Document No:	WG21 N4307
Date:	2014-11-12
References:	ISO/IEC PDTS 19568
Reply To:	Barry Hedquist <beh@peren.com>
	INCITS/PL22.16 IR

National Body Comments

ISO/IEC PDTS 19568

Technical Specification: C++ Extensions for Library Fundamentals

Attached is WG21 N4307, National Body Comments for ISO/IEC PDTS 19568, Technical Specification – C++ Extensions for Library Fundamentals.

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

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

JP 1		3.2.1		te	Current design of apply cannot be used with standard algorithms. This is not consistent with orthogonality policy of C++. We propose make_apply function to make a function object applicable to apply function.
					For reference, there is a similar design in Boost Fusion Library, fused and make_fused(). This experimental study should be taken into account .
						Introduce make_apply as below:
						#include <tuple>
						#include <utility>
						template<typename F, typename Tuple, size_t... I>
						auto apply_impl(F&& f, Tuple&& args, std::index_sequence<I...>)
						{
						return std::forward<F>(f)(std::get<I>(std::forward<Tuple>(args))...);
						}
						template<typename F, typename Tuple,
						typename Indices = std::make_index_sequence<std::tuple_size<Tuple>::value>>
						auto apply(F&& f, Tuple&& args)
						{
						return apply_impl(std::forward<F>(f), std::forward<Tuple>(args), Indices());
						}
						template<typename F, typename Tuple, size_t... I>
						auto apply_impl(F&& f, const Tuple& args, std::index_sequence<I...>)
						{
						return std::forward<F>(f)(std::get<I>(args)...);
						}
						template<typename F, typename Tuple,
						typename Indices = std::make_index_sequence<std::tuple_size<Tuple>	

[1] **MB** = Member body / **NC** = National Committee (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by **)

[2] **Type of comment:** **ge** = general **te** = technical **ed** = editorial

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

						::value>>
						auto apply(F&& f, const Tuple& args)
						{
						return apply_impl(std::forward<F>(f), args, Indices());
						}
						
						template <typename F>
						class apply_functor {
						F f_;
						public:
						explicit apply_functor(F&& f)
						: f_(std::forward<F>(f)) {}
						
						template <typename Tuple>
						auto operator()(Tuple&& args)
						{
						return apply(std::forward<F>(f_), std::forward<Tuple>(args));
						}
						
						template <typename Tuple>
						auto operator()(const Tuple& args)
						{
						return apply(std::forward<F>(f_), args);
						}
						};
						
						template <typename F>
						apply_functor<F> make_apply(F&& f)
						{
						return apply_functor<F>(std::forward<F>(f));	

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

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

						}
						Usage example:
						#include <iostream>
						#include <vector>
						#include <string>
						#include <algorithm>
						int main()
						{
						std::vector<std::tuple<int, char, std::string>> v = {
						{1, 'a', "Alice"},
						{2, 'b', "Bob"},
						{3, 'c', "Carol"}
						};
						std::for_each(v.begin(), v.end(),
						make_apply([](int a, char b, const std::string& c) {
						std::cout << a << ' ' << b << ' ' << c << std::endl;
						}
						));
						}	

GB 1		6.3.1	p15	Te	The allocator-extended copy constructor for std::experimental::any cannot be implemented as specified, so should be removed. Without this constructor, the value of allocator support in std::experimental::any is questionable.	Suggest removing all constructors taking allocator_arg_t from std::experimental::any.	
GB 2		11.2		Te	Conversion should be provided from/to any specific endianness	Addition of further conversion functions to support conversion to and from big-endian and little-endian representations (as a minimum)	

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

NB Comments PDTS 19568, Library Fundamentals

Date:2014-11-11	Document: SC22 / WG21 N4307	Project: 19568

MB/ NC[1]	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment[2]	Comments	Proposed change	Observations of the secretariat
FI 2		[any.cons]	15	te	Implementation vendors report that the signatures that take an any&& or const any& are unimplementable as currently specified.	Either remove allocator support from any or make it use a polymorphic memory resource.	
FI 5		[header.net.synop]		te	As explained in N4249, using the same names for the network byte order conversion functions as the existing posix facilities that may be macros is highly problematic.	Rename the functions so that they do not clash with the existing practice.	
FI 1		[optional.object.observe]	11, 20	te	As per https://issues.isocpp.org/show_bug.cgi?id=45, the rvalue-reference-qualified observers of optional should not return a value, but an rvalue reference instead, in order to ease perfect forwarding and to not cause double-move on emplace to containers. Such a double-move may end up being a double-copy on optionals of legacy types.	Change the signatures to return T&& instead of T and const T&& instead of T	
FI 4		[string.view.access]	19	ed	The note is confusing. basic_string::data() returns a pointer to a null-terminated buffer regardless of how and from what the basic_string was constructed. How/when is the buffer returned by string_view::data() not null-terminated when a string_view has been constructed from a literal, and how is it typical that passing data() to a function expecting a null-terminated char* a mistake?	Clarify or strike the note.	
FI 3		[string.view.cons]	6	ed	“Constructs a basic_string_view referring to the same string as str,”, str doesn’t refer to a string, and the wording is inconsistent with similar constructors for basic_string in the standard proper, where such charT* are said to “point to an array”. See [string.cons] for reference.	Use the same terminology as the standard basic_string specification uses.	

[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