

First edition
2007-11-15

**Information technology — Programming
languages — Technical Report on C++
Library Extensions**

*Technologies de l'information — Langages de programmation —
Rapport technique sur l'extensions des librairies C++*

Reference number
ISO/IEC TR 19768:2007(E)



PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2007

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Foreword	x
List of Tables	xi
1 Scope	1
1.1 Relation to C++ Standard Library Introduction	1
1.2 Categories of extensions	1
1.3 Namespaces and headers	1
2 General Utilities	3
2.1 Reference wrappers	3
2.1.1 Additions to header <functional> synopsis	3
2.1.2 Class template reference_wrapper	3
2.1.2.1 reference_wrapper construct/copy/destroy	4
2.1.2.2 reference_wrapper assignment	5
2.1.2.3 reference_wrapper access	5
2.1.2.4 reference_wrapper invocation	5
2.1.2.5 reference_wrapper helper functions	5
2.2 Smart pointers	6
2.2.1 Additions to header <memory> synopsis	6
2.2.2 Class bad_weak_ptr	7
2.2.3 Class template shared_ptr	7
2.2.3.1 shared_ptr constructors	9
2.2.3.2 shared_ptr destructor	10
2.2.3.3 shared_ptr assignment	10
2.2.3.4 shared_ptr modifiers	11
2.2.3.5 shared_ptr observers	11
2.2.3.6 shared_ptr comparison	12
2.2.3.7 shared_ptr I/O	13
2.2.3.8 shared_ptr specialized algorithms	13
2.2.3.9 shared_ptr casts	13
2.2.3.10 get_deleter	14
2.2.4 Class template weak_ptr	14
2.2.4.1 weak_ptr constructors	15

2.2.4.2	weak_ptr destructor	15
2.2.4.3	weak_ptr assignment	16
2.2.4.4	weak_ptr modifiers	16
2.2.4.5	weak_ptr observers	16
2.2.4.6	weak_ptr comparison	17
2.2.4.7	weak_ptr specialized algorithms	17
2.2.5	Class template enable_shared_from_this	17
3	Function objects	21
3.1	Definitions	21
3.2	Additions to <functional> synopsis	21
3.3	Requirements	22
3.4	Function return types	23
3.5	Function template mem_fn	24
3.6	Function object binders	24
3.6.1	Class template is_bind_expression	24
3.6.2	Class template is_placeholder	25
3.6.3	Function template bind	25
3.6.4	Placeholders	26
3.7	Polymorphic function wrappers	26
3.7.1	Class bad_function_call	26
3.7.1.1	bad_function_call constructor	27
3.7.2	Class template function	27
3.7.2.1	function construct/copy/destroy	28
3.7.2.2	function modifiers	30
3.7.2.3	function capacity	30
3.7.2.4	function invocation	30
3.7.2.5	function target access	30
3.7.2.6	undefined operators	31
3.7.2.7	null pointer comparison operators	31
3.7.2.8	specialized algorithms	31
4	Metaprogramming and type traits	33
4.1	Requirements	33
4.2	Header <type_traits> synopsis	33
4.3	Helper classes	35
4.4	General Requirements	35
4.5	Unary Type Traits	36
4.5.1	Primary Type Categories	36
4.5.2	Composite type traits	37
4.5.3	Type properties	37
4.6	Relationships between types	39
4.7	Transformations between types	40
4.7.1	Const-volatile modifications	40
4.7.2	Reference modifications	41
4.7.3	Array modifications	41

4.7.4	Pointer modifications	42
4.8	Other transformations	42
4.9	Implementation requirements	42
5	Numerical facilities	45
5.1	Random number generation	45
5.1.1	Requirements	45
5.1.2	Header <random> synopsis	49
5.1.3	Class template <code>variate_generator</code>	51
5.1.4	Random number engine class templates	52
5.1.4.1	Class template <code>linear_congruential</code>	53
5.1.4.2	Class template <code>mersenne_twister</code>	54
5.1.4.3	Class template <code>subtract_with_carry</code>	55
5.1.4.4	Class template <code>subtract_with_carry_01</code>	57
5.1.4.5	Class template <code>discard_block</code>	58
5.1.4.6	Class template <code>xor_combine</code>	59
5.1.5	Engines with predefined parameters	61
5.1.6	Class <code>random_device</code>	62
5.1.7	Random distribution class templates	63
5.1.7.1	Class template <code>uniform_int</code>	63
5.1.7.2	Class <code>bernoulli_distribution</code>	64
5.1.7.3	Class template <code>geometric_distribution</code>	65
5.1.7.4	Class template <code>poisson_distribution</code>	65
5.1.7.5	Class template <code>binomial_distribution</code>	66
5.1.7.6	Class template <code>uniform_real</code>	67
5.1.7.7	Class template <code>exponential_distribution</code>	67
5.1.7.8	Class template <code>normal_distribution</code>	68
5.1.7.9	Class template <code>gamma_distribution</code>	69
5.2	Mathematical special functions	70
5.2.1	Additions to header <cmath> synopsis	70
5.2.1.1	associated Laguerre polynomials	73
5.2.1.2	associated Legendre functions	73
5.2.1.3	beta function	74
5.2.1.4	(complete) elliptic integral of the first kind	74
5.2.1.5	(complete) elliptic integral of the second kind	74
5.2.1.6	(complete) elliptic integral of the third kind	75
5.2.1.7	confluent hypergeometric functions	75
5.2.1.8	regular modified cylindrical Bessel functions	75
5.2.1.9	cylindrical Bessel functions (of the first kind)	75
5.2.1.10	irregular modified cylindrical Bessel functions	76
5.2.1.11	cylindrical Neumann functions	76
5.2.1.12	(incomplete) elliptic integral of the first kind	77
5.2.1.13	(incomplete) elliptic integral of the second kind	77
5.2.1.14	(incomplete) elliptic integral of the third kind	77
5.2.1.15	exponential integral	77
5.2.1.16	Hermite polynomials	78

5.2.1.17	hypergeometric functions	78
5.2.1.18	Laguerre polynomials	78
5.2.1.19	Legendre polynomials	79
5.2.1.20	Riemann zeta function	79
5.2.1.21	spherical Bessel functions (of the first kind)	79
5.2.1.22	spherical associated Legendre functions	80
5.2.1.23	spherical Neumann functions	80
5.2.2	Additions to header <code><math.h></code> synopsis	80
6	Containers	81
6.1	Tuple types	81
6.1.1	Header <code><tuple></code> synopsis	81
6.1.2	Additions to header <code><utility></code> synopsis	82
6.1.3	Class template <code>tuple</code>	83
6.1.3.1	Construction	83
6.1.3.2	Tuple creation functions	84
6.1.3.3	Tuple helper classes	85
6.1.3.4	Element access	86
6.1.3.5	Relational operators	86
6.1.4	Pairs	87
6.2	Fixed size array	88
6.2.1	Header <code><array></code> synopsis	88
6.2.2	Class template <code>array</code>	88
6.2.2.1	<code>array</code> constructors, copy, and assignment	90
6.2.2.2	<code>array</code> specialized algorithms	90
6.2.2.3	<code>array</code> size	90
6.2.2.4	Zero sized arrays	90
6.2.2.5	Tuple interface to class template <code>array</code>	90
6.3	Unordered associative containers	91
6.3.1	Unordered associative container requirements	91
6.3.1.1	Exception safety guarantees	96
6.3.2	Additions to header <code><functional></code> synopsis	97
6.3.3	Class template <code>hash</code>	97
6.3.4	Unordered associative container classes	98
6.3.4.1	Header <code><unordered_set></code> synopsis	98
6.3.4.2	Header <code><unordered_map></code> synopsis	98
6.3.4.3	Class template <code>unordered_set</code>	99
6.3.4.3.1	<code>unordered_set</code> constructors	101
6.3.4.3.2	<code>unordered_set</code> swap	102
6.3.4.4	Class template <code>unordered_map</code>	102
6.3.4.4.1	<code>unordered_map</code> constructors	104
6.3.4.4.2	<code>unordered_map</code> element access	105
6.3.4.4.3	<code>unordered_map</code> swap	105
6.3.4.5	Class template <code>unordered_multiset</code>	105
6.3.4.5.1	<code>unordered_multiset</code> constructors	107
6.3.4.5.2	<code>unordered_multiset</code> swap	108

6.3.4.6	Class template <code>unordered_multimap</code>	108
6.3.4.6.1	<code>unordered_multimap</code> constructors	110
6.3.4.6.2	<code>unordered_multimap</code> swap	111
7	Regular expressions	113
7.1	Definitions	113
7.2	Requirements	113
7.3	Regular expressions summary	115
7.4	Header <code><regex></code> synopsis	115
7.5	Namespace <code>tr1::regex_constants</code>	121
7.5.1	Bitmask Type <code>syntax_option_type</code>	121
7.5.2	Bitmask Type <code>regex_constants::match_flag_type</code>	122
7.5.3	Implementation defined <code>error_type</code>	124
7.6	Class <code>regex_error</code>	125
7.7	Class template <code>regex_traits</code>	125
7.8	Class template <code>basic_regex</code>	127
7.8.1	<code>basic_regex</code> constants	129
7.8.2	<code>basic_regex</code> constructors	129
7.8.3	<code>basic_regex</code> assign	131
7.8.4	<code>basic_regex</code> constant operations	131
7.8.5	<code>basic_regex</code> locale	132
7.8.6	<code>basic_regex</code> swap	132
7.8.7	<code>basic_regex</code> non-member functions	132
7.8.7.1	<code>basic_regex</code> non-member swap	132
7.9	Class template <code>sub_match</code>	132
7.9.1	<code>sub_match</code> members	133
7.9.2	<code>sub_match</code> non-member operators	133
7.10	Class template <code>match_results</code>	138
7.10.1	<code>match_results</code> constructors	139
7.10.2	<code>match_results</code> size	140
7.10.3	<code>match_results</code> element access	140
7.10.4	<code>match_results</code> formatting	141
7.10.5	<code>match_results</code> allocator	141
7.10.6	<code>match_results</code> swap	142
7.11	Regular expression algorithms	142
7.11.1	exceptions	142
7.11.2	<code>regex_match</code>	142
7.11.3	<code>regex_search</code>	144
7.11.4	<code>regex_replace</code>	145
7.12	Regular expression Iterators	146
7.12.1	Class template <code>regex_iterator</code>	146
7.12.1.1	<code>regex_iterator</code> constructors	147
7.12.1.2	<code>regex_iterator</code> comparisons	147
7.12.1.3	<code>regex_iterator</code> dereference	148
7.12.1.4	<code>regex_iterator</code> increment	148
7.12.2	Class template <code>regex_token_iterator</code>	149

7.12.2.1	regex_token_iterator constructors	150
7.12.2.2	regex_token_iterator comparisons	151
7.12.2.3	regex_token_iterator dereference	151
7.12.2.4	regex_token_iterator increment	151
7.13	Modified ECMAScript regular expression grammar	152
8	C compatibility	155
8.1	Additions to header <complex>	155
8.1.1	Synopsis	155
8.1.2	Function acos	155
8.1.3	Function asin	155
8.1.4	Function atan	156
8.1.5	Function acosh	156
8.1.6	Function asinh	156
8.1.7	Function atanh	156
8.1.8	Function fabs	156
8.1.9	Additional Overloads	156
8.2	Header <ccomplex>	157
8.3	Header <complex.h>	157
8.4	Additions to header <cctype>	157
8.4.1	Synopsis	157
8.4.2	Function isblank	157
8.5	Additions to header <ctype.h>	157
8.6	Header <cfenv>	157
8.6.1	Synopsis	157
8.6.2	Definitions	158
8.7	Header <fenv.h>	158
8.8	Additions to header <float>	158
8.9	Additions to header <float.h>	158
8.10	Additions to header <ios>	158
8.10.1	Synopsis	158
8.10.2	Function hexfloat	158
8.11	Header <inttypes>	159
8.11.1	Synopsis	159
8.11.2	Definitions	159
8.12	Header <inttypes.h>	159
8.13	Additions to header <limits>	160
8.14	Additions to header <limits.h>	160
8.15	Additions to header <locale>	160
8.16	Additions to header <math>	160
8.16.1	Synopsis	160
8.16.2	Definitions	164
8.16.3	Function template definitions	164
8.16.4	Additional overloads	165
8.17	Additions to header <math.h>	166
8.18	Additions to header <stdarg>	166

8.19	Additions to header <stdarg.h>	166
8.20	The header <cstdbool>	166
8.21	The header <stdbool.h>	166
8.22	The header <cstdint>	166
8.22.1	Synopsis	166
8.22.2	Definitions	167
8.23	The header <stdint.h>	167
8.24	Additions to header <cstdio>	167
8.24.1	Synopsis	167
8.24.2	Definitions	168
8.24.3	Additional format specifiers	168
8.24.4	Additions to header <stdio.h>	168
8.25	Additions to header <cstdlib>	168
8.25.1	Synopsis	168
8.25.2	Definitions	169
8.25.3	Function abs	169
8.25.4	Function div	169
8.26	Additions to header <stdlib.h>	169
8.27	Header <ctgmath>	169
8.28	Header <tgmath.h>	169
8.29	Additions to header <ctime>	169
8.30	Additions to header <wchar>	169
8.30.1	Synopsis	169
8.30.2	Definitions	170
8.30.3	Additional wide format specifiers	170
8.31	Additions to header <wchar.h>	170
8.32	Additions to header <wctype>	170
8.32.1	Synopsis	170
8.32.2	Function iswblank	170
8.33	Additions to header <wctype.h>	170
A	Implementation quantities	171
	Bibliography	173
	Index	175

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, the joint technical committee may propose the publication of a Technical Report of one of the following types:

- type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;
- type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;
- type 3, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard (“state of the art”, for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 19768, which is a Technical Report of type 2, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

List of Tables

1	Utilities library summary	3
2	Function object library summary	21
3	Type traits library summary	33
4	Primary Type Category Predicates	36
5	Composite Type Category Predicates	37
6	Type Property Predicates	37
7	Type Property Queries	38
8	Type Relationship Predicates	39
9	Const-volatile modifications	40
10	Reference modifications	41
11	Array modifications	41
12	Pointer modifications	42
13	Other transformations	42
14	Numerical library summary	45
15	Uniform random number generator requirements	45
16	Pseudo-random number engine requirements (in addition to uniform random number generator, CopyConstructible, and Assignable)	46
17	Random distribution requirements (in addition to CopyConstructible, and Assignable)	48
18	Additions to header <cmath> synopsis	72
19	Container library summary	81
20	Container requirements that are not required for unordered associative containers	91
21	Unordered associative container requirements (in addition to container)	92
22	regular expression traits class requirements	114
23	syntax_option_type effects	122
24	regex_constants::match_flag_type effects when obtaining a match against a character container sequence [first,last).	123
25	error_type values in the C locale	124
26	match_results assignment operator effects	140
27	Effects of regex_match algorithm	142
28	Effects of regex_search algorithm	144

1 Scope

[tr.intro]

- 1 This Technical Report describes extensions to the *C++ standard library* that is described in the International Standard for the C++ programming language [14].
- 2 This Technical Report is non-normative. Some of the library components in this Technical Report 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 Report may never be standardized, and others may be standardized in a substantially changed form.
- 3 The goal of this Technical Report 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.

1.1 Relation to C++ Standard Library Introduction

[tr.description]

- 1 Unless otherwise specified, the whole of the ISO C++ Standard Library introduction [lib.library] is included in this Technical Report by reference.

1.2 Categories of extensions [tr.intro.ext]

- 1 This Technical Report describes four general categories of library extensions:
 1. New requirement tables, such as the regular expression traits requirements in 7.2. These are not directly expressed as software; they specify the circumstances under which user-written components will interoperate with the components described in this Technical Report.
 2. New library components (types and functions) that are declared in entirely new headers, such as the class templates in the `<unordered_set>` header (6.3.4.1).
 3. New library components declared as additions to existing standard headers, such as the mathematical special functions added to the headers `<cmath>` and `<math.h>` in 5.2.1 and 5.2.2
 4. Additions to standard library components, such as the extensions to class `std::pair` in 6.1.4.
- 2 New headers are distinguished from extensions to existing headers by the title of the *synopsis* clause. In the first case, the title is of the form “Header `<foo>` synopsis”, and the synopsis includes all namespace scope declarations contained in the header. In the second case, the title is of the form “Additions to header `<foo>` synopsis” and the synopsis includes only the extensions, *i.e.* those namespace scope declarations that are not present in the C++ standard [14].

1.3 Namespaces and headers**[tr.intro.namespaces]**

- 1 Since the extensions described in this Technical Report are 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 Report are declared in namespace `std::tr1`. [*Note*: Some components are declared in subnamespaces of namespace `std::tr1`. —*end note*]
- 2 Unless otherwise specified, references to other entities described in this Technical Report are assumed to be qualified with `std::tr1::`, and references to entities described in the International Standard are assumed to be qualified with `std::`.
- 3 Even when an extension is specified as additions to standard headers (the third category in 1.2), vendors should not simply add declarations to standard headers in a way that would be visible to users by default. [*Note*: That would fail to be standard conforming, because the new names, even within a namespace, could conflict with user macros. —*end note*] Users should be required to take explicit action to have access to library extensions.
- 4 It is recommended either that additional declarations in standard headers be protected with a macro that is not defined by default, or else that all extended headers, including both new headers and parallel versions of standard headers with non-standard declarations, be placed in a separate directory that is not part of the default search path.