Asking for a DispatcherQueue from a GUI thread you created

devblogs.microsoft.com/oldnewthing/20240509-52

May 9, 2024

Raymond Chen

The most common way to create a `DispatcherQueue` is to ask for one to be created on a dedicated thread by calling `DispatcherQueueController.CreateOnDedicatedThread`. But another way to create a `DispatcherQueue` is to ask for one to be attached to an existing message queue.

```
DispatcherQueueOptions options{
    sizeof(options),
    DQTYPE_THREAD_CURRENT,
    DQTAT_COM_NONE
};

winrt::com_ptr<ABI::Windows::System::IDispatcherQueueController> controller;
winrt::check_hresult(CreateDispatcherQueueController(options, controller.put()));
```

If you ask `CreateDispatcherQueueController` to create a a `DQTYPE_THREAD_CURRENT`, then it will create a dispatcher queue that is attached to the current thread's message queue. This dispatcher queue does not have its own message pump; it's relying on the thread's existing message pump to do the work.

Attaching a dispatcher queue to an existing message queue is convenient if you have existing Win32-style code, and you want to integrate it with components that need a dispatcher queue: You can create your Win32 thread in the traditional way, set up a message pump, and then ask for a dispatcher queue to be attached to it.

Note that when you do this, it becomes your responsibility to keep the thread alive and pumping messages until the dispatcher queue has been shut down. There are a few ways of knowing when the shutdown has completed.

One is to await the `ShutdownQueueAsync` call. Its `IAsyncAction` completes when the shutdown is complete. After the completion of that operation, you can post a message to the message queue to tell it that it's safe to exit. (You might even use `PostQuitMessage` to do that.)

```
winrt::fire_and_forget OnDestroy()
{
    co_await m_controller.ShutdownQueueAsync();
    PostQuitMessage(0);
}
```

Another way is to register a handler for the dispatcher queue's `ShutdownCompleted` event and post the "safe to exit" message from there.

```
void OnDestroy()
{
    m_controller.ShutdownCompleted([](auto&&, auto&&) {
        PostQuitMessage(0);
    });
    m_controller.ShutdownQueueAsync();
}
```

This trick works because all dispatcher queue events are raised from the dispatcher queue thread, so the `PostQuitMessage` posts back into the correct thread.