragraphs redundantly redefine an input iterator, and redundancy can be a dangerous thing in a specification. Suggest a simpler phrasing below.	2. The result of operator*() on an end of stream is undefined. For any other iterator value a char_type value is returned. 3. Two end of stream iterators are always equal. An end of stream iterator is not equal to a non-end of stream iterator. 4. As istreambuf_iterator() is an InputIterator but not a ForwardIterator, istreambuf_iterators object can be used only for one-pass algorithms. It is not possible to assign a character via an input iterator.	
UK 294	24.5.3.2	2	Te	Implicit converting constructors can be invoked at surprising times, so there should always be a good reason for declaring one.	Mark the two single-argument constructors take a stream or stream buffer as explicit. The proxy constructor should remain implicit. explicit istreambuf_iterator(basic_istream<charT,traits>& s) throw(); explicit istreambuf_iterator(basic_streambuf<charT,traits> * s) throw();	
UK 295	25		Te	THere is a level of redundancy in the library specification for many algorithms that can be eliminated with the combination of concepts and default parameters for function templates. Eliminating redundancy simplified specification and reduces the risk of inttroducing	Adopt n2743, or an update of that paper.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				accidental inconsistencies.		
JP 62	25, 25.3.1.5, 26.3.6.5		te	The return types of is_sorted_until function and is_heap_until function are iterator. But basically, the return type of is_xxx function is bool. And the return type of lower_bound function and upper_bound is iterator. So we think that it is reasonable to change those two functions.	Change "is_sorted_until" to "sorted_bound" Change "is_heap" to "heap_bound"	
UK 296	25.1.8	1	Te	The 'Returns' of adjacent_find requires only HasEqualTo, or a Predicate. Requiring EqualityComparable or EquivalenceRelation seems too strong and not useful.	Change EqualityComparable to HasEqualTo and EquivalnceRelation to Predicate	
UK 297	25.2.11	6	Ed	The definition of rotate_copy is very complicated. It is equivalent to: return copy(first, middle, copy(middle, last, result));	Change 'effects' to, returns, requires, complexity to: effects: equivalent to: return copy(first, middle, copy(middle, last, result));	
UK 298	25.2.13	13	Te	partition_point requires a partitioned array	requires: is_partitioned(first, last, pred) != false;	
UK 299	25.2.2		Ed	Should be consistent in style use of concepts in template parameter lists. The auto-OutputIterator sytle used in std::copy is probably preferred.	Change way signature is declared: template<InputIterator InIter, OutputIterator<auto, RvalueOf<InIter::reference>::type> OutIter> OutIter move(InIter first, InIter last, OutIter result);	
UK 300	25.2.3		Te	Since publishing the original standard, we have learned that swap is a fundamental operation, and several common idioms rely on it - especially those related to exception safety. As such it belongs in the common <utility> header rather than the broader <algorithm> header, and likewise should move to clause 20. For backwards compatiblility the algorithm header should be required to #include <utility>, which would be covered in the resolution of LWG issue 343. There are already dependencies in <algorithm> on types declared in this header, so this comment does not create a new dependency.	Move primary swap template from <algorithm> into <utility>. Move 25.2.3 to somewhere under 20.2. Require <algorithm> to #include <utility> to access pair and provide legacy support for finding swap.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 301	25.2.5		Te	replace and replace_if have the requirement: OutputIterator<Iter, Iter::reference> Which implies they need to copy some values in the range the algorithm is iterating over. This is not however the case, the only thing that happens is const T&s might be copied over existing elements (hence the OutputIterator<Iter, const T&>	Remove OutputIterator<Iter, Iter::reference> from replace and replace_if	
UK 302	25.3	4	Ed	the concept StrictWeakOrder covers the definition of a strict weak ordering, described in paragraph 4.	Remove 4, and mention StrictWeakOrder in paragraph 1.	
UK 303	25.3	6	Ed	This paragraph just describes is_partitioned	6 A sequence [start,finish) is partitioned with respect to an expression f(e) if is_partitioned(start, finish, e) != false	
UK 304	25.3.6		Ed	The requires clauses of push_heap, pop_heap and make_heap are inconsistently formatted, dispite being identical	Format them identically.	
UK 305	25.3.7	1, 9, 17	Te	The negative requirement on IsSameType is a hold-over from an earlier draught with a variadic template form of min/max algorith. It is no longer necessary.	Strike the !IsSameType<T, Compare> constraint on min/max/minmax algorithms	
US 84	26		ge	Parts of the numerics chapter are not concept enabled.		
FR 35	26.3 [Complex numbers]		te	Instantiations of the class template complex<> have to be allowed for integral types, to reflect existing practice and ISO standards (LIA-III).		
UK 306	26.4		Te	Random number library component cannot be used in constrained templates	Provide constraints for the random number library	
JP 63	26.4.8.5.1		te	No constructor of discrete_distribution that accepts initializer_list. discrete_distribution initialize distribution by a given range (iterators), but temporary variable of a container or an array is needed in the following case.	Add the following constructer. discrete_distribution(initializer_list<result_type>);	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				int ar[] = {1, 2, 3}; discrete_distribution<> dist(ar, ar+3); Other libraries also accept initializer_list, so change discrete_distribution library to accept initializer_list too.		
JP 64	26.5.2		te	"valarray<T>& operator+= (initializer_list<T>);" is not defined.	Add valarray<T>& operator+= (initializer_list<T>);	
UK 307	26.7	Footnote 288	Ed	The footnote refers to TR1, which is not a defined term in this standard. Drop the reference to TR1, those templates are a regular part of the standard now and how they were introduced is no longer relevant.	Drop the reference to TR1.	
US 85	27		ge	The input/output chapter is not concept enabled.		
UK 308	27		Te	iostreams library cannot be used from constrained templates	Provide constraints for the iostreams library, clause 27	
JP 65	27.4.4		te	Switch from "unspecified-bool-type" to "explicit operator bool() const".	Replace "operator *unspecified-bool-type*() const;" with "explicit operator bool() const;"	
JP 66	27.4.4.3	1st paragraph	te	Switch from "unspecified-bool-type" to "explicit operator bool() const"	Replace "operator *unspecified-bool-type*() const;" with "explicit operator bool() const;"	
FR 36	27.6.1.2.2 [istream.formatted.arithmetic]	1, 2, and 3	ed	iostate err = 0; iostate is a bitmask type and so could be an enumeration. Probably using goodbit is the solution.		
FR	27.6.1.2.2	3	ed	else if (lval < numeric_limits<int>::min()		

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
37	[istream.formatted.arithmetic]			|| numeric_limits<int>::max() < lval)) The parentheses aren't balanced.		

JP 67	27.7.1		te	basic_stringbuf dose not use concept.	Replace “class Allocator” to “Allocator Alloc”.
					Correct as follows.
					
					namespace std {
					template <class charT, class traits = char_traits<charT>,
					Allocator Alloc = allocator<charT> >
					class basic_stringbuf : public basic_streambuf<charT,traits> {
					public:
					...
					
					// 27.7.1.1 Constructors:
					explicit basic_stringbuf(ios_base::openmode which
					= ios_base::in | ios_base::out);
					explicit basic_stringbuf
					(const basic_string<charT,traits,Alloc>& str,
					ios_base::openmode which = ios_base::in | ios_base::out);
					basic_stringbuf(basic_stringbuf&& rhs);
					
					...
					
					// 27.7.1.3 Get and set:
					basic_string<charT,traits,Alloc> str() const;
					void str(const basic_string<charT,traits,Alloc>& s);
					
					...
					};	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					template <class charT, class traits, Allocator Alloc> void swap(basic_stringbuf<charT, traits, Alloc>& x, basic_stringbuf<charT, traits, Alloc>& y); template <class charT, class traits, Allocator Alloc> void swap(basic_stringbuf<charT, traits, Alloc>&& x, basic_stringbuf<charT, traits, Alloc>& y); template <class charT, class traits, Allocator Alloc> void swap(basic_stringbuf<charT, traits, Alloc>& x, basic_stringbuf<charT, traits, Alloc>&& y); }	

JP 68	27.7.2		te	basic_istringstream dose not use concept.	Replace “class Allocator” to “Allocator Alloc”.
					Correct as follows.
					
					namespace std {
					template <class charT, class traits = char_traits<charT>,
					Allocator Alloc = allocator<charT> >
					class basic_istringstream : public basic_istream<charT,traits> {
					public:
					typedef charT char_type;
					typedef typename traits::int_type int_type;
					typedef typename traits::pos_type pos_type;
					typedef typename traits::off_type off_type;
					typedef traits traits_type;
					typedef Alloc allocator_type;
					
					// 27.7.2.1 Constructors:
					explicit	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					basic_istringstream(ios_base::openmode which = ios_base::in);
					explicit basic_istringstream(
					const basic_string<charT,traits,Alloc>& str,
					ios_base::openmode which = ios_base::in);
					basic_istringstream(basic_istringstream&& rhs);
					
					// 27.7.2.2 Assign and swap:
					basic_istringstream& operator=(basic_istringstream&& rhs);
					void swap(basic_istringstream&& rhs);
					
					// 27.7.2.3 Members:
					basic_stringbuf<charT,traits,Alloc>* rdbuf() const;
					
					basic_string<charT,traits,Alloc> str() const;
					void str(const basic_string<charT,traits,Alloc>& s);
					
					private:
					// basic_stringbuf<charT,traits,Alloc> sb; exposition only
					};
					
					template <class charT, class traits, Allocator Alloc>
					void swap(basic_istringstream<charT, traits, Alloc>& x,
					basic_istringstream<charT, traits, Alloc>& y);
					template <class charT, class traits, Allocator Alloc>
					void swap(basic_istringstream<charT, traits, Alloc>&& x,	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					basic_istringstream<charT, traits, Alloc>& y); template <class charT, class traits, Allocator Alloc> void swap(basic_istringstream<charT, traits, Alloc>& x, basic_istringstream<charT, traits, Alloc>&& y); }	

JP 69	27.7.3		te	basic_ostringstream dose not use concept.	Replace “class Allocator” to “Allocator Alloc”.
					Correct as follows.
					
					namespace std {
					template <class charT, class traits = char_traits<charT>,
					Allocator Alloc = allocator<charT> >
					class basic_ostringstream : public basic_ostream<charT,traits> {
					public:
					// types:
					typedef charT char_type;
					typedef typename traits::int_type int_type;
					typedef typename traits::pos_type pos_type;
					typedef typename traits::off_type off_type;
					typedef traits traits_type;
					typedef Alloc allocator_type;
					
					// 27.7.3.1 Constructors/destructor:
					explicit basic_ostringstream(ios_base::openmode which = ios_base::out);
					explicit basic_ostringstream(
					const basic_string<charT,traits,Alloc>& str,
					ios_base::openmode which = ios_base::out);	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					basic_ostringstream(basic_ostringstream&& rhs);
					
					// 27.7.3.2 Assign/swap:
					basic_ostringstream& operator=(basic_ostringstream&& rhs);
					void swap(basic_ostringstream&& rhs);
					
					// 27.7.3.3 Members:
					basic_stringbuf<charT,traits,Alloc>* rdbuf() const;
					basic_string<charT,traits,Alloc> str() const;
					void str(const basic_string<charT,traits,Alloc>& s);
					private:
					// basic_stringbuf<charT,traits,Alloc> sb; exposition only
					};
					
					template <class charT, class traits, Allocator Alloc>
					void swap(basic_ostringstream<charT, traits, Alloc>& x,
					basic_ostringstream<charT, traits, Alloc>& y);
					template <class charT, class traits, Allocator Alloc>
					void swap(basic_ostringstream<charT, traits, Alloc>&& x,
					basic_ostringstream<charT, traits, Alloc>& y);
					template <class charT, class traits, Allocator Alloc>
					void swap(basic_ostringstream<charT, traits, Alloc>& x,
					basic_ostringstream<charT, traits, Alloc>&& y);	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					}	
JP 71	27.7.3		ed	Typo. "template" is missing in "class basic_ostringstream" of the title of the chapter.	Correct as follows. 27.7.3 Class basic_ostringstream should be 27.7.3 Class template basic_ostringstream	

JP 72	27.7.4		te	basic_stringstream dose not use concept.	Replace "class Allocator" to "Allocator Alloc".
					Correct as follows.
					
					namespace std {
					template <class charT, class traits =
					char_traits<charT>,
					Allocator Alloc = allocator<charT> >
					class basic_stringstream
					: public basic_iostream<charT,traits> {
					public:
					// types:
					typedef charT char_type;
					typedef typename traits::int_type int_type;
					typedef typename traits::pos_type pos_type;
					typedef typename traits::off_type off_type;
					typedef traits traits_type;
					typedef Alloc allocator_type;
					
					// constructors/destructor
					explicit basic_stringstream(
					ios_base::openmode which =
					ios_base::out|ios_base::in);
					explicit basic_stringstream(
					const basic_string<charT,traits,Alloc>& str,
					ios_base::openmode which =
					ios_base::out|ios_base::in);
					basic_stringstream(basic_stringstream&& rhs);	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					// 27.7.5.1 Assign/swap:
					void swap(basic_stringstream&& rhs);
					
					// Members:
					basic_stringbuf<charT,traits,Alloc>* rdbuf() const;
					basic_string<charT,traits,Alloc> str() const;
					void str(const basic_string<charT,traits,Alloc>& str);
					private:
					// basic_stringbuf<charT, traits> sb; exposition only
					};
					
					template <class charT, class traits, Allocator Alloc>
					void swap(basic_stringstream<charT, traits, Alloc>& x,
					basic_stringstream<charT, traits, Alloc>& y);
					template <class charT, class traits, Allocator Alloc>
					void swap(basic_stringstream<charT, traits, Alloc>&& x,
					basic_stringstream<charT, traits, Alloc>& y);
					template <class charT, class traits, Allocator Alloc>
					void swap(basic_stringstream<charT, traits, Alloc>& x,
					basic_stringstream<charT, traits, Alloc>&& y);
					}	

JP 73	27.8.1.14		te	It is a problem from C++98, fstream cannot appoint a filename of wide character string(const wchar_t and const wstring&).	Add interface corresponding to wchar_t, char16_t and char32_t.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
US 86	28		ge	The regular expressions chapter is not concept enabled.		
UK 309	28		Te	Regular expressions cannot be used in constrained templates	Provide constraints for the regular expression library, clause 28	
UK 310	28		Te	The regex chapter uses iterators in the old pre-concept style, it should be changed to use concepts instead.	Use concepts for iterator template arguments throughout.	
UK 314	28.4		Te	The swap overloads for regex classes are only supplied for l-value references. Other sections of the library (eg 21 strings or 23 containers) provide two extra overloads taking an r-value reference as the first and second argument respectively.	Add the missing overloads to 28.4 and the corresponding later sections in 28 for each swap function. We want to accept AMs paper which proposes a single overload with two r-value references	
UK 315	28.4	p6	Te	6 Effects: string_type str(first, last); return use_facet<collate<charT> >(getloc()).transform(&*str.begin(), &*str.end()); Is it legal to dereference str.end() ?	Reword to effect clause to use legal iterator dereferences	
UK 316	28.4 ff		Te	The constructors for regex classes do not have an r-value overload.	Add the missing r-value constructors to regex classes.	
UK 317	28.8		Te	basic_string has both a constructor and an assignment operator that accepts an initializer list, basic_regex should have the same.	In the basic_regex synopsis, after: basic_regex& operator=(const charT* ptr); add: basic_regex& operator=(initializer_list<charT> il); And after paragraph 20 add: basic_regex& operator=(initializer_list<charT> il); Effects: returns assign(il.begin(), il.end());	
JP 74	28.8		te	“basic_regx & operator= (initializer_list<T>);” is not defined.	Add basic_regx & operator= (initializer_list<T>);	
UK 318	28.8.2	para 22	Ed	Constructor definition should appear with the other constructors not after assignment ops.	Move para 22 to just after para 17.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 319	28.12.2		Te	It was always expected that regex_token_iterator would be constructible from an array literal: indeed ideally this is the prefered method of initializing such an object. However, the best we could do in C++0x was: template <std::size_t N> regex_token_iterator(BidirectionalIterator a, BidirectionalIterator b, const regex_type& re, const int (&submatches)[N], regex_constants::match_flag_type m = regex_constants::match_default); Now that we have initializer_lists we should use them to remove this particular wart.	To the synopsis for regex_token_iterator, after template <std::size_t N> regex_token_iterator(BidirectionalIterator a, BidirectionalIterator b, const regex_type& re, const int (&submatches)[N], regex_constants::match_flag_type m = regex_constants::match_default); add regex_token_iterator(BidirectionalIterator a, BidirectionalIterator b, const regex_type& re, initializer_list<int> submatches, regex_constants::match_flag_type m = regex_constants::match_default); In 28.12.2.1 add the declaration: regex_token_iterator(BidirectionalIterator a, BidirectionalIterator b, const regex_type& re, initializer_list<int> submatches, regex_constants::match_flag_type m = regex_constants::match_default); And to the end of para 3 add: The forth constructor initializes the member subs to hold a copy of the sequence of integer values in the range [submatches.begin(), submatches.end()).	
US 87	29		ge	The atomics chapter is not concept enabled. The adopted paper, N2427, did have those concepts.		
UK 311	29		Te	Atomic types cannot be used generically in a constrained template	Provide constraints for the atomics library, clause 29	
UK 312	29		Te	The contents of the <stdatomic.h> header are not listed anywhere, and <cstdatomic> is listed as a C99 header in chapter 17. If we intend to use these for compatibility with a future C standard, we should not use them now.	Remove <cstdatomic> from the C99 headers in table 14. Add a new header <atomic> to the headers in table 13. Update chapter 29 to remove reference to <stdatomic.h> and replace the use of <cstdatomic> with <atomic>. If and when WG14 adds atomic operations to C we can add corresponding headers to table 14 with a TR.	
JP	29		ed	A definition of enum or struct is the style of C using	Change to a style of C++.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

75				typedef.	Correct as follows.
					
					29.1
					
					namespace std {
					typedef enum memory_order {
					memory_order_relaxed,
					memory_order_consume, memory_order_acquire,
					memory_order_release,
					memory_order_acq_rel, memory_order_seq_cst
					} memory_order;
					}
					
					should be
					
					namespace std {
					enum memory_order {
					memory_order_relaxed,
					memory_order_consume, memory_order_acquire,
					memory_order_release,
					memory_order_acq_rel, memory_order_seq_cst
					};
					}
					
					29.3.1
					
					namespace std {
					typedef struct atomic_bool {
					...
					} atomic_bool;
					}
					
					should be
					
					namespace std {
					struct atomic_bool {
					...	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					};
					}
					
					namespace std {
					typedef struct atomic_*itype* {
					...
					} atomic_*itype*;
					}
					
					should be
					
					namespace std {
					struct atomic_*itype* {
					...
					};
					}
					
					29.3.2
					
					namespace std {
					typedef struct atomic_address {
					...
					} atomic_address;
					}
					
					should be
					
					namespace std {
					struct atomic_address {
					...
					};
					}
					
					29.5
					
					namespace std {
					typedef struct atomic_flag {	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					...
					} atomic_flag;
					}
					
					should be
					
					namespace std {
					struct atomic_flag {
					...
					};
					}	

UK 313	29.1		Te	seq_cst fences don't necessarily guarantee ordering http://home.twcny.rr.com/hinnant/cpp_extensions/issues_preview/lwg-active.html#926	Add a new paragraph after 29.1 [atomics.order]p5 that says For atomic operations A and B on an atomic object M, where A and B modify M, if there are memory_order_seq_cst fences X and Y such that A is sequenced before X, Y is sequenced before B, and X precedes Y in S, then B occurs later than A in the modifiction order of M.	
US 88	29.2		te	The "lockfree" facilities do not tell the programmer enough.	Expand the "lockfree" facilities. See the attached paper "Issues with the C++ Standard" under Chapter 29, "atomics.lockfree doesn't tell the programmer enough"	
US 89	29.3.1	Table 122	te	The types in the table "Atomics for standard typedef types" should be typedefs, not classes. These semantics are necessary for compatibility with C.	Change the classes to typedefs.	Google
US 90	29.4		te	Are atomic functions allowed to have non-volatile overloads?	Allow non-volatile overloads. See the attached paper "Issues with the C++ Standard, under Chapter 29, "Are atomic functions allowed to have non-volatile overloads?"	
US 91	29.4		te	Whether or not a failed compare_exchange is a RMW operation (as used in 1.10 [intro.multithread]) is unclear.	Make failing compare_exchange operations **not** be RMW. See the attached paper under "atomic RMW status of failed compare_exchange"	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
US 92	29.4		te	The effect of memory_order_consume with atomic RMW operations is unclear.	Follow the lead of fences [atomics.fences], and promote memory_order_consume to memory_order_acquire with RMW operations.	
JP 76	30		ed	A description for "*Throws:* Nothing." are not unified. At the part without throw, "*Throws:* Nothing." should be described.	Add "*Throws:* Nothing." to the following. 30.2.1.6 , 1st paragraph 30.3.3.1 , 4th paragraph 30.3.3.2.1 , 6th paragraph 30.4.1 , 7th and 8th paragraph 30.4.2 , 6th, 7th,19th,21th and 25th paragraph	
US 93	30		ge	The thread chapter is not concept enabled.		
UK 320	30		Te	Threads library cannot be used in constrained templates	Provide constraints for the threads library, clause 30	
UK 321	30		Ed	Throughout this clause, the term Requires: is used to introduce compile time requirements, which we expect to be replaced with concepts and requires in code. Run-time preconditions are introduced with the term "Preconditions:" which is not a defined part of the library documentation structure (17.5.2.4). However, this is exactly the direction that BSI would like to see the organisation move, replacing Requires: clauses with Preconditions: clasues throught the library. See comment against clause 17 for more details.	Decument Preconditions: paragraphs in 17.5.2.4, and use consistently through rest of the library.	
US 94	30.1.2	1	te	The first sentence of para 1 suggests that no other library function is permitted to call operating system or low level APIs.	Rewrite para 1 as: " Some functions described in this Clause are specified to throw exceptions of type system_error (19.4.5). Such exceptions shall be thrown if a call to an operating system or underlying API results in an error that prevents the library function from satisfying its postconditions or from returning a meaningful value."	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
US 95	30.1.3	1	te	"native_handle_type" is a typedef, not a class member.	Several classes described in this Clause have a member native_handle (of type native_handle_type) . The presence of this member and its semantics is implementation defined. [Note: This member allows implementations to provide access to implementation details. The name of the member and the type are specified to facilitate portable compile-time detection. Actual use of this member or the corresponding type is inherently non-portable. —end note]	
US 96	30.1.4	2	te	There is no definition here for monotonic clock.	Implementations should use a *monotonic clock* to measure time for these functions. A monotonic clock measures real time, but cannot be set, and is guaranteed to have no negative clock jumps.	
UK 322	30.1.4	2	Te	Not all systms can provide a monotonic clock. How are they expected to treat a _for function?	Add at least a note explaining the intent for systems that do not support a monotonic clock.	
UK 323	30.2.1	1	Te	The presence of a non-explicit variadic template constructor alongside an explicit single-argument constructor can lead to behaviour that is not intended: the variadic constructor will be selected for implicit conversions, defeating the purpose of the explicit single-argument constructor. Additionally the single-argument constructor is redundant; the variadic constructor can provide identical functionality with one *fewer* copies if the supplied func is an lvalue.	Mark constructor template <class F, class ...Args> thread(F&& f, Args&&... args); as explicit and remove the single-argument constructor.	
UK 324	30.2.1.1		Te	thread::id objects should be as useable in hashing containers as they are in ordered associative containers.	Add thread::id support for std::hash	
JP	30.2.1.2		te	"CopyConstructible" and "MoveConstructible" in "*Requires:* F and each Ti in Args shall be	Add a concept for constructor of thread.	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
77				CopyConstructible if an lvalue and otherwise MoveConstructible." are reflected by interface.		
JP 78	30.2.1.2	4th paragraph, 1st line	ed	In "F and each Ti in Args", 'Ti' is not clear.	Replace "Ti" with "args"	
US 97	30.2.1.3	1	te	detach-on-destruction may result in “escaped” threads accessing objects with bounded lifetime after the end of their lifetime.	See document WG21 N2802=08-0312 written by Hans Boehm.	
US 98	30.2.1.3, 30.2.1.4			The current defined behavior for the std::thread destructor is to detach the thread. Unfortunately, this behavior exposes programmers to tricky, hard-to-diagnose, undefined behavior.	Change destruction behavior to undefined behavior, with a note strongly encouraging termination. See the attached paper "Issues with the C++ Standard" under Chapter 30, "Implicit thread detach is harmful".	
UK 325	30.3.3	2	Te	We believe constexpr literal values should be a more natural expression of empty tag types than extern objects as it should improve the compilers ability to optimize the empty object away completely.	Replace the extern declarations: extern const defer_lock_t defer_lock; extern const try_to_lock_t try_to_lock; extern const adopt_lock_t adopt_lock; with constexpr values constexpr defer_lock_t defer_lock{}; constexpr try_to_lock_t try_to_lock{}; constexpr adopt_lock_t adopt_lock{};	
UK 326	30.3.3.2.1	7	Te	The precondition that the mutex is not owned by this thread offers introduces the risk of un-necessary undefined behaviour into the program. The only time it matters whether the current thread owns the mutex is in the lock operation, and that will happen subsequent to construction in this case. The lock operation has the identical pre-condition, so there is nothing gained by asserting that precondition earlier and denying the program the right to get into a valid state before calling lock.	Strike 30.3.3.2.1p7	
UK 327	30.3.3.2.2	4, 9, 14, 19	Te	Not clear what the specification for error condition resource_deadlock_would_occur means. It is perfectly	Add a precondition !owns. Change the 'i.e.' in the error condition to be 'e.g.' to allow for this	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				possible for this thread to own the mutex without setting owns to true on this specific lock object. It is also possible for lock operations to succeed even if the thread does own the mutex, if the mutex is recursive. Likewise, if the mutex is not recursive and the mutex has been locked externally, it is not always possible to know that this error condition should be raised, depending on the host operating system facilities. It is possible that 'i.e.' was supposed to be 'e.g.' and that suggests that recursive locks are not allowed. That makes sense, as the exposition-only member owns is boolean and not a integer to count recursive locks.	condition to propogate deadlock detection by the host OS.	
UK 328	30.3.3.2.2	20	Te	There is a missing precondition that owns is true, or an if(owns) test is missing from the effect clause	Add a precondition that owns == true. Add an error condition to detect a violation, rather than yield undefined behaviour.	
UK 329	30.5		Te	future, promise and packaged_task provide a framework for creating future values, but a simple function to tie all three components together is missing. Note that we only need a *simple* facility for C++0x. Advanced thread pools are to be left for TR2.	Provide a simple function along the lines of: template< typename F, typename ... Args > requires Callable< F, Args... > future< Callable::result_type > async(F&& f, Args && ...); Semantics are similar to creating a thread object with a packaged_task invoking f with forward<Args>(args...) but details are left unspecified to allow different scheduling and thread spawning implementations. It is unspecified whether a task submitted to async is run on its own thread or a thread previously used for another async task. If a call to async succeeds, it shall be safe to wait for it from any thread. The state of thread_local variables shall be preserved during async calls. No two incomplete async tasks shall see the same value of this_thread::get_id(). [Note: this effectively forces new tasks to be run on a new thread, or a fixed-size pool with no queue. If the library is unable to spawn a new thread or there are no free worker threads then the async	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
					call should fail.]	
UK 330	30.5.1		Ed	30.5.1 (and then 30.5.7) refer to a specialisation of constructible_with_allocator_prefix<> However this trait is not in the CD, so references to it should be removed.	Remove the reference to constructible_with_allocator_prefix in 30.5.1 Remove paragraph 30.5.7	
JP 79	30.5.1		te	The concept of UsesAllocator and Allocator should be used.	Correct as follows. template <class R, class Alloc> struct uses_allocator<promise<R>, Alloc>; template <class R> struct constructible_with_allocator_prefix<promise<R> >; should be template<class R, Allocator Alloc> concept_map UsesAllocator<promise<R>, Alloc>;	
UK 331	30.5.3		Te	Not clear what it means for a public constructor to be 'exposition only'. If the intent is purely to support the library calling this constructor then it can be made private and accessed through friendship. Otherwise it should be documented for public consumption.	Declare the constructor as private with a note about intended friendship, or remove the exposition-only comment and document the semantics.	
UK 332	30.5.4		Ed	It is not clear without reference to the original proposal how to use a future. In particular, the only way for the user to construct a future is via the promise API which is documented after the presentation of future. Most library clauses feature a small description of their components and intended use, it would be most useful in this case.	Provide a small introductory paragraph, docuenting intended purpose of the future class template and the way futures can only be created via the promise API.	
UK 333	30.5.4	5	Ge	We expect the complicated 3-signature specifcation for future::get() to be simplified to a single signature with a requires clause by the application of concepts.	Requires fully baked concepts for clause 30	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
UK 334	30.5.4	5	Te	Behaviour of get() is undefined if calling get() while not is_ready(). The intent is that get() is a blocking call, and will wait for the future to become ready.	Effects: If is_ready() would return false, block on the asynchronous result associated with *this.	
UK 335	30.5.4		Te	std::unique_future is MoveConstructible, so you can transfer the association with an asynchronous result from one instance to another. However, there is no way to determine whether or not an instance has been moved from, and therefore whether or not it is safe to wait for it. std::promise<int> p; std::unique_future<int> uf(p.get_future()); std::unique_future<int> uf2(std::move(uf)); uf.wait(); // oops, uf has no result to wait for.	Add a "waitable()" function to unique_future (and shared_future) akin to std::thread::joinable(), which returns true if there is an associated result to wait for (whether or not it is ready). Then we can say: if(uf.waitable()) uf.wait();	
UK 336	30.5.4		Te	It is possible to transfer ownership of the asynchronous result from one unique_future instance to another via the move-constructor. However, it is not possible to transfer it back, and nor is it possible to create a default-constructed unique_future instance to use as a later move target. This unduly limits the use of unique_future in code. Also, the lack of a move-assignment operator restricts the use of unique_future in containers such as std::vector - vector::insert requires move-assignable for example.	Add a default constructor with the semantics that it creates a unique_future with no associated asynchronous result. Add a move-assignment operator which transfers ownership.	
JP 80	30.5.4 , 30.5.5		ed	Typo, duplicated ">" "class Period>>"	Remove one	
UK 337	30.5.5		Te	shared_future should support an efficient move constructor that can avoid unnecessary manipulation of a reference count, much like shared_ptr	Add a move constructor	
UK 338	30.5.5		Te	shared_future is currently CopyConstructible, but not CopyAssignable. This is inconsistent with shared_ptr, and will surprise users. Users will then write work-arounds to provide this behaviour. We should provide it simply and efficiently as part of shared_future. Note that since the	Remove "=delete" from the copy-assignment operator of shared_future. Add a move-constructor shared_future(shared_future&& rhs), and a move-assignment operator shared_future& operator=(shared_future&& rhs). The	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				shared_future member functions for accessing the state are all declared const, the original usage of an immutable shared_future value that can be freely copied by multiple threads can be retained by declaring such an instance as "const shared_future".	postcondition for the copy-assignment operator is that *this has the same associated state as rhs. The postcondition for the move-constructor and move assignment is that *this has the same associated as rhs had before the constructor/assignment call and that rhs has no associated state.	
UK 339	30.5.6	6, 7	Te	Move assignment is goiing in the wrong direction, assigning from *this to the passed rvalue, and then returning a reference to an unusable *this	Strike 6. 7 Postcondition: associated state of *this is the same as the associated state of rhs before the call. rhs has no associated state.	
UK 340	30.5.6	11, 12, 13	Te	There is an implied postcondition that the state of the promise is transferred into the future leaving the promise with no associated state. It should be spelled out.	Postcondition: *this has no associated state.	
UK 341	30.5.6		Te	promise::swap accepts its parameter by lvalue reference. This is inconsistent with other types that provide a swap member function, where those swap functions accept an rvalue reference	Change promise::swap to take an rvalue reference.	
UK 342	30.5.6		Te	std::promise is missing a non-member overload of swap. This is inconsistent with other types that provide a swap member function	Add a non-member overload void swap(promise&& x,promise&& y){ x.swap(y); }	
UK 343	30.5.6	3	Te	The move constructor of a std::promise object does not need to allocate any memory, so the move-construct-with-allocator overload of the constructor is superfluous.	Remove the constructor with the signature template <class Allocator> promise(allocator_arg_t, const Allocator& a, promise& rhs);	

JP 81	30.5.8		ed	There are not requirements for F and a concept of Allocator dose not use.	Correct as follows.
					
					template <class F>
					explicit packaged_task(F f);
					template <class F, class Allocator>
					explicit packaged_task(allocator_arg_t, const Allocator& a, F f);
					template <class F>	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**

					explicit packaged_task(F&& f);
					template <class F, class Allocator>
					explicit packaged_task(allocator_arg_t, const Allocator& a, F&& f);
					
					should be
					
					template <class F>
					requires CopyConstructible<F> && Callable<F, ArgTypes...>
					&& Convertible<Callable<F, ArgTypes...>::result_type, R>
					explicit packaged_task(F f);
					
					template <class F, Allocator Alloc>
					requires CopyConstructible<F> && Callable<F, ArgTypes...>
					&& Convertible<Callable<F, ArgTypes...>::result_type, R>
					explicit packaged_task(allocator_arg_t, const Alloc& a, F f);
					
					template <class F>
					requires CopyConstructible<F> && Callable<F, ArgTypes...>
					&& Convertible<Callable<F, ArgTypes...>::result_type, R>
					explicit packaged_task(F&& f);
					
					template <class F, Allocator Alloc>
					requires CopyConstructible<F> && Callable<F, ArgTypes...>
					&& Convertible<Callable<F, ArgTypes...>::result_type, R>
					explicit packaged_task(allocator_arg_t, const Alloc& a, F&& f);	

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
DE-23	Annex B	p2	te	DE-23 Recursive use of constexpr functions appears to be permitted. Since such functions may be required to be evaluated at compile-time, Annex B "implementation quantities" should specify a maximum depth of recursion.	In Annex B, specify a recursion depth of 256 or a larger value.	
DE-24	Annex B	p2	te	DE-24 The number of placeholders for "bind" is implementation-defined in 20.7.12.1.4, but no minimum is suggested in Annex B.	Add a miminum of 10 placeholders to Annex B.	
DE-25	Annex B	p2	te	DE-25 Specifying a minimum of 17 recursively nested template instantiations is too small for practical purposes. The limit is too high to effectively limit compiler resource usage, see http://ubiety.uwaterloo.ca/~tveldhui/papers/2003/turing.pdf . The conclusion is that the metric "number of recursively nested template instantiations" is inapposite.	Remove the bullet "Recursively nested template instantiations [17]".	
FR 38	C.2 [diffs.library]	1	ed	What is ISO/IEC 1990:9899/DAM 1? My guess is that's a typo for ISO/IEC 9899/Amd.1:1995 which I'd have expected to be referenced here (the tables make reference to things which were introduced by Amd.1).	One need probably a reference to the document which introduce char16_t and char32_t in C (ISO/IEC TR 19769:2004?).	
UK 344	Appendix D		Ge	It is desirable to allow some mechanism to support phasing out of deprecated features in the future. Allowing compilers to implement a mode where deprecated features are not available is a good first step.	Add to the definition of deprecated features permission for compilers to maintain a conditionally supported mode where deprecated features can be disabled, so long as they also default to a mode where all deprecated features are supported.	
FR 39	Index		ed	Some definitions seem not indexed (such as /trivially copyable/ or		

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	3	4	5	6	7
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Disposition**
				/sequenced before/), indexing them would be useful (and marking specially the page -- say bold or italic -- which reference to the definition would increase the usefulness; having a separate index of all definitions is something which could also be considered).		

1 **MB** = Member body (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

NOTE Columns 1, 2, 4, 5 are compulsory.

Additional Details to USNB Comments

US-26: Use of objects, especially from other threads, during destruction

Author
Jeffrey Yasskin <jyasskin@google.com>
Section
3.8 [basic.life]

The current draft says:

[3.8-1] The lifetime of an object of type T ends when ... the destructor call starts.

[3.8-5] ... after the lifetime of an object has ended and before the storage which the object occupied is reused or released, any pointer that refers to the storage location where the object will be or was located may be used but only in limited ways. ... The program has undefined behavior if:
...
- the pointer is used to access a non-static data member or call a non-static member function of the object, or
...

This prohibits use of the object's fields from within its own destructor, conflicting with the specification in section 12.7 [class.cdtor]. It also prohibits use of the object from other threads during destruction, which isn't addressed anywhere. Thread pools form a concrete use case:

A ThreadPool? class looks something like:

```
struct ThreadPool {
 ThreadPool(int num_threads) {
   for (int i = 0; i < num_threads; ++i) {
     threads.push_back(std::thread(&ThreadPool::Worker, this));
   }
 }
 ~ThreadPool() {
   this->shutting_down = true;
   for (std::thread& thread : this->threads) {
     this->threadsafe_queue.push(&DoNothing);
   }
   for (std::thread& thread : this->threads) {
     thread.join();
   }
 }
 void execute(function<void()> task) {
   this->threadsafe_queue.push(task);
 }
 void Worker();
};
```

Additional Details to USNB Comments

US-26 (cont'd)

I'd like to define Worker() as:

```
void ThreadPool::Worker() {
 while (true) {
   function<void()> next_task = this->threadsafe_queue.pop());
   if (this->shutting_down) break;
   next_task();
 }
}
```

But this can easily access threadsafe_queue and shutting_down after the destructor starts. Importantly, however, because of the thread.join()s in ~ThreadPool, it can't access them after the destructor finishes. Copying the wording from 12.7, "For an object with a non-trivial destructor, referring to any non-static member or base class of the object after the destructor finishes execution results in undefined behavior.", to 3.8 would mostly solve this. Of course, class invariants may no longer hold after the start of the destructor.

It's a little ambiguous when "the destructor" finishes, since a single object's destruction may involve running several destructors. I think it makes sense to choose the destructor for the static type of the pointer used to access the member, although Lawrence Crowl suggests the destructor for the fully-derived type.

Now we have another wrinkle:

First, note that the following code is defined (assuming the modification above):

```
{
  ThreadPool pool(1);
  pool.execute([&]() {
    DoSomeStuff();
    pool.execute(&AnotherTask);
  });
}
```

Since the second pool.execute() call must run before the pool's threads are joined, it definitely runs before the destructor finishes.

Now say that we want to define a generic Executor class and make ThreadPool? a subclass of it:

```
struct Executor {
  virtual void execute(function<void()> task) = 0;
};
struct ThreadPool : public Executor { ... };
```

US-26 (cont'd)

Then we might want to subclass ThreadPool? again to record more information:

```
struct RecordingThreadPool : public ThreadPool {
 void execute(function<void()> task) {
   RecordStuff();
   ThreadPool::execute(task);
 }
};
```

And we run the following:

```
{
  RecordingThreadPool pool(1);
  pool.execute([&]() {
    DoSomeStuff();
    pool.execute(&AnotherTask);
  });
}
```

Which class's execute() definition does the second pool.execute() call run? There's no happens-before relation constraining it, so it's either unspecified or undefined. Even if it's just unspecified, RecordingThreadPool?::execute() would have to be really careful about accessing its own members since ~RecordingThreadPool() is can finish inside the execute() call. So I'd say it's undefined.

But not all subclasses have this problem:

```
struct DerivedThreadPool : public ThreadPool {
  // Doesn't override execute().
};

{
  DerivedThreadPool pool(1);
  pool.execute([&]() {
    DoSomeStuff();
    pool.execute(&AnotherTask);
  });
}
```

Since this always resolves to ThreadPool?::execute(), we could define that as the behavior.

Additional Details to USNB Comments

US-42: Meaning of `[[final]]` attribute applied to classes

Author
Jeffrey Yasskin <jyasskin@google.com>
Section
7.6.4 [dcl.attr.final]

7.6.4 [dcl.attr.final] says,

"If the attribute is specified for a class definition, it is equivalent to being specified for each virtual member function of that class, including inherited member functions."

This means that the following program is well-formed:

```
struct Base [[final]] { /* No virtual methods */ };
struct Derived : Base { /* Anything */ };
```

There are two problems with this:

1) This is different from Java's `final` keyword, which will surprise people who first encountered the keyword in Java.

2) One might want to prohibit deriving from a class without a virtual destructor to avoid that route to undefined behavior. But without a virtual destructor, applying `[[final]]` to a class doesn't prohibit deriving from that class, removing one of the major use cases.

Also, N2761 described the attribute with "A class with the `final` attribute will not be allowed to be a base class for another class.", which is not what the text specifies.

To make the draft standard match Java and the rough description in N2761, I propose:

1) Strike the sentence "If the attribute is specified for a class definition, it is equivalent to being specified for each virtual member function of that class, including inherited member functions."

2) Add a third paragraph:

"If a class B is marked `final` and a class D is derived from B, the program is ill-formed; no diagnostic required. [Example:

```
struct B2 {};
struct D2 : B2 {}; // ill-formed
```

-- end example]"

Additional Details to USNB Comments

US-42 (cont'd)

3) Perhaps add "[Note: Because a `final` class cannot be derived from, all of its virtual member functions behave as if they were also marked `final`. -- end note]" if that's not clear enough from the other definitions.

US-49 8.5.4/6 Editorial

8.5.4/6 [dcl.init.list] In the Example, the comments could be improved:

- char c3{y}; // error: narrows (assuming char is 8 bits)
  - I have worked with DSPs where sizeof(int) == sizeof(char) and this would not be an error.
- float f1 { x }; // error: might narrow
  - for consistency with comment on c2 declaration
  - presumably float(999) doesn't typically lose precision.

US-78: Conversion from `shared_ptr` to `unique_ptr`

Author
: Pardo <pardo@google.com>

Section
: 20.8.12 [unique.ptr] 20.8.13.2 [util.smartptr.shared]

There is presently no way to convert directly from a `shared_ptr` to a `unique_ptr`. It may be desirable to do so, as example so a class or module can use `unique_ptr` to return a value which was computed internally using `shared_ptr`, but which at return is known to be unique. C++ presently supports indirect conversion by extracting the raw pointer and checking the reference count of the `shared_ptr` is one.

To make such usage cleaner, I propose adding an interface that performs conversion. Behavior when the reference count is not one could take several forms, including leaving the behavior undefined, or throwing an exception -- though throwing may be problematic in constructors. Throwing an exception seems most natural.

Comment from James Dennett <jdennett@google.com>

It is currently not possible to take the pointer away from a shared_ptr; it's an invariant of shared_ptr that the last shared_ptr to an object **will** call the deleter on the owned pointer/object when the shared_ptr is destroyed. The way around that, such as it is, is to use a custom deleter which can be told to act as a no-op. It might be hard to persuade the committee to allow a way to "steal" from a shared_ptr, i.e., there's no way to tell the shared_ptr to release() its owned object, even if you know that the shared_ptr is unique.

Additional Details to USNB Comments

US-88: atomics.lockfree doesn't tell the programmer enough

Author

Jeffrey Yasskin <jyasskin@google.com>

Section

29.2 [atomics.lockfree]

There are 2 problems here. First, at least on x86, it's less important to me whether some integral types are lock free than what is the largest type I can pass to atomic and have it be lock-free. For example, if long longs are not lock-free, ATOMIC_INTEGRAL_LOCK_FREE is probably 1, but I'd still be interested in knowing whether longs are always lock-free. Or if long longs at any address are lock-free, I'd expect ATOMIC_INTEGRAL_LOCK_FREE to be 2, but I may actually care whether I have access to the `cmpxchg16b` instruction. None of the support here helps with that question. (There are really 2 related questions here: what alignment requirements are there for lock-free access; and what processor is the program actually running on, as opposed to what it was compiled for?)

Second, having atomic_is_lock_free only apply to individual objects is pretty useless (except, as Lawrence Crowl points out, for throwing an exception when an object is unexpectedly not lock-free). I'm likely to want to use its result to decide what algorithm to use, and that algorithm is probably going to allocate new memory containing atomic objects and then try to act on them. If I can't predict the lock-freedom of the new object by checking the lock-freedom of an existing object, I may discover after starting the algorithm that I can't continue.

To solve the first problem, I think 2 macros would help: MAX_POSSIBLE_LOCK_FREE_SIZE and MAX_GUARANTEED_LOCK_FREE_SIZE, which expand to the maximum value of sizeof(T) for which atomic may (or will, respectively) use lock-free operations. Lawrence points out that this "relies heavily on implementations using word-size compare-swap on sub-word-size types, which in turn requires address modulation." He expects that to be the end state anyway, so it doesn't bother him much.

To solve the second, I think one could specify that equally aligned objects of the same type will return the same value from atomic_is_lock_free(). I don't know how to specify "equal alignment". Lawrence suggests an additional function, atomic_is_always_lock_free().

Additional Details to USNB Comments

US-90: Are atomic functions allowed to have non-volatile overloads?

Author

Jeffrey Yasskin <jyasskin@google.com>

Section

29.4 [atomics.types.operations]

The C++0X draft declares all of the functions dealing with atomics (section 29.3) to take volatile arguments. Yet it also says (29.4-3),

[Note: Many operations are volatile-qualified. The "volatile as device register" semantics have not changed in the standard. This qualification means that volatility is preserved when applying these operations to volatile objects. It does not mean that operations on non-volatile objects become volatile. Thus, volatile qualified operations on non-volatile objects may be merged under some conditions. —end note]

I was thinking about how to implement this in gcc, and I believe that we'll want to overload most of the functions on volatile and non-volatile. Here's why:

To let the compiler take advantage of the permission to merge non-volatile atomic operations and reorder atomics in certain, we'll need to tell the compiler backend about exactly which atomic operation was used. So I expect most of the function of the form atomic_<op>_explicit() (e.g. atomic_load_explicit, atomic_exchange_explicit, atomic_fetch_add_explicit, etc.) to become compiler builtins. A builtin can tell whether its argument was volatile or not, so those functions don't really need extra explicit overloads. However, I don't expect that we'll want to add builtins for every function in chapter 29,

since most can be implemented in terms of the _explicit free functions:

```
class atomic_int {
  __atomic_int_storage value;
 public:
  int fetch_add(int increment, memory_order order =
memory_order_seq_cst) volatile {
    // &value has type "volatile __atomic_int_storage*".
    atomic_fetch_add_explicit(&value, increment, order);
  }
  ...
};
```

Additional Details to USNB Comments

US-90 (cont'd)

But now this *always* calls the volatile builtin version of atomic_fetch_add_explicit(), even if the atomic_int wasn't declared volatile. To preserve volatility and the compiler's permission to optimize, I'd need to write:

```
class atomic_int {
  __atomic_int_storage value;
 public:
  int fetch_add(int increment, memory_order order =
memory_order_seq_cst) volatile {
    atomic_fetch_add_explicit(&value, increment, order);
  }
  int fetch_add(int increment, memory_order order =
memory_order_seq_cst) {
    atomic_fetch_add_explicit(&value, increment, order);
  }
  ...
};
```

But this is visibly different from the declarations in the standard because it's now overloaded. (Consider passing &atomic_int::fetch_add as a template parameter.)

The implementation may already have permission to add overloads to the member functions:

[17.6.5.5-2] An implementation may declare additional non-virtual member function signatures within a class:
...

- by adding a member function signature for a member function name.

but I don't see an equivalent permission to add overloads to the free functions.

US-91: atomic RMW status of failed compare_exchange

Author
: Lawrence Crowl <crowl@google.com>

Section
: 29.4 [atomics.types.operations]

Whether or not a failed compare_exchange is a RMW operation (as used in 1.10 [intro.multithread]) is unclear.

The proposed resolution is to make failing compare_exchange operations **not** be RMW.

Additional Details to USNB Comments

Author
Anthony Williams <anthony.ajw@gmail.com>
Section
29.4 [atomics.types.operations]

In 29.4p18 it says that "These operations are atomic read-modify-write operations" (final sentence). This is overly restrictive on the implementations of compare_exchange_weak and compare_exchange_strong on platforms without a native CAS instruction.

Proposed resolution:

Replace that sentence with "If the comparison is true, these operations are atomic read-modify-write operations (1.10). If the comparison is false, these operations are atomic load operations."

US-98: Implicit thread detach is harmful

Author
Lawrence Crowl <crowl@google.com>
Section
30.2.1.3 [thread.thread.destr], 30.2.1.4 [thread.thread.assign]

The current defined behavior for the std::thread destructor is to detach the thread.

~std::thread() { if (joinable()) detach(); }

Unfortunately, this behavior exposes programmers to tricky, hard-to-diagnose, undefined behavior.

The problem is that many threads will be created with references to the creating thread's call stack. If the creating thread encounters an exception, it will detach all threads it owns, leaving them with dangling references to memory.

The propose resolution is to change destruction behavior to undefined behavior, with a note strongly encouraging:

~std::thread() { if (joinable()) terminate(); }

The reason for undefined behavior now is to leave the standard room to change its mind later. The reason to encourage terminate is to diagnose problem early.

The same reasoning applies to a thread over-written by assignment.