

Improving improved shared_ptr

1. Overview

This paper propose the introduction of a new terminology in the description of templates `shared_ptr` and `weak_ptr`, as well as improved specifications of a few related functions. The changes affect clauses 20.6.6.2¹ [util.smartptr.shared] and 20.6.6.3 [util.smartptr.weak].

2. Motivation

Paper N2351 “Improving `shared_ptr` for C++0x”, which was incorporated in the draft standard N2369, introduced a new constructor to the class template `shared_ptr`, the so called “aliasing” constructor. While there is consensus about the increase in expressive power provided by the new constructor, a discussion on comp.std.c++ pointed out that the impact of the addition on the rest of the specification has been underestimated. The author believes that the wording may be improved while maintaining the new feature. The main issues are:

1. N2351 introduces a new concept of “empty `shared_ptr` with non-NULL stored pointer”. The concept is quite hard to understand and may be a source of confusion for beginners
2. there is no way to observe the “stored pointer” of an empty `shared_ptr`, as the observer `get()` is defined to return 0 when the pointer is empty.
3. several functions, for example the `*_pointer_cast` and the move constructor, do not specify the value of the stored pointer when an empty `shared_ptr` is created
4. the concept of “empty `shared_ptr`” is poorly defined (in fact it's not defined at all)

The key to this proposal is the introduction in normative text of the possibility of having `shared_ptr` instances *with no ownership*. Such concept effectively replaces the concept of empty `shared_ptr`. A `shared_ptr` with no ownership has the same semantic as a regular pointer, so it's not hard to imagine it taking any pointer value, including NULL.

In addition, the author would want to take the opportunity to remove the unnecessary overuse of the italic all over the place and to replace post-condition like `use_count() == r.use_count()` with more explicit statements about the ownership (no change in semantic is proposed, the new wording implies the old one, but better describes the intent).

The author also feels that the “ownership” terminology is inappropriate when speaking about a `weak_ptr`, which, allegedly, “stores a weak reference to an object that is already managed by a `shared_ptr`.” However, in order to improve chances of getting this proposal approved, changes in this direction are not suggested here, but are deferred to a separate proposal.

The author acknowledges Joe Gottman for having first raised the issue on comp.std.c++.

3. Proposed text

§20.6.6.2 [util.smartptr.shared]

- Paragraph 1, add at the end:

As a degenerate case, a `shared_ptr` object can have no ownership, such an object implements the

¹ All numeric references are relative to the current draft, at the time of writing, paper N2369.

semantic of a regular pointer.

§20.6.6.2.1 [util.smartptr.shared.const]

- Paragraph 1, replace:

Effects: Constructs an *empty* `shared_ptr` object.

with:

Effects: Constructs a `shared_ptr` object that has no ownership and stores the null pointer.

- Paragraph 14, replace:

Effects: Constructs a `shared_ptr` instance that stores `p` and *shares ownership* with `r`.

with²

Effects: Constructs a `shared_ptr` instance that has the same ownership as `r` and stores `p`.

- Paragraph 15, replace:

Postconditions: `get() == p && use_count() == r.use_count()`

with:

Postconditions: `get() == p, *this` and `r` share ownership or both have no ownership.

- Remove paragraph 18³ entirely:

[Note: this constructor allows creation of an empty `shared_ptr` instance with a non-NULL stored pointer. —end note]

- Paragraph 20, replace:

Effects: If `r` is *empty*, constructs an *empty* `shared_ptr` object; otherwise, constructs a `shared_ptr` object that *shares ownership* with `r`.

with:

Effects: Constructs a `shared_ptr` object that has the same ownership as `r` and stores a copy of the pointer stored in `r`.

- Paragraph 21, replace:

Postconditions: `get() == r.get() && use_count() == r.use_count()`.

with:

Postconditions: `get() == r.get(), *this` and `r` share ownership or both have no ownership.

- Paragraph 25, replace:

² The expression “has the same ownership” has been consistently preferred to “shares ownership” in a few contexts, because the former is suitable also when the pointers have no ownership, while the latter would have implied ownership.

³ The note in this paragraph is the main motivator of the entire proposal. As the proposal defines the concept of “no ownership” in normative text, the note is no longer necessary.

Postconditions: `*this` shall contain the old value of `r`. `r` shall be *empty*.

with⁴:

Postconditions: `get() == r.get()`. The ownership of `r` is transferred to `*this`, `r` shall have no ownership.

- Paragraph 29, replace:

Postconditions: `use_count() == r.use_count()`.

with:

Postconditions: `r` and `*this` share ownership.

§20.6.6.2.2 [util.smartptr.shared.dest]

- Paragraph 1, replace:

— If `*this` is *empty* or *shares ownership* with another `shared_ptr` instance (`use_count() > 1`), there are no side effects.

with:

— If `*this` has no ownership or *shares ownership* with another `shared_ptr` instance (`use_count() > 1`), there are no side effects.

§20.6.6.2.5 [util.smartptr.shared.obs]

- Paragraph 1, replace:

Returns: the stored pointer. Returns a null pointer if `*this` is *empty*.

with⁵:

Returns: the stored pointer.

- Paragraph 10, replace:

Returns: the number of `shared_ptr` objects, `*this` included, that *share ownership* with `*this`, or 0 when `*this` is *empty*.

with:

Returns: 0 if `*this` has no ownership; otherwise, the number of `shared_ptr` objects, `*this` included, that *share ownership* with `*this`.

- Paragraph 15, remove from note text⁶:

If you are using `unique()` to implement copy on write, do not rely on a specific value when `get() == 0`.

⁴ This wording clarifies the value of `r.get()` after the move.

⁵ There is a change in semantic here. See motivation.

⁶ Having changed the semantic of `get()`, whose return value is now unaffected by the value of `use_count()`, this phrase no longer makes sense.

§20.6.6.2.7 [util.smartptr.shared.cmp]

- Paragraph 5, replace:

two `shared_ptr` instances are equivalent if and only if they *share ownership* or are both *empty*.

with:

two `shared_ptr` instances are equivalent if and only if they share ownership or both have no ownership.

§20.6.6.2.10 [util.smartptr.shared.cast]

- Paragraph 2, replace:

Returns: If `r` is *empty*, an *empty* `shared_ptr<T>`; otherwise, a `shared_ptr<T>` object that stores `static_cast<T*>(r.get())` and *shares ownership* with `r`.

with⁷:

Returns: `shared_ptr<T>(r, static_cast<T*>(r.get()))`.

- Paragraph 6, replace:

Returns:

- When `dynamic_cast<T*>(r.get())` returns a nonzero value, a `shared_ptr<T>` object that stores a copy of it and shares ownership with `r`;
- Otherwise, an empty `shared_ptr<T>` object.

with⁸:

Returns:

- If `p = dynamic_cast<T*>(r.get())` is a non-null pointer, `shared_ptr<T>(r, p)`;
- Otherwise, `shared_ptr<T>()`.

- Paragraph 10, replace:

Returns: If `r` is *empty*, an *empty* `shared_ptr<T>`; otherwise, a `shared_ptr<T>` object that stores `const_cast<T*>(r.get())` and *shares ownership* with `r`.

with⁹:

Returns: `shared_ptr<T>(r, const_cast<T*>(r.get()))`.

§20.6.6.3.1 [util.smartptr.weak.const]

- Paragraph 1, replace:

Effects: Constructs an empty `weak_ptr` object.

with:

⁷ This wording clarifies the value of the stored pointer when `r` has no ownership.

⁸ This wording clarifies the value of the stored pointer when `r` has no ownership. This alternative wording has also been considered:

Returns: `shared_ptr<T>(r, dynamic_cast<T*>(r.get()))`.

⁹ This wording clarifies the value of the stored pointer when `r` has no ownership.

Effects: Constructs a **weak_ptr** object with no ownership. [Note: the value of the stored pointer is left unspecified. In fact the user has no way to access it. —end note]

- Paragraph 5, replace:

Effects: If **r** is *empty*, constructs an *empty weak_ptr* object; otherwise, constructs a **weak_ptr** object that *shares ownership* with **r** and stores a copy of the pointer stored in **r**

with:

Effects: Constructs a **weak_ptr** object that has the same ownership as **r** and stores a copy of the pointer stored in **r**.

- Paragraph 6, replace:

Postconditions: `use_count() == r.use_count()`.

with:

Postconditions: ***this** and **r** share ownership or both have no ownership.

§20.6.6.3.5 [util.smartptr.weak.obs]

- Paragraph 1, replace:

Returns: 0 if ***this** is *empty*; otherwise, the number of **shared_ptr** instances that *share ownership* with ***this**.

with:

Returns: 0 if ***this** has no ownership; otherwise, the number of **shared_ptr** instances that share ownership with ***this**.

§20.6.6.3.6 [util.smartptr.weak.cmp]

- Paragraph 1, replace:

two **weak_ptr** instances are equivalent if and only if they *share ownership* or are *both empty*.

with:

two **weak_ptr** instances are equivalent if and only if they share ownership or both have no ownership.

4.Additional proposed text

It's clear from the definition that a **shared_ptr** has no ownership if and only if `use_count() == 0`. However, member function `use_count()` is meant for debugging purposes only and may not be efficient. On the other hand, it's quite possible that the implementation is able to detect whether a **shared_ptr** has ownership or not in an efficient way. Given that **shared_ptr** with no ownership are proposed to be first-class citizens, the author believes it would be good to expose such efficient detection path.

§20.6.6.2 [util.smartptr.shared]

Add, after the declaration of member function `unique()`:

```
bool has_ownership() const;
```

§20.6.6.2.5 [util.smartptr.shared.obs]

Add, after the definition of member function `unique()`:

```
bool has_ownership() const;
```

Returns: `use_count() != 0`.

Throws: nothing.

[Note: `has_ownership()` may be faster than `use_count()`. —end note]