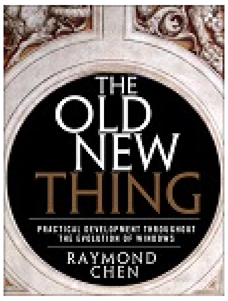
How can I wait for Clipboard History to recognize a clipboard change before I change it again?

devblogs.microsoft.com/oldnewthing/20250509-00/?p=111169

May 9, 2025



Last time, we learned why the clipboard history service doesn’t capture rapid changes to clipboard contents. This was a problem for a program whose express purpose was to “load up” the clipboard history. How can that program check that their change made it into the history before changing the clipboard again?

The secret is the `Clipboard.HistoryChanged` event, which tells you that there has been a change to the clipboard history. It could be that a new item was added, or that the user deleted an item from the history, or that new history items roamed to the system, or that the user decided to clear all of the history.

For simplicity, we’ll assume that the only source of clipboard history changes is the ingestion of new clipboard data that our program added. (If you wanted to be sure, you can look at the clipboard history and see if your string is in it.)

```
// All error checking elided for expository purposes
#include <windows.h>
#include <winrt/Windows.ApplicationModel.DataTransfer.h>
#include <winrt/Windows.Foundation.h>
#include <winrt/Windows.System.h>

namespace winrt
{
    using namespace winrt::Windows::ApplicationModel::DataTransfer;
    using namespace winrt::Windows::Foundation;
    using namespace winrt::Windows::System;
}

void SetClipboardText(HWND hwnd, PCWSTR text)
{
    OpenClipboard(hwnd);
    EmptyClipboard();
    auto size = sizeof(wchar_t) * (1 + wcslen(text));
    auto clipData = GlobalAlloc(GMEM_MOVEABLE, size);
    auto buffer = (LPWSTR)GlobalLock(clipData);
    strcpy_s(buffer, size, text);
    GlobalUnlock(clipData);
    SetClipboardData(CF_UNICODETEXT, clipData));
    CloseClipboard();
}

// Put these strings in the clipboard history for quick access.
static constexpr PCWSTR messages[] = {
    L"314159", // the bug number we want to edit
    L"e83c5163316f89bfbde7d9ab23ca2e25604af290", // the commit to link the bug to
    L"Widget polarity was set incorrectly.", // the comment to add
};

winrt::IAsyncAction Sample(winrt::DispatcherQueue queue)
{
    co_await winrt::resume_foreground(queue);

    if (!winrt::Clipboard::IsHistoryEnabled()) {
        // Oops
        co_return;
    }

    winrt::handle changed(winrt::check_pointer(
        CreateEventW(nullptr, FALSE, FALSE, nullptr)));

    auto historyChanged = winrt::Clipboard::HistoryChanged(winrt::auto_revoke,
        [h = changed.get()](auto&&, auto&&) {
        SetEvent(h);
    });

    auto tempWindow = CreateWindowExW(0, L"static", nullptr, WS_POPUPWINDOW,
            0, 0, 0, 0, nullptr, nullptr, nullptr, nullptr);

    for (auto message : messages) {
        SetClipboardText(tempWindow, message);
        co_await winrt::resume_on_signal(changed.get());
        co_await winrt::resume_foreground(queue);
    }
    DestroyWindow(tempWindow);
}

int wmain([[maybe_unused]] int argc,
          [[maybe_unused]] wchar_t* argv[])
{
    winrt::init_apartment();
    {
        auto controller = winrt::DispatcherQueueController::
                                   CreateOnDedicatedThread();
        Sample(controller.DispatcherQueue()).get();
        controller.ShutdownQueueAsync().get();
    }
    winrt::uninit_apartment();
    return 0;
}
```

The Windows Runtime clipboard requires access from a UI thread, so we spin up a dispatcher queue to serve as our UI thread. We do it as a separate thread so we can wait on it from the `wmain` function. (The `main` and `wmain` functions cannot be coroutines.)

On the dispatcher queue thread, we first check whether clipboard history is even enabled. If not, then we bail out, because all of this work is pointless, and the program would hang if we tried, because we'd be waiting for a history change event that never happens.

Once we confirm that clipboard history is enabled, we create a kernel event object that we will use to tell us when the clipboard history has changed, and register an event handler that sets the event object. Then we go into our loop, and after setting each string, we wait for the signal before moving on to the next one.

There are still some gaps here: If the user disables clipboard history while the program is running, it will get halfway through the list and then hang waiting for an event that will never occur. But those are fine tuning steps. The essential point is that we are using the history change notification to tell us that the clipboard history service has processed the new clipboard text, so we can move on to the next one. This is just a sketch of the solution.