How can I get information about media playing on the system, and optionally control their playback?

devblogs.microsoft.com/oldnewthing/20231108-00

November 8, 2023

Raymond Chen

Apps can publish their media playback information by using the `SystemMediaTransport-Controls`, but what about the other side of the coin? How can one app *retrieve* the media playback information from other apps?

The class that gets you access to what other apps are doing is the `GlobalSystemMedia-TransportControlsSessionManager` That class name is quite a mouthful, but it breaks down as the “global” “system media transport controls” “session manager”.

Just as a quick and dirty demonstration, here’s a simple console program that uses the `GlobalSystemMediaTransportControlsSessionManager` to find all the apps that have registered their media playback with a `SystemMediaTransportControls`, display some information about them, and (just for fun) pause any playback that is within one minute of the end.

```
#include <stdio.h> // Horrors! Mixing C and C++!
#include <winrt/Windows.ApplicationModel.h>
#include <winrt/Windows.Foundation.h>
#include <winrt/Windows.Foundation.Collections.h>
#include <winrt/Windows.Media.Control.h>

winrt::hstring DisplayNameFromAppId(winrt::hstring const& appid) try
{
    using namespace winrt::Windows::ApplicationModel;
    return AppInfo::GetFromAppUserModelId(appid).DisplayInfo().DisplayName();
}
catch (...)
{
    return appid;
}
```

The `DisplayNameFromAppId` function is a helper function that takes an app user model ID and returns the name for the app that is suitable for display. Now, it’s possible that the app playing media is not packaged, so we catch any failure and just return the original `appid`.

```
void PrintSeconds(PCWSTR label, winrt::Windows::Foundation::TimeSpan time)
{
    auto seconds = static_cast<uint32_t>(time / std::chrono::seconds(1));
    printf("\t%ls: %us\n", label, seconds);
}
```

The `PrintSeconds` helper function just prints a `TimeSpan` (rounded down to the nearest second).

Those are just helper functions for the main attraction:

```
winrt::Windows::Foundation::IAsyncAction Run()
{
    using namespace winrt::Windows::Media::Control;
    auto manager = co_await
        GlobalSystemMediaTransportControlsSessionManager::RequestAsync();
    auto current = manager.GetCurrentSession();
    if (current) {
        printf("Current media app: %ls\n",
            DisplayNameFromAppId(current.SourceAppUserModelId()).c_str());
    }
```

First, we acquire a `GlobalSystemMediaTransportControlsSessionManager` by requesting it from the system. We ask it for the current session, and if there is one, we print the name of the app that the system considers to be the one to which the media control hardware buttons will apply.

```
    auto sessions = manager.GetSessions();
    for (auto session : sessions)
    {
        printf("Session from: %ls\n",
            DisplayNameFromAppId(current.SourceAppUserModelId()).c_str());
```

Next, we ask the system for all the media sessions and iterate over them. For each session, we start by printing the name of the app the session belongs to.

```
        auto mediaProperties = co_await session.TryGetMediaPropertiesAsync();
        if (mediaProperties)
        {
            printf("\tTitle: %ls\n", mediaProperties.Title().c_str());
            printf("\tSubtitle: %ls\n", mediaProperties.Subtitle().c_str());
            printf("\tArtist: %ls\n", mediaProperties.Artist().c_str());
            printf("\tTrack number: %d\n", mediaProperties.TrackNumber());
        }
```

Next, we ask for the properties of the media being played. If we get something, we print the title, subtitle, artist, and track number. There are other properties available, but we'll content ourselves with this information.

This information comes from the media player app, so it’s up to the media player app to provide as much or as little information as it wishes.

```
        auto timelineProperties = session.GetTimelineProperties();
        PrintSeconds(L"Position", timelineProperties.Position());
        PrintSeconds(L"Start", timelineProperties.StartTime());
        PrintSeconds(L"End", timelineProperties.EndTime());
```

Next, we ask for information about the playback timeline and print the current playback position, as well as the start and end timecode of the playback. Again, there’s more information available, but our sample prints only these three pieces of information.

```
        auto info = session.GetPlaybackInfo();
        auto rate = info.PlaybackRate();
        if (rate != nullptr)
        {
            printf("\tPlayback speed: %gx\n", rate.Value());
        }
```

Next, we get the playback info and print the playback rate if known.

```
        auto controls = info.Controls();
        printf("\tCan pause: %ls\n", controls.IsPauseEnabled() ? L"true" : L"false");
```

We also get the playback controls and report whether the Pause button is enabled.

```
        if (info.PlaybackStatus() ==
            GlobalSystemMediaTransportControlsSessionPlaybackStatus::Playing &&
            controls.IsPauseEnabled() &&
            timelineProperties.EndTime() - timelineProperties.Position() <
                std::chrono::minutes(1))
        {
            co_await session.TryPauseAsync();
        }
    }
}
```

Finally, we do some arbitrary work: if the media is currently playing, the Pause button is enabled, and the playback is within one minute of the end, then we programmatically press the Pause button.

Finally, we wrap the whole thing inside a simple `main()`:

```
int __cdecl main() try
{
    winrt::init_apartment(winrt::apartment_type::multi_threaded);
    Run().get();
    return 0;
}
catch (...)
{
    auto hr = winrt::to_hresult();
    printf("Error 0x%08x, oops.\n", hr);
    return 0;
}
```

The intended audience of the `GlobalSystemMediaTransportControlsSessionManager` family of objects is programs that allow you to control media playback from your smart watch or other device. The idea is that the helper program can query media information from the system and send the relevant details to the smart watch, and then when you tap the Pause button on the smart watch (say), the helper program can turn around and pause the current media session.

For a full version of this program, you would want to subscribe to the various events like `CurrentSessionChanged` and `MediaPropertiesChanged` so you can track and respond to changes in the system, but this quick sample doesn't bother because it just reports on a point in time.

Here's a translation to C#:

```
using System;
using System.Threading.Tasks;
using Windows.ApplicationModel;
using Windows.Media.Control;

class Program
{
    static string DisplayNameFromAppId(string appid)
    {
        try
        {
            return AppInfo.GetFromAppUserModelId(appid).DisplayInfo.DisplayName;
        }
        catch (Exception)
        {
            return appid;
        }
    }


    static async Task Run()
    {
        var manager =
            await GlobalSystemMediaTransportControlsSessionManager.RequestAsync();
        var current = manager.GetCurrentSession();
        if (current != null)
        {
            Console.WriteLine("Current media app: " +
                              DisplayNameFromAppId(current.SourceAppUserModelId));
        }
        foreach (var session in manager.GetSessions())
        {
            Console.WriteLine("Session from: "
                              DisplayNameFromAppId(session.SourceAppUserModelId));

            var timelineProperties = session.GetTimelineProperties();
            Console.WriteLine($"\tPosition: {timelineProperties.Position}");
            Console.WriteLine($"\tStart: {timelineProperties.StartTime}");
            Console.WriteLine($"\tEnd: {timelineProperties.EndTime}");

            var info = session.GetPlaybackInfo();
            var rate = info.PlaybackRate;
            if (rate != null)
            {
                Console.WriteLine($"\tPlayback speed: {rate.Value}");
            }

            var controls = info.Controls;
            Console.WriteLine($"\tCan pause: {controls.IsPauseEnabled}");

            if (info.PlaybackStatus ==
                GlobalSystemMediaTransportControlsSessionPlaybackStatus.Playing &&
                controls.IsPauseEnabled &&
                timelineProperties.EndTime - timelineProperties.Position <
                                                        TimeSpan.FromMinutes(1))
            {
                await session.TryPauseAsync();
            }
        }
    }

    [MTAThread]
    public static void Main()
    {
        Run().Wait();
    }
}
```