There are many ways in which an object might or might not provide a nonthrowing *move* operation. As `C2` in the example above suggests, even a C++03 class that happened to decorate its explicitly declared copy constructor with **`throw()`** would automatically satisfy the requirements of a nonthrowing *move*. A more likely scenario for a class designed prior to C++11 to wind up with a nonthrowing *move* operation is that it followed the **rule of zero**, thereby allowing each of the special member functions to be generated. In this case, all that might be needed to generate a nonthrowing *move* constructor is simply to recompile it under C++11! The takeaway here is that, irrespective of how a type is implemented, we can use the **`noexcept`** operator in combination with `std::move` to reliably determine, at compile time, whether an object of a given type *may* throw when we ask it to move. Note that while it is typical for C++03-targeted code to either have both copy and move operations nonthrowing, or both potentially throwing, a C++03 template instantiated with a C++11 type might have the **`noexcept`** specification of the copy and move operations be distinct, following those of the template argument type:

```
template <typename T>
struct NamedValue
{
    const char* d_name;
    T           d_value;
};
```

While `NamedValue` might be a class template shipping from a C++03-targeted library, the copy constructor for this class will clearly be **`noexcept(false)`**, but the move constructor might be throwing or not based solely on the properties of the template argument, `T`.

Applying the `noexcept` operator to functions in the C Standard Library

According to the C++03 Standard[7]:

> None of the functions from the Standard C library shall report an error by throwing an exception, unless it calls a program-supplied function that throws an exception.

This paragraph is accompanied by a footnote[8]:

> That is, the C library functions all have a `throw()` *exception-specification*. This allows implementations to make performance optimizations based on the absence of exceptions at runtime.

Note that this footnote applies only to functions in the C Standard Library, not to arbitrary functions having **`extern "C"`** linkage. It is not clear what the normative implications of the footnote might be, as it seems to be a non-normative note clarifying something not

[7] **iso03**, section 17.4.4.8, "Restrictions on exception handling," paragraph 2, p. 332
[8] Ibid., footnote 176, p. 332