Gotcha: Be careful how you shut down your dispatcher queues

devblogs.microsoft.com/oldnewthing/20240223-00

February 23, 2024

Raymond Chen

Last time, we learned that it is important to shut down your dispatcher queues. This is done by calling `DispatcherQueueController.ShutdownQueueAsync()`. The operation completes after the queue has shut down and its thread has become useless. But there's a catch.

The catch is the case where you await a call to `DispatcherQueueController.ShutdownQueue-Async()` *from the dispatcher queue thread*. In this case, you are asking the current thread to shut itself down, and then expecting to regain control on the very thread that has been shut down.

You are now skating on thin ice.

```
namespace winrt
{
    using namespace winrt::Windows::Foundation;
    using namespace winrt::Windows::System;
}

winrt::IAsyncAction Example()
{
    // Create a dispatcher queue controller
    // for a new dispatcher queue thread
    auto controller =
        winrt::DispatcherQueueController::CreateOnDedicatedThread();

    // To demonstrate the problem, switch to the dispatcher queue thread
    co_await winrt::resume_foreground(controller.DispatcherQueue());

    // Shut down the dispatcher queue while we're on its thread (!)
    co_await controller.ShutdownQueueAsync();
}
```

Both C# and C++/WinRT have a default policy of resuming execution after an `await` / `co_await` in the same COM apartment in which the await started.[1] If you start the await on the dispatcher queue thread, then execution resumes on a dispatcher queue thread that has already shut down and is about to exit.

Even though the thread is about to exit, it is still a single-threaded COM apartment (at least for a little while longer), so any `await` / `co_await` / PPL task is going to try to return to that single-thread COM apartment which has been destroyed. Furthermore, any operations you perform that involve multiple threads may fail because they no longer have a way to get back to the dispatcher queue thread.

```
winrt::IAsyncAction Example()
{
    // Create a dispatcher queue controller
    // for a new dispatcher queue thread
    auto controller =
        winrt::DispatcherQueueController::CreateOnDedicatedThread();

    // To demonstrate the problem, switch to the dispatcher queue thread
    co_await winrt::resume_foreground(controller.DispatcherQueue());

    // Shut down the dispatcher queue while we're on its thread (!)
    co_await controller.ShutdownQueueAsync();

    // Oh no, this will try to resume on the dispatcher queue thread
    // after it has been shut down.
    co_await winrt::Launcher::LaunchUriAsync(
        winrt::Uri{L"https://microsoft.com"});

    DoSomeMoreStuff();
}
```

That call to `LaunchUriAsync` is in big trouble. After the launch completes, it's going to try to return to the dispatcher queue thread, which likely no longer exists by the time it completes. The code will throw an exception representing `RPC_E_DISCONNECTED`, and nothing after the `co_await` will execute.

Probably the best way to avoid this problem is to shut down the dispatcher queue from a background thread. That way, you never find yourself running on a thread that is slated for death.

```
winrt::IAsyncAction Example()
{
    // Create a dispatcher queue controller
    // for a new dispatcher queue thread
    auto controller =
        winrt::DispatcherQueueController::CreateOnDedicatedThread();

    // Switch to the dispatcher queue thread
    co_await winrt::resume_foreground(controller.DispatcherQueue());

    // Get off the dispatcher queue thread
    // before shutting it down.
    co_await winrt::resume_background();

    // Shut down the dispatcher queue thread from a safe distance.
    co_await controller.ShutdownQueueAsync();

    // This code is now safely running on a background thread.
    co_await winrt::Launcher::LaunchUriAsync(
        winrt::Uri{L"https://microsoft.com"});

    DoSomeMoreStuff();
}
```

[1] PPL also has the default policy of resuming execution in the same COM apartment provided the task chain started with a Windows Runtime `IAsyncXxx`.