Project: ISO JTC1/SC22/WG21: Programming Language C++
Doc No: WG21 **P0599R0**
Date: 2017-01-14
Reply to: Nicolai Josuttis (nico@josuttis.de)
Audience: LWG
Prev. Version: ----

noexcept for Hash Functions

For C++17; US 140 requests:

> Specializations of std::hash for arithmetic, pointer, and standard library types should not be allowed to throw. The constructors, assignment operators, and function call operator should all be marked as noexcept. It might be reasonable to consider making this a binding requirement on user specializations of the hash template as well (in p1) but that may be big a change to make at this stage.

Discussing it informally in LWG seems to result in the following conclusion:

hash	**should be noexcept?**	**Remark**
hash<error_code>	yes	
hash<error_condition>	yes	
hash<optional<T>>	**no**	same hash as with underlying type
hash<variant<Types...>>	no	
hash<monostate>	yes	
hash<bitset<N>>	yes	
hash<unique_ptr<T, D>>	no	same hash as for underlying raw pointer
hash<shared_ptr<T>>	no	same hash as for underlying raw pointer
hash<NUMERIC>	yes	for all integral types (incl. bool and char) and floating-point types
hash<T*>	yes	(uses the address (can't look at the value because it might change))
hash<type_index>	yes	same as hash_code() of passed index
hash<string>	yes	
hash<u16string>	yes	
hash<u32string>	yes	
hash<wstring>	yes	
hash<string_view>	yes	guarantee to match string hash value
hash<u16string_view>	yes	guarantee to match u16string hash value
hash<u32string_view>	yes	guarantee to match u32string hash value
hash<wstring_view>	yes	guarantee to match wstring hash value
hash<vector<bool, Allocator>>	no	
hash<thread::id>	yes	

That is, for wrapper types we do not require it (yet).

For this reason, this paper proposes:

a) For the moment not to require to mark all hash specializations as noexcept
b) Add the no except requirement for the hash functions as stated above.

Proposed Wording

(All against N4618)

19.5.6 System error hash support [syserr.hash]

§1 (hash for error_code and error_condition) change:

The specializations are enabled (20.14.14) and hash functions are marked noexcept.

20.7.11 Hash support [variant.hash]

§2 (hash for monostate):

The specialization is enabled (20.14.14) and the hash function is marked noexcept.

20.9.3 bitset hash support [bitset.hash]

§1 (for bitset):

The specialization is enabled (20.14.14) and the hash function is marked noexcept.

20.14.14 Class template hash [unord.hash]

§2 (general statement):

… Each header that declares the template hash provides enabled specializations of hash for nullptr_t and all cv-unqualified arithmetic, enumeration, and pointer types, for which the hash functions are marked noexcept.

20.18.4 Hash support [type.index.hash]

For (type_index), add §2:

The specializations are enabled (20.14.14) and the hash functions are marked noexcept.

21.3.4 Hash support [basic.string.hash]

for strings, add §2:

The specialization are enabled (20.14.14) and the hash function is marked noexcept

21.4.5 Hash support [string.view.hash]

§1 (for string_view's):

The specializations are is enabled (20.14.14) and the hash functions are marked noexcept.

30.3.1.1 Class thread::id [thread.thread.id]

§14 (for thread::id):

The specialization is enabled (20.14.14) the hash function is marked noexcept..