Remove Deprecated Atomic Initialization API from C++26

Document #:	P3366R1
Date:	2025-03-16
Project:	Programming Language C++
Audience:	LWG

Reply-to:	Alisdair Meredith <ameredith1@bloomberg.net>

Contents

1 Abstract **1**

2 Revision History **2**

3 Introduction **2**

4 Analysis **2**

5 Design Principles **2**

6 Proposed Solution **3**

7 C++26 Reviews **3**
7.1 SG1 Review: Hagenberg, 2025/02/12 3
7.2 LEWG Review: Telecon, 2025/03/04 3

8 Wording **4**

9 Acknowledgements **5**

10 References **5**

1 Abstract

This paper proposes removing the deprecated atomic initialization facility from the next C++ Standard.

2 Revision History

R1 March 2025 (post-Hagenberg mailing)

- Recorded SG1 review at Hagenberg
- Recorded LEWG telecon review following Hagenberg
- Forwarded to LWG for C++29 or, time and resource permitting, for C++26
- Rebased wording onto latest working draft, N5008
  - cleaned up presentation of adding entries to tables and lists
  - added Annex C wording
  - incorporated additions to Annex D from Hagenberg

R0 August 2024 (midterm mailing)

Initial draft of this paper, based on content in [P2863]

3 Introduction

The topic of this paper has been extracted from the general deprecation review paper, [P2863], into its own paper so as to better track its progress, since this topic has had a couple of reviews but is not reaching a real conclusion while embedded in the broader paper.

The original API to initialize atomic variables for C++11 was deprecated for C++20 when the `atomic` template was given a default constructor to correctly perform the necessary initialization — see [P0883R2] for details. This paper proposes that now is the right time to remove that API from the C++ Standard.

4 Analysis

This legacy API continues to function but is more cumbersome than necessary. No compelling case appears to be made that the API is a risk through misuse. However, if updating the C++ Standard's reference to the C Library up to C23 removes the `ATOMIC_VAR_INT` macro, we might want to consider its removal for C++26 as well.

While the `ATOMIC_VAR_INT` macro does no active harm, maintaining text in the Standard always comes with a cost; for example, [P2866] required LWG time to review and update D.22.3 [depr.atomics.nonmembers].

The deprecation and removal of this feature is reflected in the C Standard that initially deprecated the `ATOMIC_VAR_INT` macro (marked it as obsolescent) in C17 and actively removed it from the C23 Standard, per [WG14:N2390]. WG21 should strongly consider removing this macro but perhaps as part of a broader paper to update our reference to the C23 Standard Library.

Note that the C standard retains a generic `atomic_init` function that is *not* part of C++; i.e., we do not support that generic function in `<stdatomic.h>`.

5 Design Principles

Remove deprecated features from the Standard specification at the earliest *practical* opportunity to minimize the burden of accumulating obsolete specifications to maintain, reference, distract, and teach (to avoid).

6 Proposed Solution

Remove the deprecated Standard Library API from C++26 while granting vendors permission to continue supplying it as a conforming extension, for as long as they desire, through the use of zombie names.

Note that, assuming [P2866] lands, which is ahead of this paper in the pipeline to plenary, then this paper will remove the remaining parts of D.22 [depr.atomics], so we will present wording assuming that paper will have landed. If that paper fails to proceed, then the only change to the wording would be that the parent clause D.22 [depr.atomics] is not removed.

7 C++26 Reviews

7.1 SG1 Review: Hagenberg, 2025/02/12

Concerns about removal breaking code were addressed by the Zombie Names clause.

Concerns were raised about whether this removal breaks our compatibility with C `<stdatomic.h>`, but the group seemed satisfied that the compatible symbols are marked as obsolescent (deprecated in C) in the C23 Standard.

There was a question of whether this deprecated API is even a concurrency feature for SG1 to opine on, but agreement was reached that, since the proposed changes touch `<atomic>`, SG1 is the appropriate group for initial review.

Poll: Forward P3366R0 to LEWG for C++26.

```
SF F  N  A SA
2  2  1  0  1
```

Consensus

SA : it's always a burden to programmers if they have to change code that worked before.

7.2 LEWG Review: Telecon, 2025/03/04

Review for this paper was deferred from review in Hagenberg to the first following telecon, along with other deprecation-removal papers. The review intent is to poll forwarding these papers to LWG for C++29 and, if that poll succeeds by a follow-up poll, if time and LWG resources permit, for C++26.

The same concerns that were raised by SG1 were independently raises by LEWG and addressed in the same manner.

The review noted a number of formatting issues and the lack of Annex C wording. The author noted that he usually defers the effort of writing Annex C wording until the design is approved. However, Annex C wording is produced ahead of time if the examples are thought to be helpful in evaluating a given removal.

Two polls were taken.

POLL: Fix P3366R0 formatting as needed (and other minor fixes needed) and forward to LWG for C++29.

```
SF  F  N  A SA
10  8  0  0  0
```

Unanimous approval.

POLL: Fix P3366R0 formatting as needed (and other minor fixes needed) and forward to LWG with recommendation to apply for C++26 (if possible).

```
SF F  N  A SA
8  7  2  0  0
```

Strong consensus.

8 Wording

Make the following changes to the C++ Working Draft. All wording is relative to [N5008], the latest draft at the time of writing, and, for purposes of parallel merges, assumes that the latest update of [P2866] has been applied allowing the removal of the whole of D.22.1 [depr.atomics.general]. Note that the addition of D.22.5 **[depr.atomics.order]** did *not* add any entries to this header as its addition is entirely to an `enum` that is already declared in the header synopsis of the primary clause.

16.4.5.3.2 [zombie.names] Zombie names

Add new identifiers to table 38 [tab:zombie.names.std]

— `ATOMIC_VAR_INIT`
— `atomic_init`

C.1.8 [diff.cpp23.depr] Annex D: compatibility features

x **Change:** Remove the deprecated function `atomic_init` and the macro `ATOMIC_VAR_INIT`.

Rationale: The feature was initially intended to improve compatibility between C and C++. It did not serve well and is deprecated or obsolescent in both languages. Ongoing support remains at the implementers' discretion, exercising freedoms granted by 16.4.5.3.2 [zombie.names].

Effect on original feature: A valid C++ 2023 program using this function or macro may fail to compile.

D.22 [depr.atomics] Deprecated atomic operations

D.22.1 [depr.atomics.general] General

1 The header `<atomic>` (32.5.2 [atomics.syn]) has the following additions.

```
namespace std {
  template<class T>
    void atomic_init(volatile atomic<T>*, typename atomic<T>::value_type) noexcept;
  template<class T>
    void atomic_init(atomic<T>*, typename atomic<T>::value_type) noexcept;

  #define ATOMIC_VAR_INIT(value) see below
}
```

D.22.3 [depr.atomics.nonmembers] Non-member functions

```
template<class T>
  void atomic_init(volatile atomic<T>* object, typename atomic<T>::value_type desired) noexcept;
template<class T>
  void atomic_init(atomic<T>* object, typename atomic<T>::value_type desired) noexcept;
```

1 *Constraints:* For the `volatile` overload of this function, `atomic<T>::is_always_lock_free` is `true`.

2 *Effects:* Equivalent to: `atomic_store_explicit(object, desired, memory_order::relaxed);`

D.22.4 [depr.atomics.types.operations] Operations on atomic types

```
#define ATOMIC_VAR_INIT(value) see below
```

1 The macro expands to a token sequence suitable for constant initialization of an atomic variable of static storage duration of a type that is initialization-compatible with value.

[*Note 1:* This operation possibly needs to initialize locks. —*end note*]

Concurrent access to the variable being initialized, even via an atomic operation, constitutes a data race.

[*Example 1:*

```
atomic<int> v = ATOMIC_VAR_INIT(5);
```

—*end example*]

Update cross-reference for stable labels for C++23

Add the following entries to the list of cross-references for stable labels in previous standards.

depr.atomics.general *removed*
depr.atomics.nonmembers *removed*
depr.atomics.operations *removed*

9 Acknowledgements

Thanks to Michael Park for the pandoc-based framework used to transform this document's source from Markdown.

Thanks to Lori Hughes for reviewing this paper.

10 References

[N5008] Thomas Köppe. Working Draft, Programming Languages — C++.
https://wg21.link/n5008

[P0883R2] Nicolai Josuttis. 2019-11-08. Fixing Atomic Initialization.
https://wg21.link/p0883r2

[P2863] Alisdair Meredith. Review Annex D for C++26.
https://wg21.link/p2863

[P2866] Remove Deprecated Volatile Features from C++26. Review Annex D for C++26.
https://wg21.link/p2866

[WG14:N2390] Jens Gustedt. 2019-06-07. Remove ATOMIC VAR INIT.
https://www.open-std.org/jtc1/sc22/wg14/www/docs/n2390.pdf