From a Windows app, how can I check whether there is an app installed that implements a particular URI scheme?

devblogs.microsoft.com/oldnewthing/20230308-04

March 8, 2023

Raymond Chen

A customer had a scenario where their main program wanted to launch their companion program via a URL deep link. If the companion isn't installed, then they want to open an alternate companion program with a different URL deep link. And if neither companion program is installed, they want to open the Store app to invite the user to download the primary companion program. They saw that the `LauncherOptions` class has a place to set a fallback URI to use if the primary URI could not be launched, but how do you set multiple fallbacks?

You can't.

The `FallbackUri` is a single fallback, and it must be an `http`, `https`, or `ms-windows-store` URI. The idea is that you try to launch the custom protocol URI, and if that doesn't work, you send the user to a web page equivalent. For example, the fallback for a social media app could be the same posting on the social media Web site. The fallback for a shopping app could be the same item on the shopping Web site. And if your app doesn't have an equivalent Web page for a deep link, you can use the `ms-windows-store` link that opens the Store app so the user can download the missing app.

If you want multiple levels of fallback, you'll have to code it up yourself. You can use the `QueryUriSupportAsync` method to check whether a protocols is supported, and you can even check whether the protocol is supported by a specific app, to avoid confusion if the protocol is registered by multiple apps.

```
// C#

// Prefer to launch in Contoso Deluxe
var uri1 = new Uri("contoso-deluxe:record=31415");
var app1 = "Contoso.Deluxe_3p14159265358";

// But will accept Contoso Classic
var uri2 = new Uri("contoso:record=31415");
var app2 = "Contoso_3p14159265358";

// The Store page for Contoso Deluxe
var store = new Uri("ms-windows-store://pdp/?ProductId=...");

var options = new LauncherOptions();

// Launch uri1 if Contoso Deluxe is installed and
// supports uri1.
var status = await Launcher.QueryUriSupportAsync(uri1,
    LaunchQuerySupportType.Uri, app1);
if (status == LaunchQuerySupportStatus.Available) {
    options.TargetApplicationPackageFamilyName = app1;
    return await Launcher.LaunchUriAsync(uri1, options);
}

// Launch uri2 if Contoso Classic is installed
// and supports uri2.
status = await Launcher.QueryUriSupportAsync(uri2,
    LaunchQuerySupportType.Uri, app2);
if (status == LaunchQuerySupportStatus.Available) {
    options.TargetApplicationPackageFamilyName = app2;
    return await Launcher.LaunchUriAsync(uri2, options);
}

// Launch the Store to install Contoso Deluxe.
return await Launcher.LaunchUriAsync(store);

// C++/WinRT

// Prefer to launch in Contoso Deluxe
auto uri1 = Uri(L"contoso-deluxe:record=31415");
auto app1 = L"Contoso.Deluxe_3p14159265358";

// But will accept Contoso Classic.
auto uri2 = Uri(L"contoso:record=31415");
auto app2 = L"Contoso_3p14159265358";

// The Store page for Contoso Deluxe.
auto store = Uri(L"ms-windows-store://pdp/?ProductId=...");

var options = LauncherOptions();

// Launch uri1 if Contoso Deluxe is installed
```

```
auto status = co_await Launcher::QueryUriSupportAsync(uri1,
    LaunchQuerySupportType::Uri, app1);
if (status == LaunchQuerySupportStatus::Available) {
    options.TargetApplicationPackageFamilyName(app1);
    co_return co_await Launcher::LaunchUriAsync(uri1, options);
}

// Launch uri2 if Contoso Classic is installed.
status = co_await Launcher::QueryUriSupportAsync(uri2,
    LaunchQuerySupportType.Uri, app2);
if (status == LaunchQuerySupportStatus::Available) {
    options.TargetApplicationPackageFamilyName(app2);
    co_return co_await Launcher::LaunchUriAsync(uri2, options);
}

// Launch the Store to install Contoso Deluxe.
co_return co_await Launcher::LaunchUriAsync(store);
```

The `QueryUriSupportAsync` trick is handy if you want to just check whether a URI could be launched, but without actually launching it. For example, you might want to show a "Open in Contoso Deluxe" button only if Contoso Deluxe is installed and supports the `contoso-deluxe:` scheme.

For our customer's specific case, we can do a little optimizing because we can combine the last two steps into one by using the built-in FallbackUri.

```
// C#

var options = new LauncherOptions();

// Launch uri1 if Contoso Deluxe is installed and
// supports uri1.
var status = await Launcher.QueryUriSupportAsync(uri1,
    LaunchQuerySupportType.Uri, app1);
if (status == LaunchQuerySupportStatus.Available) {
    options.TargetApplicationPackageFamilyName = app1;
    return await Launcher.LaunchUriAsync(uri1, options);
}

// Launch uri2 if Contoso Classic is installed
// and supports uri2; else launch the Store to
// install Contoso Deluxe.
options.TargetApplicationPackageFamilyName = app2;
options.FallbackUri = store;
return await Launcher.QueryUriSupportAsync(uri2,
    LaunchQuerySupportType.Uri, app2);

// C++/WinRT

var options = LauncherOptions();

// Launch uri1 if Contoso Deluxe is installed
auto status = co_await Launcher::QueryUriSupportAsync(uri1,
    LaunchQuerySupportType::Uri, app1);
if (status == LaunchQuerySupportStatus::Available) {
    options.TargetApplicationPackageFamilyName(app1);
    return co_await Launcher::LaunchUriAsync(uri1, options);
}

// Launch uri2 if Contoso Classic is installed
// and supports uri2; else launch the Store to
// install Contoso Deluxe.
options.TargetApplicationPackageFamilyName(app2);
options.FallbackUri(store);
co_return co_await Launcher::QueryUriSupportAsync(uri2,
    LaunchQuerySupportType::Uri, app2);
```

You could go even further and use the `PreferredApplication` information to provide the name and package family name of the program that Windows will recommend if the URI has no handler installed, allowing you to integrate with the existing “Open With” flow.

```
// C#

var options = new LauncherOptions();

// Launch uri1 if Contoso Deluxe is installed and
// supports uri1.
var status = await Launcher.QueryUriSupportAsync(uri1,
    LaunchQuerySupportType.Uri, app1);
if (status == LaunchQuerySupportStatus.Available) {
    options.TargetApplicationPackageFamilyName = app1;
    return await Launcher.LaunchUriAsync(uri1, options);
}

// Launch uri2 if Contoso Classic is installed
// and supports uri2; else launch the Store to
// install Contoso Deluxe.
options.TargetApplicationPackageFamilyName = app2;
options.PreferredApplicationDisplayName = "Contoso Deluxe";
options.PreferredApplicationPackageFamilyName = app1;
return await Launcher.QueryUriSupportAsync(uri2,
    LaunchQuerySupportType.Uri, app2);

// C++/WinRT

var options = LauncherOptions();

// Launch uri1 if Contoso Deluxe is installed
auto status = co_await Launcher::QueryUriSupportAsync(uri1,
    LaunchQuerySupportType::Uri, app1);
if (status == LaunchQuerySupportStatus::Available) {
    options.TargetApplicationPackageFamilyName(app1);
    return co_await Launcher::LaunchUriAsync(uri1, options);
}

// Launch uri2 if Contoso Classic is installed
// and supports uri2; else launch the Store to
// install Contoso Deluxe.
options.TargetApplicationPackageFamilyName(app2);
options.PreferredApplicationDisplayName(L"Contoso Deluxe");
options.PreferredApplicationPackageFamilyName(app1);
co_return co_await Launcher::QueryUriSupportAsync(uri2,
    LaunchQuerySupportType::Uri, app2);
```

Note that these tricks work only for detecting packaged apps which support a particular protocol. Next time, we'll look at looking for unpackaged apps which support a particular protocol.