In practice, when implementing **perfect forwarding**, making a mistake in any one of these facets will result in not having a forwarding reference and compilation errors. Being unable to state clearly the intent to have a forwarding reference makes misuse by developers more likely and compilation errors more difficult to diagnose.

Value categories are a moving target

C++03 had just *lvalues* and *rvalues*. In the original design of C++11, the only *xvalues* were once *lvalues*. In C++14, members of *prvalue* user-defined types also became *xvalues*. In C++17 even more *prvalues* were identified as *xvalues*. Some of these changes have been adopted as defect reports against older standards, and some have introduced subtle changes in behavior between language standards.

In any case, the progression is in one direction: there were no *rvalues* in C++03 that were not *prvalues* in C++11, and then the demarcation between *prvalue* and *xvalue* continued to drift so that the categories of non*lvalues* that were deemed to be *xvalues* grew. The criterion now is *not* that an *xvalue* is a non*lvalue* that is reachable but that it is a non*lvalue* that refers to an object in memory; a *prvalue* now becomes everything else that isn't an *lvalue* and, unless **`void`**, must be a complete type. Once something becomes an *xvalue* in the Standard, it can never go back to being a *prvalue*. Understanding the evolution is helpful to understanding how the C++ language is evolving; see the *Appendix — The evolution of value categories* on page 813.

Overall, what the literature has lacked and the Standard's evolution has made difficult to understand is a clear designation of what the value categories are and what their purpose is. The realization that the *xvalue* category needed to encompass all objects whose data is no longer needed — whether due to being a temporary whose lifetime is ending or due to an explicit cast in code — took a great deal of time to clarify, with various edge cases continuing to surface.[30]

Standard Library requirements on a moved-from object are overly strict

By Sean Parent

Given an object, `rv`, of type `T` that has been moved from, the C++14[31] Standard specifies the required postconditions of a moved-from object[32]:

> rv's state is unspecified [*Note*: rv must still meet the requirements of the library component that is using it. The operations listed in those requirements must work as specified whether rv has been moved from or not. — *end note*]

[30] Though the distinction between a *prvalue* and an *xvalue* is largely academic prior to C++17, with the adoption of proposal P0135R0 (**smith15c**), knowing the difference becomes important in light of guaranteed copy elision and, in particular, **mandatory RVO** for *prvalues*.

[31] Similar wording having the same intent appears in every version of the C++ Standard since C++11.

[32] **iso14**, Table 20, p. 427