**Doc No:** D1685R1

**Date:** 2019-06-14

**Audience:** LEWG

**Authors:** Pablo Halpern <[phalpern@halpernwightsoftware.com](mailto:phalpern@halpernwightsoftware.com)>

Make get/set\_default\_resource Replaceable

**Contents**

1 Abstract 1

2 Change History 1

2.1 R0 (Pre-Cologne, June 2019) 1

3 Motivation and Proposal Overview 1

4 Impact on the Standard 2

5 Potential extensions 2

6 Formal Wording 2

6.1 Document Conventions 2

6.2 Changes to set/get\_default\_resource descriptions 3

6.3 Changes to Replacement Functions section 4

# Abstract

This paper proposes that the functions std::pmr::get\_default\_resource and std::pmr::set\_default\_resource be modified to be replaceable functions (similar to operator new and operator delete) so that the application developer can replace them with more efficient or more flexible alternative implementations.

This proposal is targeted for the C++23 working paper.

# Change History

## R1 (Cologne, July 2019)

LEWG-I reviewed the paper. There was concern that replaceable functions should be EWG-I’s purview.

As per LEWG-I’s guidance, this version deprecates set\_default\_resource. The original scheme is documented as an alternative. As the author, I disagree with this change, but it is acceptable if it increases consensus.

## R0 (Pre-Cologne, June 2019)

Initial revision.

# Motivation and Proposal Overview

The default constructor for std::pmr::polymorphic\_allocator<T>, obtains a pointer to the *default memory resource* by calling std::pmr::get\_default\_resource(). The default memory resource can be changed by calling std::pmr::set\_default\_resource(). This interface, while providing a simple way for the application owner to choose a default memory-allocation strategy for the program, has a few disadvantages:

1. The get/set interface must use atomic variables to avoid race conditions. The acquire barrier on each call to get\_default\_resource can have a negative performance impact on certain architectures.
2. There is no reliable way to set the default resource for objects defined in global scope or namespace scope.
3. A poorly written library can modify the default resource without the knowledge of the application owner, causing serious performance or even correctness problems.

In common usage, the default resource is set at most once at the start of a program, ideally before any use of get\_default\_resource, and is never changed throughout the course of the program.[[1]](#footnote-1) This use case is best supported by defining get\_default\_resource() to simply return the application’s preferred default memory resource and defining set\_default\_resource() either to be a no-op or to abort the program (as it could be considered an error to ever call set\_default\_resource in this case).

Another use case would be to have a thread-local default memory resource. In this case, set\_default\_resource would store a value in thread-local storage and get\_default\_resource would retrieve the value from thread-local storage, with no need for atomic operations.

Both of the above use cases, as well as just about any other scenario, can be supported in the standard by allowing the programmer to replace get\_default\_resource and set\_default\_resource. This paper proposes exactly that, borrowing language from other replaceable functions in the standard, namely operator new and operator delete.

# Impact on the Standard

Although it changes the attributes of a pair of standard library functions, this proposal should have no effect on existing programs and minimal effect on existing standard library implementations.

# Alternatives considered

The R0 version of this paper did not deprecate set\_default\_resource. Authors note: I disagree with the deprecation because I don’t see a reason to break C++17 and C++20 code that used this function. This changed had better consensus in LEWG-I in Cologne than did the original paper, hence the change.

It is also possible to deprecate *both* set\_default\_resource and get\_default\_resource, requiring anybody who wants to write tests or logging allocators for global use to, instead, replace operator new and operator delete or change things outside of the standard (such as linking a different malloc or free). This change seems far too drastic, severely breaks code written in C++17 and C++20, and is a significant change from the accepted PMR design.

# Potential extensions

The committee might consider extending replaceability to all of the functions that install and retrieve handlers, specifically set/get\_new\_handler and set/get\_terminate. It is not clear that there is sufficient motivation to do so, as these functions are used much less frequently than pmr::get\_default\_resource. However, if the LEWG decides that this direction is desirable, the author would be willing to update this paper or submit a new paper proposing such a change.

# Formal Wording

## Document Conventions

All section names and numbers are relative to the March 2019 C++ Working Paper, [N4810](%16http:/www.open-std.org/JTC1/SC22/WG21/docs/papers/2019/n4810.pdf).

Existing working paper text is indented and shown in dark blue. Edits to the working paper are shown with red strikeouts for deleted text and green underlining for inserted text within the indented blue original text.

Comments and rationale mixed in with the proposed wording appears as shaded text.

## Changes to set/get\_default\_resource descriptions

In Section 20.12.4 [mem.res.global], amend the definitions of set\_default\_resource and get\_default\_resource as follows:

The *default memory resource* pointer is a pointer to a memory resource that is used by certain facilities when an explicit memory resource is not supplied through the interface. Its initial value is the return value of new\_delete\_resource(). The value of the default memory resource pointer is retrieved by calling get\_default\_resource()and can usually be set by calling set\_default\_resource(). The library provides default definitions for these functions, which are replaceable. A C++ program shall provide at most one definition of each of these functions. Any such function definition replaces the default version provided in the library (16.5.4.6 [replacement.functions]).

memory\_resource\* set\_default\_resource(memory\_resource\* r) noexcept;

*Effects*: If r is non-null, sets the value of the default memory resource pointer to r, otherwise sets the default memory resource pointer to new\_delete\_resource().

*Returns:* The previous value of the default memory resource pointer.

Remarks: Calling the set\_default\_resource and get\_default\_resource functions shall not incur a data race. A call to the set\_default\_resource function shall synchronize with subsequent calls to the set\_default\_resource and get\_default\_resource functions.

*Replaceable:* A C++ program may define a function with this function signature, and thereby displace the default version defined by the C++ standard library.

*Required behavior:* Return the previous value of the default memory resource pointer or else terminate the program. [*Note:* There is no requirement that a replacement function modify the default memory resource pointer in any way – *end note*]

The required behavior does not require much – not even that the set/get functions be thread safe (not incur races). It is quite possible that set\_default\_resource would be called only before multiple threads were started. Thus, certain replacement versions of these functions could depend on the program to avoid races, rather than guaranteeing thread-safe behavior themselves.

*Default behavior:* If r is non-null, sets the value of the default memory resource pointer to r, otherwise sets the default memory resource pointer to new\_delete\_resource(). Calling the set\_default\_resource and get\_default\_resource functions shall not incur a data race. A call to the set\_default\_resource function shall synchronize with subsequent calls to the set\_default\_resource and get\_default\_resource functions.

memory\_resource\* get\_default\_resource() noexcept;

*Returns:* The current value of the default memory resource pointer.

*Replaceable:* A C++ program may define a function with this function signature, and thereby displace the default version defined by the C++ standard library.

*Required behavior:* Return a non-null pointer to a memory resource, whether or not set\_default\_resource() has previously been called.

*Default behavior:* Return the value set by the most recent call to set\_default\_resource(), if any; otherwise the value returned by new\_delete\_resource().

## Changes to Replacement Functions section

Amend section 16.5.4.6 [replacement.functions] as follows:

A C++ program may provide the definition for any of the following dynamic memory allocation function signatures declared in header <new> (6.6.5.4, 17.6):

operator new(std::size\_t)

operator new(std::size\_t, std::align\_val\_t)

…

operator delete[](void\*, const std::nothrow\_t&)

operator delete[](void\*, std::align\_val\_t, const std::nothrow\_t&)

or any of the following default-resource management function signatures declared in header <memory\_resource> (20.12.4 [mem.res.global]):

std::pmr::memory\_resource\*  
 std::pmr::set\_default\_resource(memory\_resource\* r) noexcept

std::pmr::memory\_resource\* std::pmr::get\_default\_resource() noexcept

The program’s definitions are used instead of the default versions supplied by the implementation (17.6, 20.12.4). Such replacement occurs prior to program startup (6.2, 6.8.3). The program’s declarations shall not be specified as inline. No diagnostic is required.

1. This assertion (that the common use case is to set the default resource once) comes from experience with Bloomberg’s large codebase. Other common use patterns that may emerge are also supported by the change proposed in this paper. [↑](#footnote-ref-1)