What’s up with error C4838: a WinRT type cannot be a member of a union, and how can I work around it?

devblogs.microsoft.com/oldnewthing/20200914-00

September 14, 2020

Raymond Chen

If you try to put a Windows Runtime type inside a `std::optional` from C++/CX, you get this error:

```
std::optional<::Windows::Foundation::Rect> optRect;

error C2848: 'std::_Optional_destruct_base<_Ty,true>::Value': A WinRT type cannot be
a member of a union
```

Why can’t a WinRT type be a member of a union?

The C++/CX language extension was written back when the latest C++ standard was C++03, and in C++03, unions could not (among other things) contain objects that had a nontrivial destructor.

Windows Runtime types usually have destructors: Reference types use hat pointers which release the underlying ABI object at destruction. And structures might contain `String^`, which also entails a nontrivial destructor.

C++/CX just gives up and says, “Nope, not gonna do that.” It doesn’t bother to check whether the structure in question actually has a nontrivial destructor; it just assumes that it does, and the compiler won’t let you put it into a union. C++/CX won’t even let you put a Windows Runtime enum in a union!

Fortunately, C++/WinRT doesn’t have this problem. C++/WinRT is standard C++, and the compiler can see that the `Rect` structure is just a bunch of `float`s with no special destructor. Furthermore, C++/WinRT can take advantage of advances in the C++ language, such as permitting objects with destructors in unions.

```
// compiles just fine
std::optional<winrt::Windows::Foundation::Rect> optRect;
```

Just one of many reasons why C++/CX is deprecated in favor of C++/WinRT.

Bonus chatter: If you really need to put a C++/CX `Rect` into a union, you can wrap it inside another structure first.

```
struct RectStruct
{
  ::Windows::Foundation::Rect rect;
};

std::optional<RectStruct> optRect;
```

Raymond Chen

Follow