What happened to the custom exception description I threw from a C++/WinRT IAsyncAction?

devblogs.microsoft.com/oldnewthing/20231116-00

November 16, 2023

Raymond Chen

A customer designed one of their methods such that it returns the answer on success, and on failure, it throws a Windows Runtime exception whose custom description is a JSON payload that describes what went wrong.

```
winrt::IAsyncAction GetItemNameAsync()
{
    auto lifetime = get_strong();
    auto result = co_await m_httpClient.TryGetStringAsync(m_uri);
    if (!result.Succeeded()) {
        // Network request failed. Return an empty object.
        throw winrt::hresult_error(result.ExtendedError(),
            L"{ }");
    }

    auto o = winrt::JsonObject::Parse(result.Value());

    auto name = o.TryLookup(L"name");
    if (!name) {
        // If there is no name, then fail and include
        // the JSON response (which will contain error details).
        throw winrt::hresult_error(E_FAIL, result.Value());
    }
    return name;
}
```

They found that sometimes the custom description they attached to the Windows Runtime exception was being truncated.

```
// consumer

try {
    auto name = co_await GetItemNameAsync(itemId);
    ⟦ use the name ⟧
} catch (winrt::hresult_error const& e) {
    // Sometimes this parse fails with bad JSON.
    auto o = winrt::JsonObject::Parse(e.message());
    ⟦ study the JSON to see what went wrong ⟧
}
```

They wondered if maybe they should throw some other type of exception to encode the JSON information.

We learned a little while ago that the custom description associated with a Windows Runtime exception is a courtesy and may not survive transport across the ABI. In particular, the custom description is not intended for programmatic use. Its intended use is to inform the developer of additional information that may help diagnose the problem. For example, for an `E_INVALIDARG` failure, it may contain the name of the invalid parameter. But it's not intended as a reliable side channel.

The `RoOriginateError` function take a description for an about-to-be-returned failure and saves it in the per-thread side channel data. However, it saves only the first 250-ish characters of the message. I don't know why it sets an artificial limit, but I have some ideas. For one thing, the description is already advisory anyway, and it's supposed to be a brief message to help debug the problem. You weren't expected to be passing large amounts of information, and perhaps the fixed limit is to prevent problems if somebody decides to pass a 1 gigabyte string as an "error description".

At any rate, you can't put functionally significant information in the error description because it may not even survive at all.

What the component should do is report the JSON error message as part of a compound return value.

```
namespace Contoso
{
    runtimeclass ItemNameResult
    {
        Boolean Succeeded { get; };
        String Name { get; };
        String ErrorJson { get; };
    }
}
```

We can then use this `ItemNameResult` as the return value of `GetItemNameAsync`.

```
namespace winrt::Contoso::implementation::ItemNameResult
{
    struct ItemNameResult : ItemNameResultT<ItemNameResult>
    {
        static auto CreateName(hstring const& name)
        {
            return make<ItemNameResult>(true, name, L"");
        }
        static auto CreateError(hstring const& errorJson)
        {
            return make<ItemNameResult>(false, L"", errorJson);
        }

        auto Succeeded() { return m_succeeded; }
        auto Name() { return m_name; }
        auto ErrorJson() { return m_errorJson; }

    private:
        ItemNameResult(bool succeeeded,
                       hstring const& name,
                       hstring const& errorJson) :
            m_succeeded(succeeded), m_name(name), m_errorJson(errorJson) {}

        bool m_succeeded;
        hstring m_name;
        hstring m_errorJson;
    };
}

winrt::IAsyncOperation<Contoso::ItemNameResult> GetItemNameAsync()
{
    auto lifetime = get_strong();
    auto result = co_await m_httpClient.TryGetStringAsync(m_uri);
    if (!result.Succeeded()) {
        // Network request failed.
        return ItemNameResult::CreateError(L"{ }");
    }

    auto o = winrt::JsonObject::Parse(result.Value());

    auto name = o.TryLookup(L"name");
    if (!name) {
        // If there is no name, then fail and include
        // the JSON response (which will contain error details).
        return ItemNameResult::CreateError(result.Value());
    }
    return ItemNameResult::CreateName(name);
}

// consumer

auto result = co_await GetItemNameAsync(itemId);
```

```
if (result.Succeeded()) {
    auto name = result.Name();
    〚 use the name 〛
} else {
    auto o = winrt::JsonObject::Parse(result.ErrorJson());
    〚 study the JSON to see what went wrong 〛
}
```