Technical Specification

ISO/IEC TS 9922

Programming Languages — Technical specification for C++ extensions for concurrency 2

Langages de programmation — Spécification technique pour les extensions C++ de concurrency 2

First edition
2024-11

Reference number
ISO/IEC TS 9922:2024(en)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Foreword iv

1 Scope 1

2 Normative references 2

3 Terms and definitions 3

4 General 4
4.1 Implementation compliance . . . 4
4.2 Namespaces and headers and modifications to standard classes . . . 4
4.3 **Feature-testing recommendations** . . . 4

5 Synchronized Value 6
5.1 General . . . 6
5.2 Header <experimental/synchronized_value> synopsis . . . 6
5.3 Class template `synchronized_value` . . . 6
5.4 `apply` function . . . 7

6 Safe reclamation 8
6.1 General . . . 8
6.2 Hazard pointers . . . 8
6.3 Read-copy update (RCU) . . . 13

7 Bytewise Atomic Memcpy 16
7.1 General . . . 16
7.2 Header `<experimental/bytewise_atomic_memcpy>` synopsis . . . 16
7.3 `atomic_load_per_byte_memcpy` . . . 16
7.4 `atomic_store_per_byte_memcpy` . . . 16

8 Asymmetric Fence 17
8.1 General . . . 17
8.2 Header <experimental/asymmetric_fence> synopsis . . . 17
8.3 asymmetric_thread_fence_heavy . . . 17
8.4 asymmetric_thread_fence_light . . . 17

9 Order and consistency 19

Foreword [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 o f t echnical a c tivity. I SO a nd IEC technical committees collaborate in fields o f m utual i n terest. O ther i nternational o rganizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

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 a p proval c r iteria n e eded f o r t h e different types of document 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 or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific t erms a nd 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 www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

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*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

1 Scope [scope]

This document builds upon ISO/IEC 14882 by describing requirements for implementations of an interface that computer programs written in the C++ programming language could use to invoke algorithms with concurrent execution. The algorithms described by this document are realizable across a broad class of computer architectures. This document is written as a set of differences from the base standard.

Some of the functionality described by this document might be considered for standardization in a future version of C++, but it is not currently part of ISO/IEC 14882:2020. Some of the functionality in this document might never be standardized, and other functionality might be standardized in a substantially different form.

The goal of this document is to build widespread existing practice for concurrency in the ISO/IEC 14882:2020 algorithms library. It gives advice on extensions to those vendors who wish to provide them.

2 Normative references [refs]

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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 14882:2020, *Programming Languages — C++*

3 Terms and definitions [defs]

No terms and definitions are listed in this document.
ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— IEC Electropedia: available at www.electropedia.org

— ISO Online browsing platform: available at www.iso.org/obp

4 General [general]

4.1 Implementation compliance [general.compliance]

Conformance requirements for this document are those defined in 4.1, as applied to a merged document consisting of ISO/IEC 14882:2020 amended by this document.

NOTE Conformance is defined in terms of the behaviour of programs.

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

Since the extensions described in this document are experimental and not part of the ISO/IEC 14882:2020 library, they are not declared directly within namespace `std`. Unless otherwise specified, all components described in this document either:

— modify an existing interface in the ISO/IEC 14882:2020 library in-place,

— are declared in a namespace whose name appends `::experimental::concurrency_v2` to a namespace defined in the ISO/IEC 14882:2020 library, such as `std`, 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`.

Whenever an unqualified name is used in the specification of a declaration `D`, its meaning is established in accordance with 4.1.2 by performing unqualified name lookup in the context of `D`.

NOTE 1 Argument-dependent lookup is not performed.

Similarly, the meaning of a qualified-id is established in accordance with performing qualified name lookup in the context of `D`.

NOTE 2 Operators in expressions are not so constrained.

Table 1 shows the headers described in this document

Table 1 — C++ library headers

`<experimental/asymmetric_fence>`
`<experimental/bytewise_atomic_memcpy>`
`<experimental/hazard_pointer>`
`<experimental/rcu>`
`<experimental/synchronized_value>`

4.3 Feature-testing recommendations [general.feature.test]

An implementation that provides support for this document should define each feature test macro defined in Table 2 and Table 3 if no associated headers are indicated for that macro, and if associated headers are indicated for a macro, that macro is defined after inclusion of one of the corresponding headers specified in Table 2 and Table 3.

Table 2 — Feature-test macros name

Title	Subclause	Macro name
Synchronized Value	5	`__cpp_lib_experimental_synchronized_value`
Hazard pointers	6.2	`__cpp_lib_experimental_hazard_pointer`
Read-copy update(RCU)	6.3	`__cpp_lib_experimental_rcu`
Bytewise atomic memcpy	7	`__cpp_lib_experimental_bytewise_atomic_memcpy`
Asymmetric Fence	8,33	`__cpp_lib_experimental_asymmetric_fence`

Table 3 — Feature-test macros header

Title	Value	Header
`Synchronized Value`	202406	`<experimental/synchronized_value>`
`Hazard pointers`	202406	`<experimental/hazard_pointer>`
`Read-copy update(RCU)`	202406	`<experimental/rcu>`
`Bytewise atomic memcpy`	202406	`<experimental/bytewise_atomic_memcpy>`
`Asymmetric Fence`	202406	`<experimental/asymmetric_fence>`

5 Synchronized Value [synchronizedvalue]

5.1 General [synchronizedvalue.general]

This clause describes a class template to provide locked access to a value in order to facilitate the construction of race-free programs.

5.2 Header <experimental/synchronized_value> synopsis [synchronizedvalue.syn]

```
namespace std::experimental::inline concurrency_v2 {
    template<class T>
    class synchronized_value;

    template<class F,class... ValueTypes>
    invoke_result_t<F, ValueTypes&...> apply(
        F&& f,synchronized_value<ValueTypes>&... values);
}
```

5.3 Class template synchronized_value [synchronizedvalue.class]

```
namespace std::experimental::inline concurrency_v2 {
  template<class T>
    class synchronized_value
    {
    public:
        synchronized_value(const synchronized_value&) = delete;
        synchronized_value& operator=(const synchronized_value&) = delete;

        template<class... Args>
        synchronized_value(Args&&... args);

    private:
        T value;    // exposition only
        mutex mut;  // exposition only
    };

template<class T>
synchronized_value(T)
-> synchronized_value<T>;
}
```


An object of type `synchronized_value<T>` wraps an object of type `T`. The wrapped object can be accessed by passing a callable object or function to `apply`. All such accesses are done with a lock held to ensure that only one thread may be accessing the wrapped object for a given `synchronized_value` at a time.

```
template<class... Args>
synchronized_value(Args&&... args);
```

Constraints:

- `(sizeof...(Args) != 1)` is `true` or `(!same_as<synchronized_value,remove_cvref_t<Args>>&&...)` is `true`
- `is_constructible_v<T,Args...>` is `true`

Effects: Direct-non-list-initializes *value* with `std::forward<Args>(args)...`.

Throws: Any exceptions emitted by the initialization of *value*.
`system_error` if any necessary resources cannot be acquired.