Document No:	WG21 N4146
Date:	2014-09-23
References:	ISO/IEC DIS 14882 (Ed 4)
Reply To:	Barry Hedquist <beh@peren.com>
	INCITS/PL22.16 IR

Disposition of Comments

ISO/IEC DIS 14882

C++ 2014

Attached is WG21 N4146, the Disposition of Comments for ISO/IEC DIS 14882, Draft Information Standard ISO/IEC 14882:2014.

Document numbers referenced in the ballot comments are WG21 documents unless otherwise stated.

Template for comments and secretariat observations

Date: 2014-08-12	Document: **ISO/IEC DIS 14882 (Ed4)**	Project: 64029

MB/NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
JP 01				ed	This DIS does not have Foreword. It should be given before the first clause.		To be fixed by ISO secretariat
JP 02		1.2	Paragraph 1	ed	ISO/IEC 2382 - Vocabulary is listed in 1.2 Normative References, but it should be moved to Bibliography. 2382 is not referred to in normative part of this DIS.		This is consistent with all prior revisions of C++. Terms defined in this document are used throughout the International Standard.
JP 03		2.14.2	Paragraph 0 (Syntax)	ge	Current integer literal syntax allows single quotation mark (') as a digit separator. In octal integer literal, single quotation mark after prefix is allowed. In other notations, single quotation mark after prefix is not allowed. (e.g. 0'01 is well-formed, but 0b'01 and 0x'01 are ill-formed.) We think this asymmetry makes tools such as automatic code generator complicated.	Allow digit separator after binary and hexadecimal prefix, too. Or, Disallow it after octal prefix.	<not editorial> Assigned as Core Issue 1947, for consideration in a future revision to the Standard. Out-of-scope for this revision.
JP 04		2.14.2	Table 6	ed	In Table 6, at the header of column 3, the order of type names of integer literals is not consistent, where "Binary" is placed before "octal" and "hexadecimal", but at the other places, the order is always decimal, octal, hexadecimal and then binary, as in the syntax section.	Change to "Octal, hexadecimal, or binary literal"	These lists should consistently be in radix order, that is, "binary, octal, decimal, hexadecimal". This will be fixed in a future revision of the C++ standard.

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

Template for comments and secretariat observations

Date: 2014-08-12	Document: **ISO/IEC DIS 14882 (Ed4)**	Project: 64029

MB/NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
JP 05		5.3.4	Paragraph 10	ed	> The implementation may extend the allocation of a new-expression e1 > to provide storage for a new-expression e2 if the following would > be true were the allocation not extended: Is this sentence correct syntax? We are not sure whether the list of the descriptions is "positive" condition or "negative" condition. "Positive" means that if the condition is satisfied, then the allocation may extend. "Negative" means that if the condition is satisfied, then the allocation does not extend. For example, the 4th description seems to be "positive". But the description above the list of conditions says that "if the following would be true were the allocation *not* extended"	Make it correct and easy-to-understand description. Purely grammatically, "if the following would be true were the allocation not extended" seems to lack "is" before "not".	Not a defect: the sentence is establishing a hypothetical, and is grammatically correct. The bulleted items are positive statements within that hypothetical.

JP 06		7.1.5	Paragraph 8	ed	By N3652, the statement "The class of which .... a literal type (3.9)" remains, but it is removed in DIS. Please check whether the removal is intended or not.	If it is intended, we think that the following part of the example should be removed because it is not related to this clause in DIS.
						
						`class debug_flag {`
						`public:`
						`explicit debug_flag(bool);`
						`constexpr bool is_on() const; // error: debug_flag not`
						`// literal type`
						`private:`
						`bool flag;`
						`};`
							The removal is correct; the sentence was removed by CWG issue 1684.
							The example is removed in IS.

JP 07		7.1.6.4	Paragraph 7	ge	In the sentence, "When a variable…type of its specifier", we think specification for function return type deduction is unclear, because the term "initializer" is not directly related to return statement.	Change to: (e.g.) *the deduced return type is determined from the type of expression specified for the return statement.*	This warrants further investigation, but will not be changed for the IS. This is editorial issue #371.

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

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**

JP 08		20.9.5	Paragraph 7 to 12	ed	All auto operator() lacks "constexpr".	7 operator() returns x <= y.
						template <> struct equal_to<void> {
						template <class T, class U> **constexpr** auto operator()(T&& t, U&& u) const
						-> decltype(std::forward<T>(t) == std::forward<U>(u));
						typedef unspecified is_transparent;
						};
						8 operator() returns std::forward<T>(t) == std::forward<U>(u).
						template <> struct not_equal_to<void> {
						template <class T, class U> **constexpr** auto operator()(T&& t, U&& u) const
						-> decltype(std::forward<T>(t) != std::forward<U>(u));
						typedef unspecified is_transparent;
						};
						9 operator() returns std::forward<T>(t) != std::forward<U>(u).
						template <> struct greater<void> {
						template <class T, class U> **constexpr** auto operator()(T&& t, U&& u) const
						-> decltype(std::forward<T>(t) > std::forward<U>(u));
						typedef unspecified is_transparent;
						};
						10 operator() returns std::forward<T>(t) > std::forward<U>(u).
						template <> struct less<void> {
						template <class T, class U> **constexpr** auto operator()(T&& t, U&& u) const
						-> decltype(std::forward<T>(t) < std::forward<U>(u));
							Missing edit from approved paper. Fixed in IS.

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

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**

						typedef unspecified is_transparent;
						};
						11 operator() returns std::forward<T>(t) < std::forward<U>(u).
						template <> struct greater_equal<void> {
						template <class T, class U> **constexpr** auto operator()(T&& t, U&& u) const
						-> decltype(std::forward<T>(t) >= std::forward<U>(u));
						typedef unspecified is_transparent;
						};
						12 operator() returns std::forward<T>(t) >= std::forward<U>(u).
						template <> struct less_equal<void> {
						template <class T, class U> **constexpr** auto operator()(T&& t, U&& u) const
						-> decltype(std::forward<T>(t) <= std::forward<U>(u));
						typedef unspecified is_transparent;
						};
						13 operator() returns std::forward<T>(t) <= std::forward<U>(u).	

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

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**

JP 09		20.9.6	Paragraph 1 to 3	ed	"constexpr" specifier should be placed before type specifier.	1 The library provides basic function object classes for all of the logical operators in the language (5.14, 5.15, 5.3.1).
						template <class T = void> struct logical_and {
						constexpr bool operator()(const T& x, const T& y) const;
						typedef T first_argument_type;
						typedef T second_argument_type;
						typedef bool result_type;
						};
						2 operator() returns x && y.
						template <class T = void> struct logical_or {
						constexpr bool operator()(const T& x, const T& y) const;
						typedef T first_argument_type;
						typedef T second_argument_type;
						typedef bool result_type;
						};
						3 operator() returns x || y.
						template <class T = void> struct logical_not {
						constexpr bool operator()(const T& x) const;
						typedef T argument_type;
						typedef bool result_type;
						};
							Misapplied edit from approved paper. Fixed in IS.

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

Template for comments and secretariat observations

Date: 2014-08-12	Document: **ISO/IEC DIS 14882 (Ed4)**	Project: 64029

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
JP 10		23.3.2.1	Paragraph 3	ed	In struct array definition, the member function, data() lack “constexpr”. It meets the requirements of constexpr function as with other constexpr member functions.	**constexpr** const T * data() const noexcept;	<not editorial> Assigned to Core Issues list for consideration in a future revision to the Standard. Not in scope for this revision.
JP 11		30.4.1.1	Paragraph 1	ge	Following explanation of mutex types seems not to correspond to shared timed mutex adopted to this DIS where multiple threads can own single shared mutex. *The mutex types supplied by the standard library provide exclusive ownership semantics: only one thread may own the mutex at a time.*	The sentence should be removed or changed to correct description.	This sentence is removed in the IS. The normative requirements are clearly stated elsewhere.
JP 12		30.4.1.4	Paragraph 2	ed	The maximum number of threads is specified in the following sentence: *The maximum number of execution agents which can share a shared lock on a single shared mutex type is unspecified, but shall be at least 10000.* Is there any rationale about the number, 10000? Alothough it is required, it seems better to move such a kind of implementation limitation at runtime to ‘Annex B: Implementation quantities' .	Move the sentence to Annex B. In addition, it seems better to add any rationale about the value.	<not editorial: Annex B is non-normative, so the proposed change would have conformance impact>
JP 13		30.4.1.4	Paragraph 5, 13, 19, 26, and so on	ed	Some semantics clauses, i.e. ‘Requires’, ‘Effects’, or ‘Postcondition’, lack a "subject". To clarify intent, it should be included in sentences.	Add a corresponding subject .	Not a defect: while informal, this is consistent with the style used in the rest of the standard, and adding a leading “This function” to each of these would not improve clarity. This will be fixed in a future revision of the C++ standard.

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

Template for comments and secretariat observations

Date: 2014-08-12	Document: **ISO/IEC DIS 14882 (Ed4)**	Project: 64029

MB/ NC[1]	**Line number** (e.g. 17)	**Clause/ Subclause** (e.g. 3.1)	**Paragraph/ Figure/ Table/** (e.g. Table 1)	**Type of comment**[2]	**Comments**	**Proposed change**	**Observations of the secretariat**
JP 14		30.4.1.4.1	Paragraph 0 (definition)	ed	Formatting for class shared_timed_mutex is not compatible with others. In addition, at lock() function line, there is a comment, “//blocking”, but in the similar class, timed_mutex, there’s no comment for lock() function. They should be uniformed.	Remove a blank line under “namespace std{“. Indent whole class definition by 2 columns. Remove the comment at lock() and lock_shared(). (or add the same comment at timed_mutex::lock())	This will be fixed in a future revision of the C++ standard.
JP 15		30.4.1.4.1	Paragraph 0 (definition)	te	The shared_timed_mutex class does not contain 'native_handle' for incorporating implementation specific details which is contained in all other mutex type. Is there any reason not to include 'native_handle' in shared_timed_mutex class?	Just confirmation. If intended, it’s ok.	<not editorial>
JP 16		30.4.1.4.1	Paragraph 3	ed	The mixed condition of 3 clauses is ambiguous. Possibly, ‘or’ should be added after the 2nd clause. Please note that there is a period at the end of the original 2nd clause. It should be a comma.	The behavior of a program is undefined if: - it destroys a shared_timed_mutex object owned by any thread, - a thread attempts to recursively gain any ownership of a shared_timed_mutex**, or** - a thread terminates while possessing any ownership of a shared_timed_mutex.	This will be fixed in a future revision of the C++ standard.
JP 17		30.6.4	Paragraph 5	ed	"and" after a sentence at the first dash (“- if the return...”) remains. In N3776, it is removed. Is it correct?	If not correct, remove the “and”.	Not a defect: including “and” or “or” at the end of each list element is correct and consistent with nearby wording.
JP 18				ed	Hanging paragraphs can often be found in this DIS. ISO/IEC Directives do not allow hanging paragraphs.		This is a large-scale reformatting of the standard, and cannot be accommodated within the timeframe for C++14. It will be fixed in a future revision of the C++ standard.

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