Accessing a member of a Windows Runtime class raises an InvalidCastException / throws a hresult_no_interface, what does this mean?

devblogs.microsoft.com/oldnewthing/20200324-00

March 24, 2020

Raymond Chen

You're minding your own business and you decide to call a method on some object. Everything compiles fine, but it crashes at runtime with `InvalidCastException`:

```
// C#
var options = new LauncherOptions();
options.IgnoreAppUriHandlers = true; // System.InvalidCastException

// C++/WinRT
LauncherOptions options;
options.IgnoreAppUriHandlers(true); // winrt::hresult_no_interface

// C++/CX
auto options = ref new LauncherOptions();
options->IgnoreAppUriHandlers = true; // Platform::InvalidCastException
```

Why am I getting an "invalid cast exception" when there is no casting going on at all?

The clue is in the C++/WinRT example, which throws `winrt::hresult_no_interface`.

Under the covers, Windows Runtime objects are COM objects, and their members are methods on COM interfaces. When you use a member, what happens behind the scenes is that the language projection queries the object for the COM interface that implements the desired member, and then it calls the corresponding interface method.

One of the rules of COM is that interfaces are immutable. Therefore, in order to add new members to the object, those new members need to be put on a new interface.

For example, the members of the `LauncherOptions` runtime class were introduced as follows:

Windows 8
`ILauncherOptions`
	`TreatAsUntrusted`
	`DisplayApplicationPicker`
	`UI`
	`PreferredApplicationPackageFamilyName`
	`PreferredApplicationDisplayName`
	`FallbackUri`
	`ContentType`

Windows 10 version 1507 `ILauncherOptions2`	`TargetApplicationPackageFamilyName` `NeighboringFilesQuery`
Windows 10 version 1607 `ILauncherOptions3`	`IgnoreAppUriHandlers`

Internally, a Windows Runtime object is represented by its "default interface". Deciding upon a default interface is usually a no-brainer, because a freshly-introduced object typically implements only one interface anyway.[1] For example, in Windows 8, the only interface supported by `LauncherOptions` is `ILauncherOptions`, which makes `ILauncher-Options` the default interface, seeing as you have no choice.

Using one of the Windows 8 properties goes like this:

```
// C#:        options.TreatAsUntrusted = true;
// C++/CX:    options->TreatAsUntrusted = true;
// C++/WinRT: options.TreatAsUntrusted(true);

// options is already a ILauncherOptions.
options->put_TreatAsUntrusted(true);
```

But using one of the properties added later takes a little more work:

```
// C#:        options.IgnoreAppUriHandlers = true;
// C++/CX:    options->IgnoreAppUriHandlers = true;
// C++/WinRT: options.IgnoreAppUriHandlers(true);

ILauncherOptions3* options3;
options->QueryInterface(IID_PPV_ARGS(&options3));
options3->put_IgnoreAppUriHandlers(true);
options3->Release();
```

If you take a program that uses `IgnoreAppUriHandlers` and run it on on a version of Windows that doesn't support the property, the `QueryInterface` call fails with `E_ NO-INTERFACE`. The language projection then converts this into a language-specific exception.

- C# converts it to a `System.InvalidCastException`.
- C++/CX converts it to a `Platform.InvalidCastException`.
- C++/WinRT converts it to a `winrt.hresult_no_interface`.

C# and C++/CX report this as an `InvalidCastException`, because the common case for this is where you try to cast an object to an interface that it doesn't support.

Instead of adding new interfaces, you might be tempted to add new members to the existing interface, in violation of COM rules. But that would result in profound sadness if a program tried to use one of those new members when running on a system that doesn't support it:

```
// C#:        options.IgnoreAppUriHandlers = true;
// C++/CX:    options->IgnoreAppUriHandlers = true;
// C++/WinRT: options.IgnoreAppUriHandlers(true);
//
// In hypothetical world where new members
// are added to ILauncherOptions.

options->put_IgnoreAppUriHandlers(true);
```

Since there is no `put_IgnoreAppUriHandlers` method in the vtable on older versions of Windows, this results not only in reading past the end of the vtable, but taking the undefined value past the end of the vtable and treating it as a function pointer! If you're lucky, this crashes unrecoverably. If you're unlucky, this is a security vulnerability.

Now that we understand the source of the invalid cast exception, we can look next time at what we can do about it.

[1] The default interface may not be `IUnknown` or `IInspectable`.

Raymond Chen

Follow