What happens if C++/WinRT is unable to resume execution on a dispatcher thread?

devblogs.microsoft.com/oldnewthing/20250721-00/?p=111396

July 21, 2025

The C++/WinRT library provides the `winrt::resume_foreground()` function for resuming a coroutine on the foreground thread of a dispatcher. It supports three types of dispatchers: `Windows.UI.Core.CoreDispatcher`, `Windows.System.DispatcherQueue`, `Microsoft.UI.Dispatching.DispatcherQueue`.

But what happens if the thread switch fails? This happens if the dispatcher thread is no longer accepting new work, either because it is shutting down, or because it no longer exists at all.

If a `CoreDispatcher` is unable to dispatch a work item, it silently releases the work item without executing it. For `winrt::resume_foreground()`, that means that your coroutine appears to hang. The resumption never happens, nothing in the coroutine executes beyond the `co_await winrt::resume_foreground(coreDispatcher)`, and your coroutine is leaked.

If either of the `DispatcherQueue` objects is unable to dispatch a work item, then the `TryEnqueue()` method returns `false`. The `winrt::resume_foreground()` function propagates this return value as the return value of `co_await`. This means that you are expected to check the return value to find out whether the thread switch is successful.

```
if (co_await winrt::resume_foreground(dispatcherQueue))
{
    // Yay, executing on the dispatcher thread
} else
{
    // Rats, was not able to switch threads
}
```

In practice, nobody[1] checks the return value of `co_await winrt::resume_foreground(dispatcherQueue)`, so the result is that the code continues running on the wrong thread. It thinks that it's on the dispatcher thread, but it's not.

This sure looks like a disaster. We'll find an escape route next time.

[1] I have yet to find any code in any code base that checks the return value. Not even the C++/WinRT unit tests check the return value!