The subtleties of CreateStreamOnHGlobal, part 2: Suppressing the deletion of an unknown HGLOBAL

devblogs.microsoft.com/oldnewthing/20210929-00

September 29, 2021

Raymond Chen

Last time, we looked at the easy case of the `CreateStreamOnHGlobal` function, where you just hand an `HGLOBAL` to the stream and wipe your hands of it.

The weird case is where you pass `fDeleteOnRelease = FALSE`. For now, let's look at the case where you pass a null `HGLOBAL`, but also pass `fDeleteOnRelease = FALSE`, indicating that you want the stream to create its own `HGLOBAL`, but not to destroy it on destruction.

The typical use case for this pattern is where you create a stream, write to it, and then extract all the data in the form of an `HGLOBAL`. To get the `HGLOBAL` back out, you must call `Get-HGlobalFromStream`. If you don't do that, then the `HGLOBAL` inside the stream is leaked, and you have no way to rescue it.

```
// First, the version without RAII types.
HGLOBAL CreateHGlobalFromStuff()
{
    HGLOBAL hglob = nullptr;
    IStream* stream = nullptr;

    if (SUCCEEDED(
        CreateStreamOnHGlobal(nullptr, FALSE, &stream))) {
        if (FAILED(GetHGlobalFromStream(stream, &hglob))) {
            __fastfail(FAST_FAIL_FATAL_APP_EXIT);
        }
        WriteStuffToStream(stream);
        stream->Release();
    }
    return hglob;
}

void Sample()
{
    HGLOBAL hglob = CreateHGlobalFromStuff();
    if (hglob) {
        DoStuffWith(hglob); // maybe put it on the clipboard
        GlobalFree(hglob);
    }
}
```

And with RAII:

```
wil::unique_hglobal CreateHGlobalFromStuff()
{
    wil::unique_hglobal hglob;
    wil::com_ptr<IStream> stream;

    THROW_IF_FAILED(
        CreateStreamOnHGlobal(nullptr, FALSE, &stream));

    FAIL_FAST_IF_FAILED(
        GetHGlobalFromStream(stream.get(), &hglob));

    WriteStuffToStream(stream.get());

    return hglob;
}

void Sample()
{
    wil::unique_hglobal = CreateHGlobalFromStuff();
    DoStuffWith(hglob.get()); // maybe put it on the clipboard
}
```

The idea here is that you create an `HGLOBAL`-based stream with no initial memory block, which means that the stream object creates its own empty one. From that, you immediately get the inner `HGLOBAL` so you can access the data that is being managed. You do this immediately so that the memory block is held in an RAII type so it won't be leaked if something goes wrong later.

You then write stuff to the stream, which causes it to go into the `HGLOBAL`. You then throw the stream away and keep the `HGLOBAL`. The stream's destruction does not destroy the `HGLOBAL` because you passed `fDeleteOnRelease = FALSE`.

Note that there is a bit of a scary place: If `GetHGlobalFromStream` fails, we are kind of stuck. The `HGLOBAL` inside the stream is going to be leaked, and there's nothing we can do about it. Fortunately, `GetHGlobalFromStream` always succeeds if the stream it was given came from the `CreateStreamOnHGlobal` function.

The caller takes the resulting `HGLOBAL`, does something with it, and then frees the `HGLOBAL`.

It is crucially important that you keep your eye on the stream. You have to be sure that nobody has extended the lifetime of the stream (by calling `AddRef`), because you just freed the `HGLOBAL`, and any attempt to use the stream beyond that point will be a use-after-free bug.

One mistake people make with passing `fDeleteOnRelease = FALSE` is forgetting that they are now responsible for freeing the `HGLOBAL` that is inside the stream. If you passed an `HGLOBAL` of `NULL` when creating the stream, then you must call `GetHGlobalFromStream` to find out what `HGLOBAL` ended up being allocated for the stream, because it's your responsibility to free it.

```
// Do not use! This code leaks the global handle.

IStream* stream;
if (SUCCEEDED(CreateStreamOnHGlobal(nullptr,
        FALSE, &stream)) {
    WriteStuffToStream(stream);
    ReadStuffFromStream(stream);
    stream->Release();
}
```

This code is under the the mistaken belief that the `fDeleteOnRelease` parameter controls not whether the stream frees its internal `HGLOBAL` but rather whether it frees the `HGLOBAL` that was passed in. And since they passed `nullptr` as the incoming `HGLOBAL`, they certainly don't want the stream to try to free a null pointer. But that's not what `fDeleteOn-Release` means.

In the case where you aren't interested in extending the lifetime of the inner `HGLOBAL` beyond the lifetime of the stream, just pass `fDeleteOnRelease = TRUE` and let the `HGLOBAL` be destroyed as part of the natural destruction of the stream.

Next time, we'll look at what happens if you provide an initial `HGLOBAL` and combine it with `fDeleteOnRelease = FALSE`.

Raymond Chen

Follow