

Document No: WG21 N4678

Date: 2017-07-29

Project: Programming Language C++

References: ISO/IEC PDTS 22277

Reply to: Gor Nishanov <gorn@microsoft.com>

Attached are responses to National Body Comments for ISO/IEC PDTS 22277, C++ Extensions for Coroutines

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
CH 001				te	This TS disallows stackful coroutines. This is too restrictive and stackful coroutines should be allowed as well.	Allow as suspension context functions that were called from a top-level coroutine.	Reject. No consensus for change.
US 002	na	05.03.8	4	Ed	in "The await-expression has the same type and value category as the await-resume expression." await-resume is marked up in bold, should be italics.	Change to italics.	Accept
US 003	na	06.05.4	1	Te	Update range based for statement after C++17	The range-based for statement for co_await _{opt} (<i>for-range-declaration</i> : <i>for-range-initializer</i>) <i>statement</i> is equivalent to { auto &&__range = <i>for-range-initializer</i> ; auto __begin = co_await _{opt} begin-expr ; auto __end = end-expr ; for (; __begin != __end; co_await _{opt} ++__begin) { <i>for-range-declaration</i> = *__begin; <i>statement</i> } }	Reject. Rebase will happen prior to merging into the working paper.
US	na	06.06.3	all	Ge	There are many new cases of undefined behaviour	No action for now. However, experience with TS	Accept. No action for now.

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
004		6.6.3.1 8.4.4 8.11.2.5 18.10 18.11.2.5			introduced by the TS which are somewhat easily triggered by independent parts of the mechanisms, e.g., the result type of the coroutine interacting through the promise_type to allow flow of control to run off the end of a coroutine. In general it would be good to minimize undefined behaviour.	implementation may allow reducing UB. This should form part of any review for integrating coroutines as part of a future standard.	
US 005	na	06.06.3.1	1	Te	Simplify the grammar for <i>coroutine-return-statement</i> : <i>co_return expression_opt</i> ; <i>co_return braced-init-list</i> ;	coroutine-return-statement: co_return expr-or-braced-init-list opt ;	Reject. Rebase will happen prior to merging into the working paper.
US 006	na	08.04.4	12	Technical	Stateful allocators (pmr) do not work this way, there's no mechanism for allocator propagation to the captured state.	Strike section 12, or provide mechanism for holding allocator	Accept. Example in paragraph 12 is removed
US 007	na	08.04.4	3	Ge	Is unhandled_exception() a requirement for a promise_type?	a) Call std::terminate if not present or b) Add unhandled_exception() to the complete example of promise_type in 8.4.4 paragraph 11, the generator example.	Accept. B) is correct. Example was missing unhandled_exception (and a few other things) and now is fixed.

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

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

MB/ NC ¹	Line number	Clause/ Subclause	Paragraph/ Figure/Table	Type of comment ²	Comments	Proposed change	Observations of the secretariat
US 008	4	08.04.4, p15	11	Ed	Note about possibly undefined behaviour	If a coroutine has a parameter passed by reference, resuming the coroutine after the lifetime of the entity referred to by that parameter has ended is likely to result in undefined behavior Strike "likely to result in"	Reject. No consensus for change.
CA 009		18.01 [support.general]	Table 30	ed	The entry for subclause 18.11 appears before the entry for subclause 18.10.	Move the insertion of the entry for subclause 18.11 to appear after the entry for subclause 18.10.	Accept
US 010	na	18.11.01.1	1	Te	Is the template coroutines_traits intended to be a user-extension point? If so, spell out the contract for users to customize this trait. Otherwise, restrict user specialization with the wording for all type traits in the <type_traits> header. 18.11.1p2 suggests the former, while the latter is much simpler to specify for the initial TS.	Specify the exact behaviour of user-customization of coroutine_traits.	Accept.
US 011	na	18.11.02	all	Ge	The specification of each operation is not explicitly clear whether it applied to the specialization of coroutine_handle<void>, or the primary coroutine_handle template.	Break this section into two, to clearly provide definitions for both versions of the template.	Accept
US 012	na	18.11.02	all	Ge	Coroutine handles have essentially raw pointer semantics. Should there be a library type as part of the TS that does destroy / set to nullptr ?	If a library type is needed, please add it.	Reject. No consensus for change
US 013	na	18.11.02	all	Ge	Promise types are required to implement either return_value() or return_void(), but not both, and it is undefined behaviour for a coroutine to run off the end, where return_void would be called.	Consider implementing both either_return() and return_value() for promise types, and eliminate the undefined behaviour.	Reject. No consensus for change

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
					Why not allow both? It could make types that implement the promise_type contract more reuseable.		
US 014	na	18.11.02.5	6	Ed	a concurrent resumption of a coroutine by multiple threads may result in a data race	Possibly means concurrent destruction here, in the destroy method.	Accept
US 015	na	18.11.02.7	all	Te	As coroutine_handle<void> is a literal type, should the comparison operators be constexpr?	Add constexpr to the declaration/definition of operator==, operator!=, operator<, operator<=, operator>=, and operator> for arguments of type coroutine_handle<>.	Accept
US 016	na	18.11.03	1	Te	The names suspend_never and suspend_always should be (inline) constexpr variables of type suspend_never_t and suspend_always_t respectively.	Change suspend_never and suspend_always as appropriate.	Reject. No consensus for change
US 017	na	2 [intro.refs]	1	Ge	<p>We are in the process of balloting the final text of the next C++ standard, provisionally ISO/IEC 14882:2017.</p> <p>We should hold back publishing this TS long enough to rebase on the text of the new standard.</p> <p>Other than updating this reference, the change is almost entirely updating section numbers and cross-references.</p> <p>The normative changes would be</p>	<p>The following referenced document is indispensable for the application of this document. For dated references, only the edition cited applies.</p> <p>(1.1) — ISO/IEC 14882:2014⁷, <i>Programming Languages – C++</i></p> <p>ISO/IEC 14882:2014⁷ is hereafter called the C++ Standard. ...</p> <p>(Still to add a mapping of section numbers and stable-refs)</p> <p>(File separate comments on the noted list of issues - such as range-for below)</p>	Reject. Rebase will happen prior to merging to the working paper.

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
					<p>updating the range based 'for' loop syntax;</p> <p>the text for a 'return' statement would need adjusting;</p> <p>the wording on restrictions with respect to longjmp should be reviewed;</p> <p>hash support for coroutine_handle should be updated with the "enabled" terminology.</p>		
CA 018		2 [intro.refs]		ed	The form required by ISO/IEC Directives, Part 2, 2016 subclause 15.5.1 is not followed.	Use the text provided by the Directives.	Accept
US 019	na	All	all	Ge	The TS presents only low level mechanisms to implement coroutines. For final release in a C++ standard, standard library implementations of generators, futures from coroutines, guard types for handles, etc. should also ship.	Please consider adding standard library implementations of generators, futures from Coroutines, guard types for handles and any others that may be needed when Coroutines are incorporated into the C++ Standard.	Accept. No action for now.
US 020	na	All	all	Ge	Coroutines are invocable types, can they be stored by a std::function? What about a std::function<void()> that discards the result on invocation?	Disallow storing coroutines in std::function objects that discard their result.	Reject. No consensus for change

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