

PD ISO/IEC TS 19568:2017



BSI Standards Publication

Programming languages — C++ extensions for library fundamentals

National foreword

This Published Document is the UK implementation of ISO/IEC TS 19568:2017. It supersedes PD ISO/IEC TS 19568:2015 which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee IST/5, Programming languages, their environments and system software interfaces.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2017.

Published by BSI Standards Limited 2017

ISBN 978 0 580 93295 3

ICS 35.060

Compliance with a British Standard cannot confer immunity from legal obligations.

This Published Document was published under the authority of the Standards Policy and Strategy Committee on 30 April 2017.

Amendments/corrigenda issued since publication

Date	Text affected
-------------	----------------------

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*



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

© ISO/IEC 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

Contents

Forewordvii
1 General	1
1.1 Scope	1
1.2 Normative references	1
1.3 Namespaces, headers, and modifications to standard classes	1
1.4 Terms and definitions	2
1.5 Future plans (Informative)	2
1.6 Feature-testing recommendations (Informative)	3
2 Modifications to the C++ Standard Library	5
2.1 Uses-allocator construction	5
3 General utilities library	6
3.1 Utility components	6
3.1.1 Header <experimental/utility> synopsis	6
3.1.2 Class erased_type	6
3.2 Tuples	6
3.2.1 Header <experimental/tuple> synopsis	6
3.2.2 Calling a function with a tuple of arguments	7
3.3 Metaprogramming and type traits	7
3.3.1 Header <experimental/type_traits> synopsis	7
3.3.2 Other type transformations	11
3.3.3 Logical operator traits	12
3.3.4 Detection idiom	13
3.4 Compile-time rational arithmetic	14
3.4.1 Header <experimental/ratio> synopsis	14
3.5 Time utilities	15
3.5.1 Header <experimental/chrono> synopsis	15
3.6 System error support	15
3.6.1 Header <experimental/system_error> synopsis	15
3.7 Class template propagate_const	15
3.7.1 Class template propagate_const general	15
3.7.2 Header <experimental/propagate_const> synopsis	16
3.7.3 propagate_const requirements on T	18
3.7.3.1 propagate_const requirements on class type T	18
3.7.4 propagate_const constructors	19
3.7.5 propagate_const assignment	19
3.7.6 propagate_const const observers	20
3.7.7 propagate_const non-const observers	20
3.7.8 propagate_const modifiers	21
3.7.9 propagate_const relational operators	21
3.7.10 propagate_const specialized algorithms	23
3.7.11 propagate_const underlying pointer access	23
3.7.12 propagate_const hash support	23
3.7.13 propagate_const comparison function objects	23
4 Function objects	25
4.1 Header <experimental/functional> synopsis	25
4.2 Class template function	26
4.2.1 function construct/copy/destroy	28
4.2.2 function modifiers	29

4.3	Searchers	29
4.3.1	Class template <code>default_searcher</code>	29
	4.3.1.1 <code>default_searcher</code> creation functions	30
4.3.2	Class template <code>boyer_moore_searcher</code>	30
	4.3.2.1 <code>boyer_moore_searcher</code> creation functions	31
4.3.3	Class template <code>boyer_moore_horspool_searcher</code>	31
	4.3.3.1 <code>boyer_moore_horspool_searcher</code> creation functions	32
4.4	Function template <code>not_fn</code>	33
5	Optional objects	34
5.1	In general	34
5.2	Header <code><experimental/optional></code> synopsis	34
5.3	optional for object types	35
	5.3.1 Constructors	37
	5.3.2 Destructor	39
	5.3.3 Assignment	40
	5.3.4 Swap	43
	5.3.5 Observers	43
5.4	In-place construction	44
5.5	No-value state indicator	44
5.6	Class <code>bad_optional_access</code>	45
5.7	Relational operators	45
5.8	Comparison with <code>nullopt</code>	45
5.9	Comparison with <code>T</code>	46
5.10	Specialized algorithms	47
5.11	Hash support	47
6	Class <code>any</code>	48
6.1	Header <code><experimental/any></code> synopsis	48
6.2	Class <code>bad_any_cast</code>	49
6.3	Class <code>any</code>	49
	6.3.1 <code>any</code> construct/destroy	49
	6.3.2 <code>any</code> assignments	50
	6.3.3 <code>any</code> modifiers	51
	6.3.4 <code>any</code> observers	51
6.4	Non-member functions	51
7	<code>string_view</code>	54
7.1	Header <code><experimental/string_view></code> synopsis	54
7.2	Class template <code>basic_string_view</code>	55
7.3	<code>basic_string_view</code> constructors and assignment operators	57
7.4	<code>basic_string_view</code> iterator support	58
7.5	<code>basic_string_view</code> capacity	59
7.6	<code>basic_string_view</code> element access	59
7.7	<code>basic_string_view</code> modifiers	60
7.8	<code>basic_string_view</code> string operations	60
	7.8.1 Searching <code>basic_string_view</code>	62
7.9	<code>basic_string_view</code> non-member comparison functions	63
7.10	Inserters and extractors	64
7.11	Hash support	65
8	Memory	66
8.1	Header <code><experimental/memory></code> synopsis	66
8.2	Shared-ownership pointers	69
	8.2.1 Class template <code>shared_ptr</code>	69
	8.2.1.1 <code>shared_ptr</code> constructors	72

	8.2.1.2	shared_ptr observers	74
	8.2.1.3	shared_ptr casts	75
	8.2.1.4	shared_ptr hash support	75
	8.2.2	Class template weak_ptr	75
	8.2.2.1	weak_ptr constructors	76
8.3		Type-erased allocator	77
8.4		Header <experimental/memory_resource> synopsis	77
8.5		Class memory_resource	78
	8.5.1	Class memory_resource overview	78
	8.5.2	memory_resource public member functions	79
	8.5.3	memory_resource protected virtual member functions	79
	8.5.4	memory_resource equality	80
8.6		Class template polymorphic_allocator	80
	8.6.1	Class template polymorphic_allocator overview	80
	8.6.2	polymorphic_allocator constructors	81
	8.6.3	polymorphic_allocator member functions	81
	8.6.4	polymorphic_allocator equality	83
8.7		template alias resource_adaptor	83
	8.7.1	resource_adaptor	83
	8.7.2	resource_adaptor_imp constructors	84
	8.7.3	resource_adaptor_imp member functions	84
8.8		Access to program-wide memory_resource objects	85
8.9		Pool resource classes	85
	8.9.1	Classes synchronized_pool_resource and unsynchronized_pool_resource	85
	8.9.2	pool_options data members	87
	8.9.3	pool resource constructors and destructors	88
	8.9.4	pool resource members	88
8.10		Class monotonic_buffer_resource	89
	8.10.1	Class monotonic_buffer_resource overview	89
	8.10.2	monotonic_buffer_resource constructor and destructor	90
	8.10.3	monotonic_buffer_resource members	91
8.11		Alias templates using polymorphic memory resources	91
	8.11.1	Header <experimental/string> synopsis	91
	8.11.2	Header <experimental/deque> synopsis	92
	8.11.3	Header <experimental/forward_list> synopsis	92
	8.11.4	Header <experimental/list> synopsis	92
	8.11.5	Header <experimental/vector> synopsis	93
	8.11.6	Header <experimental/map> synopsis	93
	8.11.7	Header <experimental/set> synopsis	94
	8.11.8	Header <experimental/unordered_map> synopsis	94
	8.11.9	Header <experimental/unordered_set> synopsis	95
	8.11.10	Header <experimental/regex> synopsis	95
8.12		Non-owning pointers	96
	8.12.1	Class template observer_ptr overview	96
	8.12.2	observer_ptr constructors	97
	8.12.3	observer_ptr observers	97
	8.12.4	observer_ptr conversions	97
	8.12.5	observer_ptr modifiers	97
	8.12.6	observer_ptr specialized algorithms	98
	8.12.7	observer_ptr hash support	99
9		Containers	100
	9.1	Uniform container erasure	100

9.1.1	Header synopsis	100
9.1.2	Function template erase_if	101
9.1.3	Function template erase	102
9.2	Class template array	102
9.2.1	Header <experimental/array> synopsis	102
9.2.2	Array creation functions	103
10	Iterators library	104
10.1	Header <experimental/iterator> synopsis	104
10.2	Class template ostream_joiner	104
10.2.1	ostream_joiner constructor	105
10.2.2	ostream_joiner operations	105
10.2.3	ostream_joiner creation function	105
11	Futures	106
11.1	Header <experimental/future> synopsis	106
11.2	Class template promise	106
11.3	Class template packaged_task	107
12	Algorithms library	109
12.1	Header <experimental/algorithm> synopsis	109
12.2	Search	109
12.3	Sampling	110
12.4	Shuffle	110
13	Numerics library	111
13.1	Generalized numeric operations	111
13.1.1	Header <experimental/numeric> synopsis	111
13.1.2	Greatest common divisor	111
13.1.3	Least common multiple	111
13.2	Random number generation	112
13.2.1	Header <experimental/random> synopsis	112
13.2.2	Utilities	112
13.2.2.1	Function template randint	112
14	Reflection library	113
14.1	Class source_location	113
14.1.1	Header <experimental/source_location> synopsis	113
14.1.2	source_location creation	114
14.1.3	source_location field access	114

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

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:

- 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.

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.

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 {}  
    }  
}
```