Project: ISO JTC1/SC22/WG21: Programming Language C++
Doc No: WG21 **P1973R1**
Date: 2020-02-12
Reply to: Nicolai Josuttis (nico@josuttis.de)
Audience: **LEWG**, LWG
Prev. Version: P1973R0

Rename "_default_init" Functions, Rev1

For C++20 we introduce to use the term "default initialization" as part of the name for some functions, such as make_unique_default_init(), make_shared_default_init(), allocate_shared_default_init().

According to the NB comment **DE002** this name in the API should be changed because it is confusing for ordinary programmers. It sounds like functions having this suffix are safer (because the naïve assumption is that they initialize data with default values, which therefore the other function don't do). but they are less safe (because default initialization does in fact ***not*** always initialize).

As with C++20 we establish this API name part as a common base for all future ways to have default initialization, we think it is important to propose a better name right now before we ship C++20.

In the corresponding LEWG discussion we had different votes. The first one in Belfast 2019 was to come up with the suffix "…uninitialized" and restrict it to trivially constructible types with the following effect:

```
auto p1 = std::make_unique_uninitialized<double[]>(1024);  // indetermined value
struct X { double data[1024]; };
auto p2 = std::make_shared_uninitialized<X>();    // indetermined value
… // set/initialize the value
```

So that we also had:

```
auto p3 = std::make_shared_uninitialized<vector<int>>();    // compile-time error
```

See P1973R0.

Then in Prague 2020, we had another LEWG vote just only to change the suffix in "…for_overwrite" so that we more express the intent than trying to use one of the many confusing terms for initialization and we don't have any constraint. As a result we can easily use these functions in generic code where we don't know whether we have a trivially copyable type:

```
auto p1 = std::make_unique_for_overwrite<T>();  // indetermined value
… // set/initialize the value
```

This code now express the intent to create something that may be not initialized so (yet) so that we should overwrite the value. And the name can't be interpreted as a safer way for object creation with guaranteed initialization with default values.

Proposed Wording

(All against N4849)

In 20.10.2 Header <memory> synopsis [memory.syn]

Rename as follows:

```
namespace std {
…
  template<class T>
  unique_ptr<T> make_unique_default_initfor_overwrite();     // T is not array
  template<class T>
  unique_ptr<T> make_unique_default_initfor_overwrite(size_t n);  // T is U[]
  template<class T, class... Args>
  unspecified make_unique_default_initfor_overwrite(Args&&...) = delete;   // T is U[N]
  …
  template<class T>
  shared_ptr<T> make_shared_default_initfor_overwrite(); // T is not U[]
  template<class T, class A>
  shared_ptr<T> allocate_shared_default_initfor_overwrite(const A& a); // T is not U[]
  template<class T>
  shared_ptr<T> make_shared_default_initfor_overwrite(size_t N); // T is U[]
  template<class T, class A>
  shared_ptr<T> allocate_shared_default_initfor_overwite(const A& a, size_t N);
                                                                  // T is U[]
};
```

20.11.1.4 Creation [unique.ptr.create]

Change:

```
template<class T> unique_ptr<T> make_unique_default_initfor_overwrite();

template<class T> unique_ptr<T> make_unique_default_initfor_overwrite(size_t n);

template<class T, class... Args> unspecified
make_unique_default_initfor_overwrite(Args&&…)= delete;
```

20.11.3.6 Creation [util.smartptr.shared.create]:

1 The common requirements that apply to all make_shared, allocate_shared, make_shared_default_initfor_overwrite, and allocate_shared_default_initfor_overwrite overloads, unless specified otherwise, are described below.

…

```
template<class T, ...>
shared_ptr<T> make_shared_default_initfor_overwrite(args);
template<class T, class A, ...>
shared_ptr<T> allocate_shared_default_initfor_overwrite(const A& a, args);
```

…

3 *Effects:* … The allocate_shared and allocate_shared_default_initfor_overwrite templates…

…

(7.8) — … initialized by make_shared_default_initfor_overwrite or allocate_shared_default_initfor_overwrite …

…

```
template<class T>
shared_ptr<T> make_shared_default_initfor_overwrite();
template<class T, class A>
shared_ptr<T> allocate_shared_default_initfor_overwrite(const A& a);
```

23 *Constraints:* T is not an array of unknown bound.

24 *Returns:* A shared_ptr to an object of type T.

25 [*Example*:

```
struct X { double data[1024]; };
shared_ptr<X> p = make_shared_default_initfor_overwrite<X>();
// shared_ptr to a default-initialized X, where each element in X::data has an indeterminate value
```

```
shared_ptr<double[1024]> q =
make_shared_default_initfor_overwrite<double[1024]>();
```
// shared_ptr to a default-initialized double[1024], where each element has an indeterminate value
—end example]

```
template<class T>
shared_ptr<T> make_shared_default_initfor_overwrite(size_t N);
template<class T, class A>
shared_ptr<T> allocate_shared_default_initfor_overwrite(const A& a, size_t N);
```

26 *Constraints:* T is an array of unknown bound.

27 *Returns:* A shared_ptr to an object of type U[N], where U is remove_extent_t<T>.

28 [*Example*:
```
shared_ptr<double[]> p = make_shared_default_initfor_overwrite<double[]>(1024);
```
// shared_ptr to a default-initialized double[1024], where each element has an indeterminate value
—end example]

In **17.3.2 Header <version> synopsis [version.syn]**
Fix the name of the feature macro as follows:

Remove the existing feature macro
```
#define __cpp_lib_smart_ptr_default_init    …   // also in <memory>
```

And introduce a new feature test macro
```
#define __cpp_lib_smart_ptr_for_overwrite    …   // also in <memory>
```

Feature Test Macro

This is fixing new features and does not need a feature test macro in itself. But as written the name of the feature macros for the corresponding new functions should change.

Acknowledgements

Thanks to a lot of people who discussed the issue, proposed information, and possible wording (especially as a heated discussion made it to a constructive result).