

C++ coroutines: no callable 'await_resume' function found for type

 devblogs.microsoft.com/oldnewthing/20191217-00

December 17, 2019



Raymond Chen

You try to `co_await` something and get the error message

```
no callable 'await_resume' (or 'await_ready' or 'await_suspend') function found for type  
'Expression'
```

What does this mean?

Recall how the compiler generates code for `co_await` :

calculate `x`
obtain `awaiter`

```
co_await    if (!awaiter.await_ready())  
            {  
            save state for resumption  
            if (awaiter.await_suspend(handle))  
            {  
            return to caller  
            }  
            [Invoking the handle resumes execution here]  
            }  
            restore state after resumption  
            }  
            result = awaiter.await_resume();
```

execution continues

The “obtain awaiter” step always succeeds because of rule 3:

1. (We’re not ready to talk about step 1 yet.)
2. (We’re not ready to talk about step 2 yet.)
3. Otherwise, `x` is its own awaiter.

Even if the mysterious first two steps fail, the third always succeeds.

The other parts of the code generation require that the awaiter implement methods named `await_ready`, `await_suspend`, and `await_resume`. If any of them is missing, the compiler will generate a corresponding message.

And if *all of them* are missing, then the one the compiler complains about first is unpredictable. The current implementation of the compiler looks for `await_resume` first, but that is not contractual, and future versions may look for one of the other two methods first.

One of the reasons you may get this error is that you are awaiting something that simply isn't awaitable.

```
struct something { };  
  
co_await something();
```

The `something` structure doesn't have any of the required methods for being an awaitable object, so you will get an error.

Another reason you may get this error is that you were expecting one of the first two steps (which we haven't talked about yet) to produce an awaiter, but they failed. We'll investigate this after we learn about the mysterious step 2.

Raymond Chen

Follow

