TECHNICAL SPECIFICATION

ISO/IEC TS 19568

Second edition
2017-03

Programming languages — C++ extensions for library fundamentals

Langages de programmation — Extensions C++ pour la bibliothèque fondamentaux

iTeh STANDARD PREVIEW
(standards.iteh.ai)

ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-4186403ea99e/iso-iec-ts-19568-2017

Reference number
ISO/IEC TS 19568:2017(E)

© ISO/IEC 2017

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-4186403ea99e/iso-iec-ts-19568-2017

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2017, Published in Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

© ISO/IEC 2017 – All rights reserved

Contents

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-4186403ea99e/iso-iec-ts-19568-2017

© ISO/IEC 2017 — All rights reserved

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-4186403ea99e/iso-iec-ts-19568-2017

© ISO/IEC 2017 — All rights reserved

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-41f6403ea99e/iso-iec-ts-19568-2017

© ISO/IEC 2017 — All rights reserved

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-4186403ea98a/iso-iec-ts-19568-2017

© ISO/IEC 2017 — All rights reserved

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement. For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html

iTeh STANDARD PREVIEW
(standards.iteh.ai)

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, Information Technology, Subcommittee SC 22, Programming languages, their environments and system software interfaces. This edition of ISO/IEC 19568:2017 cancels and replaces the edition ISO/IEC 19568:2015, which has been technically revised and includes the following changes:

ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-4186403ea99e/iso-iec-ts-19568-2017

- Addition of the `sample` algorithm.
- Addition of new random-number generation facilities, and algorithms which use them.
- Addition of algorithms for uniform container erasure.
- Addition of function template `not_fn`.
- Addition of logical operator type traits `conjunction`, `disjunction`, and `negation`.
- Addition of templates to support the "detection idiom".
- Addition of the `propagate_const` class template.
- Addition of the `observer_ptr` class template.
- Addition of the `make_array` and `to_array` function templates.
- Addition of the `ostream_joiner` class template.
- Addition of the `gcd` and `lcm` algorithms.
- Addition of the `source_location` struct.
- Changes to the return types of search algorithms.
- Moving all libraries to the inline namespace `fundamentals_v2`.
- Miscellaneous defect resolutions.

© ISO/IEC 2017 — All rights reserved

iTeh STANDARD PREVIEW
(standards.iteh.ai)

ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-4186403ea99e/iso-iec-ts-19568-2017

1 General [general]

1.1 Scope [general.scope]

1 This technical specification describes extensions to the C++ Standard Library (1.2). These extensions are classes and functions that are likely to be used widely within a program and/or on the interface boundaries between libraries written by different organizations.

2 This technical specification is non-normative. Some of the library components in this technical specification may be considered for standardization in a future version of C++, but they are not currently part of any C++ standard. Some of the components in this technical specification may never be standardized, and others may be standardized in a substantially changed form.

3 The goal of this technical specification is to build more widespread existing practice for an expanded C++ standard library. It gives advice on extensions to those vendors who wish to provide them.

1.2 Normative references [general.references]

1 The following referenced document is indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

— ISO/IEC 14882:2014, *Programming Languages — C++*

2 ISO/IEC 14882:2014 is herein called the *C++ Standard*. References to clauses within the C++ Standard are written as "C++14 §3.2". The library described in ISO/IEC 14882:2014 clauses 17–30 is herein called the *C++ Standard Library*.

3 Unless otherwise specified, the whole of the C++ Standard's Library introduction (C++14 §17) is included into this Technical Specification by reference.

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ee9ee06-f1ca-4ea0-81eb-4186403ea99e/iso-iec-ts-19568-2017

1.3 Namespaces, headers, and modifications to standard classes [general.namespaces]

1 Since the extensions described in this technical specification are experimental and not part of the C++ standard library, they should not be declared directly within namespace `std`. Unless otherwise specified, all components described in this technical specification either:

— modify an existing interface in the C++ Standard Library in-place,
— are declared in a namespace whose name appends `::experimental::fundamentals_v2` to a namespace defined in the C++ Standard Library, such as `std` or `std::chrono`, or
— are declared in a subnamespace of a namespace described in the previous bullet, whose name is not the same as an existing subnamespace of namespace `std`.

[*Example:* This TS does not define `std::experimental::fundamentals_v2::chrono` because the C++ Standard Library defines `std::chrono`. This TS does not define `std::pmr::experimental::fundamentals_v2` because the C++ Standard Library does not define `std::pmr`. — *end example*]

2 Each header described in this technical specification shall import the contents of `std::experimental::fundamentals_v2` into `std::experimental` as if by

```
namespace std {
  namespace experimental {
    inline namespace fundamentals_v2 {}
```

```
  }
}
```

3 This technical specification also describes some experimental modifications to existing interfaces in the C++ Standard Library. These modifications are described by quoting the affected parts of the standard and using underlining to represent added text and ~~strike-through~~ to represent deleted text.

4 Unless otherwise specified, references to other entities described in this technical specification are assumed to be qualified with `std::experimental::fundamentals_v2::`, and references to entities described in the standard are assumed to be qualified with `std::`.

5 Extensions that are expected to eventually be added to an existing header `<meow>` are provided inside the `<experimental/meow>` header, which shall include the standard contents of `<meow>` as if by

```
#include <meow>
```

6 New headers are also provided in the `<experimental/>` directory, but without such an `#include`.

Table 1 — C++ library headers

`<experimental/algorithm>`	`<experimental/map>`	`<experimental/string>`
`<experimental/any>`	`<experimental/memory>`	`<experimental/string_view>`
`<experimental/array>`	`<experimental/memory_resource>`	`<experimental/system_error>`
`<experimental/chrono>`	`<experimental/optional>`	`<experimental/tuple>`
`<experimental/deque>`	`<experimental/propagate_const>`	`<experimental/type_traits>`
`<experimental/forward_list>`	`<experimental/random>`	`<experimental/unordered_map>`
`<experimental/functional>`	`<experimental/ratio>`	`<experimental/unordered_set>`
`<experimental/future>`	`<experimental/regex>`	`<experimental/utility>`
`<experimental/iterator>`	`<experimental/set>`	`<experimental/vector>`
`<experimental/list>`	`<experimental/source_location>`	

iTeh STANDARD PREVIEW
(standards.iteh.ai)

1.4 Terms and definitions [general.defns]

ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-4186403ea99e/iso-iec-ts-19568-2017

1 For the purposes of this document, the terms and definitions given in the C++ Standard and the following apply.

1.4.1 [general.defns.direct-non-list-init]

direct-non-list-initialization
A direct-initialization that is not list-initialization.

1.5 Future plans (Informative) [general.plans]

1 This section describes tentative plans for future versions of this technical specification and plans for moving content into future versions of the C++ Standard.

2 The C++ committee intends to release a new version of this technical specification approximately every year, containing the library extensions we hope to add to a near-future version of the C++ Standard. Future versions will define their contents in `std::experimental::fundamentals_v3`, `std::experimental::fundamentals_v4`, etc., with the most recent implemented version inlined into `std::experimental`.

3 When an extension defined in this or a future version of this technical specification represents enough existing practice, it will be moved into the next version of the C++ Standard by removing the `experimental::fundamentals_v`*N* segment of its namespace and by removing the `experimental/` prefix from its header's path.

© ISO/IEC 2017 — All rights reserved

1.6 Feature-testing recommendations (Informative) [general.feature.test]

1 For the sake of improved portability between partial implementations of various C++ standards, WG21 (the ISO technical committee for the C++ programming language) recommends that implementers and programmers follow the guidelines in this section concerning feature-test macros. [*Note:* WG21's SD-6 makes similar recommendations for the C++ Standard itself. — *end note*]

2 Implementers who provide a new standard feature should define a macro with the recommended name, in the same circumstances under which the feature is available (for example, taking into account relevant command-line options), to indicate the presence of support for that feature. Implementers should define that macro with the value specified in the most recent version of this technical specification that they have implemented. The recommended macro name is "`__cpp_lib_experimental_`" followed by the string in the "Macro Name Suffix" column.

3 Programmers who wish to determine whether a feature is available in an implementation should base that determination on the presence of the header (determined with `__has_include(<header/name>)`) and the state of the macro with the recommended name. (The absence of a tested feature may result in a program with decreased functionality, or the relevant functionality may be provided in a different way. A program that strictly depends on support for a feature can just try to use the feature unconditionally; presumably, on an implementation lacking necessary support, translation will fail.)

Table 2 — Significant features in this technical specification

Doc. No.	Title	Primary Section	Macro Name Suffix	Value	Header
N3915	apply() call a function with arguments from a tuple	3.2.2	`apply`	201402	`<experimental/tuple>`
N3932	Variable Templates For Type Traits	3.3.1	`type_trait_variable_templates`	201402	`<experimental/type_traits>`
N3866	Invocation type traits	3.3.2	`invocation_type`	201406	`<experimental/type_traits>`
P0013R1	Logical Operator Type Traits	3.3.3	`logical_traits`	201511	`<experimental/type_traits>`
N4502	The C++ Detection Idiom	3.3.4	`detect`	201505	`<experimental/type_traits>`
N4388	A Proposal to Add a Const-Propagating Wrapper to the Standard Library	3.7	`propagate_const`	201505	`<experimental/propagate_const>`
N3916	Type-erased allocator for `std::function`	4.2	`function_erased_allocator`	201406	`<experimental/functional>`
N3905	Extending `std::search` to use Additional Searching Algorithms	4.3	`boyer_moore_searching`	201411	`<experimental/functional>`
N4076	A proposal to add a generalized callable negator	4.4	`not_fn`	201406	`<experimental/functional>`
N3672, N3793	A utility class to represent optional objects	5	`optional`	201411	`<experimental/optional>`
N3804	Any Library Proposal	6	`any`	201411	`<experimental/any>`

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-4186405ea79c/iso-iec-ts-19568-2017

© ISO/IEC 2017 — All rights reserved

Doc. No.	Title	Primary Section	Macro Name Suffix	Value	Header
N3921	`string_view`: a non-owning reference to a string	7	`string_view`	201411	`<experimental/string_view>`
N3920	Extending shared_ptr to Support Arrays	8.2	`shared_ptr_arrays`	201406	`<experimental/memory>`
N3916	Polymorphic Memory Resources	8.4	`memory_resources`	201402	`<experimental/memory_resource>`
N4282	The World's Dumbest Smart Pointer	8.12	`observer_ptr`	201411	`<experimental/memory>`
N4273	Uniform Container Erasure	9.1	`erase_if`	201411	`<experimental/vector>`
N4391	make_array	9.2.2	`make_array`	201505	`<experimental/array>`
N4257	Delimited iterators	10.2	`ostream_joiner`	201411	`<experimental/iterator>`
N3916	Type-erased allocator for `std::promise`	11.2	`promise_erased_allocator`	201406	`<experimental/future>`
N3916	Type-erased allocator for `std::packaged_task`	11.3	`packaged_task_erased_allocator`	201406	`<experimental/future>`
N3925	A `sample` Proposal	12.3	`sample`	201402	`<experimental/algorithm>`
N4061	Greatest Common Divisor and Least Common Multiple	13.1.2, 13.1.3	`gcd_lcm`	201411	`<experimental/numeric>`
N4531	`std::rand` replacement	13.2.2.1	`randint`	201511	`<experimental/random>`
N4519	Source-Code Information Capture	14.1	`source_location`	201505	`<experimental/source_location>`

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-4186403ea99e/iso-iec-ts-19568-2017

© ISO/IEC 2017 — All rights reserved

2 Modifications to the C++ Standard Library [mods]

1 Implementations that conform to this technical specification shall behave as if the modifications contained in this section are made to the C++ Standard.

2.1 Uses-allocator construction [mods.allocator.uses]

1 The following changes to the `uses_allocator` trait and to the description of uses-allocator construction allow a `memory_resource` pointer act as an allocator in many circumstances. [*Note:* Existing programs that use standard allocators would be unaffected by this change. — *end note*]

20.7.7 uses_allocator [allocator.uses]

20.7.7.1 uses_allocator trait [allocator.uses.trait]

```
template <class T, class Alloc> struct uses_allocator;
```

Remarks: Automatically detects whether `T` has a nested `allocator_type` that is convertible from `Alloc`. Meets the BinaryTypeTrait requirements (C++14 §20.10.1). The implementation shall provide a definition that is derived from `true_type` if a type `T::allocator_type` exists and either `is_convertible_v<Alloc, T::allocator_type> != false` or `T::allocator_type` is an alias for `std::experimental::erased_type` (3.1.2), otherwise it shall be derived from `false_type`. A program may specialize this template to derive from `true_type` for a user-defined type `T` that does not have a nested `allocator_type` but nonetheless can be constructed with an allocator where either:

- the first argument of a constructor has type `allocator_arg_t` and the second argument has type `Alloc` or
- the last argument of a constructor has type `Alloc`.

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-41f6402ea99e/iso-iec-ts-19568-2017

20.7.7.2 uses-allocator construction [allocator.uses.construction]

Uses-allocator construction with allocator `Alloc` refers to the construction of an object `obj` of type `T`, using constructor arguments `v1, v2, ..., vN` of types `V1, V2, ..., VN`, respectively, and an allocator `alloc` of type `Alloc`, where `Alloc` either (1) meets the requirements of an allocator (C++14 §17.6.3.5), or (2) is a pointer type convertible to `std::experimental::pmr::memory_resource*` (8.5), according to the following rules:

© ISO/IEC 2017 — All rights reserved

3 General utilities library [utilities]

3.1 Utility components [utility]

3.1.1 Header <experimental/utility> synopsis [utility.synop]

```
#include <utility>

namespace std {
namespace experimental {
inline namespace fundamentals_v2 {

  // 3.1.2, Class erased_type
  struct erased_type { };

} // namespace fundamentals_v2
} // namespace experimental
} // namespace std
```

3.1.2 Class erased_type [utility.erased.type]

iTeh STANDARD PREVIEW
(standards.iteh.ai)

1 `struct erased_type { };`

2 The `erased_type struct` is an empty `struct` that serves as a placeholder for a type `T` in situations where the actual type `T` is determined at runtime. For example, the nested type, `allocator_type`, is an alias for `erased_type` in classes that use *type-erased allocators* (see 8.3).

ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-4186403ea99e/iso-iec-ts-19568-2017

3.2 Tuples [tuple]

3.2.1 Header <experimental/tuple> synopsis [header.tuple.synop]

```
#include <tuple>

namespace std {
namespace experimental {
inline namespace fundamentals_v2 {

  // See C++14 §20.4.2.5, tuple helper classes
  template <class T> constexpr size_t tuple_size_v
    = tuple_size<T>::value;

  // 3.2.2, Calling a function with a tuple of arguments
  template <class F, class Tuple>
  constexpr decltype(auto) apply(F&& f, Tuple&& t);

} // namespace fundamentals_v2
} // namespace experimental
} // namespace std
```


© ISO/IEC 2017 — All rights reserved

3.2.2 Calling a function with a `tuple` of arguments [tuple.apply]

1
```
template <class F, class Tuple>
  constexpr decltype(auto) apply(F&& f, Tuple&& t);
```

2 *Effects:* Given the exposition only function

```
template <class F, class Tuple, size_t... I>
constexpr decltype(auto) apply_impl(  // exposition only
    F&& f, Tuple&& t, index_sequence<I...>) {
  return INVOKE(std::forward<F>(f), std::get<I>(std::forward<Tuple>(t))...);
}
```

Equivalent to

```
return apply_impl(std::forward<F>(f), std::forward<Tuple>(t),
    make_index_sequence<tuple_size_v<decay_t<Tuple>>>{});
```

3.3 Metaprogramming and type traits [meta]

3.3.1 Header <experimental/type_traits> synopsis [meta.type.synop]

```
#include <type_traits>

namespace std {
namespace experimental {
inline namespace fundamentals_v2 {

  // See C++14 §20.10.4.1, primary type categories
  template <class T> constexpr bool is_void_v
    = is_void<T>::value;
  template <class T> constexpr bool is_null_pointer_v
    = is_null_pointer<T>::value;
  template <class T> constexpr bool is_integral_v
    = is_integral<T>::value;
  template <class T> constexpr bool is_floating_point_v
    = is_floating_point<T>::value;
  template <class T> constexpr bool is_array_v
    = is_array<T>::value;
  template <class T> constexpr bool is_pointer_v
    = is_pointer<T>::value;
  template <class T> constexpr bool is_lvalue_reference_v
    = is_lvalue_reference<T>::value;
  template <class T> constexpr bool is_rvalue_reference_v
    = is_rvalue_reference<T>::value;
  template <class T> constexpr bool is_member_object_pointer_v
    = is_member_object_pointer<T>::value;
  template <class T> constexpr bool is_member_function_pointer_v
    = is_member_function_pointer<T>::value;
  template <class T> constexpr bool is_enum_v
    = is_enum<T>::value;
  template <class T> constexpr bool is_union_v
    = is_union<T>::value;
  template <class T> constexpr bool is_class_v
```

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC TS 19568:2017
https://standards.iteh.ai/catalog/standards/sist/3ec9ecf6-f1ca-4ea0-8feb-4186403ea99e/iso-iec-ts-19568-2017

© ISO/IEC 2017 — All rights reserved