How can I check whether a Windows Runtime object supports a member before I try to use it?

devblogs.microsoft.com/oldnewthing/20200325-00

March 25, 2020

Raymond Chen

Last time, we saw that an invalid cast exception can mean that you tried to use a class member that isn't supported by the current operating system. What can you do to avoid this problem?

You might be running into this problem because your program is running on a version of Windows that you never intended to support. For example, your program is totally dependent upon some feature added in Windows 10 version 1903, and there's no point trying to run it on anything earlier. You can go to your manifest and set the `MinVersion` in your `TargetDeviceFamily` element to the minimum version that supports the thing you need. Your application will be deployed only to systems that satisfy your manifest's minimum requirements.

You can get the minimum version from MSDN under **Additional features and requirements**:

Device family	Windows 10, version 1803 (introduced v10.0.17134.0)

In this case, the magic number to put in your manifest is 10.0.17134.0. Naturally, if you have many minimum requirements, then the one you pick for your manifest is the highest.

```
<TargetDeviceFamily Name="Windows.Universal"
    MinVersion="10.0.17134.0"
    MaxVersionTested="10.0.18362.0" />
```

Another option is to perform a runtime check before trying to use the possibly-nonexistent member. The most direct way is to check specifically for the thing:

```
// C#
if (ApiInformation.IsMethodPresent("MyNamespace.MyClass", "SomeMember"))
{
  // There is a method called SomeMember
  myObject.SomeMember();
}
```

If overloads were added at different times, you will need to use the check that takes an arity.

```
if (ApiInformation.IsMethodPresent("MyNamespace.MyClass", "SomeMember", 1))
{
  // it is okay to call the overload of the SomeMember method with 1 parameter
  myObject.SomeMember(true);
}
```

You can use C#'s `nameof` operator and `Type.FullName` property to avoid hard-coding strings. In C++/WinRT, you can use `winrt::name_of<T>()` .

You can also check for the existence of properties, and even narrow your check to read-only properties or writable properties.

Another option for the runtime check is to check for the presence of the corresponding contract that introduced support for it. This information is also provided in MSDN under **Additional features and requirements**:

API contract	Windows.Foundation.UniversalApiContract (introduced v6)

The contract name and version are what you pass to the `IsApiContractPresent` method.

```
// C#
if (ApiInformation.IsApiContractPresent(
    "Windows.Foundation.UniversalApiContract", 6))
{
  // it is okay to use things that were introduced in
  // Windows.Foundation.UniversalApiContract version 6.
}
```

Bonus chatter: What about JavaScript? In JavaScript, an attempt to read a nonexistent member succeeds but returns `undefined` . This behavior is consistent with the overall design of the JavaScript language.

Raymond Chen

Follow