The SettingsIdentifier property of the various file pickers lets you give names to your pickers

devblogs.microsoft.com/oldnewthing/20200525-00

May 25, 2020

Raymond Chen

A few of the Windows Runtime pickers have a property named `SettingsIdentifier`. What is that thing?

Basically, it's a way for you to give a name to your pickers.

First, let's talk about how things work normally.

When you use a `FileOpenPicker`, the picker normally defaults to the same location that the user picked the last time your program displayed a `FileOpenPicker`. Similarly for `FileSavePicker` and `FolderPicker`. Each kind of picker gets its own last-remembered location.

But suppose your program is a movie editor. Users can add movie clips to the project, or they could add background music. So let's get started.

The user clicks *Add movie clip*, and find a movie clip. Then they click *Add background music*, and the default location is the folder from which they picked the movie clip. They navigate out of that folder to the folder where they keep their music, and then pick some music.

Now they want to do it again. Once again, they click *Add movie clip*, and it defaults to the folder from which they picked the background music. Again, they have to navigate out of that folder to the folder where they keep their videos. Then they click *Add background music*, and again it defaults to the folder of videos, and they have to navigate out of that folder to where they keep their music.

Each time they want to pick something, it always starts in the wrong place.

To solve this problem, you can assign custom names to the *Add movie clip* picker and the *Add background music* picker:

```
// C#
async Task<StorageFile> PickMovieClipAsync()
{
  var picker = new FileOpenPicker {
    SuggestedStartLocation = PickerLocationId.VideosLibrary,
    FileTypeFilter = { ".mp4", ".wmv" },
    SettingsIdentifier = "MovieClip"
  };
  return await picker.PickSingleFileAsync();
}

async Task<StorageFile> PickBackgroundMusicAsync()
{
  var picker = new FileOpenPicker {
    SuggestedStartLocation = PickerLocationId.MusicLibrary,
    FileTypeFilter = { ".mp3", ".m4a", ".wav", ".wma" },
    SettingsIdentifier = "BackgroundMusic"
  };
  return await picker.PickSingleFileAsync();
}

// C++/WinRT
IAsyncOperation<StorageFile> PickMovieClipAsync()
{
  auto picker = FileOpenPicker();
  picker.SuggestedStartLocation(PickerLocationId::VideosLibrary);
  picker.FileTypeFilter().ReplaceAll({ L".mp4", L".wmv" });
  picker.SettingsIdentifier(L"MovieClip");
  return co_await picker.PickSingleFileAsync();
}

IAsyncOperation<StorageFile> PickBackgroundMusicAsync()
{
  auto picker = FileOpenPicker();
  picker.SuggestedStartLocation(PickerLocationId::MusicLibrary);
  picker.FileTypeFilter().ReplaceAll({ ".mp3", ".m4a", ".wav", ".wma" });
  picker.SettingsIdentifier(L"BackgroundMusic");
  return co_await picker.PickSingleFileAsync();
}

// C++/CX
task<StorageFile> PickMovieClipAsync()
{
  auto picker = ref new FileOpenPicker();
  picker->SuggestedStartLocation = PickerLocationId::VideosLibrary;
  picker->FileTypeFilter->Append(L".mp4");
  picker->FileTypeFilter->Append(L".wmv");
  picker->SettingsIdentifier = L"MovieClip";
  return co_await picker->PickSingleFileAsync();
}

task<StorageFile> PickBackgroundMusicAsync()
```

```
{
  auto picker = ref new FileOpenPicker();
  picker->SuggestedStartLocation = PickerLocationId::MusicLibrary;
  picker->FileTypeFilter->Append(L".mp3");
  picker->FileTypeFilter->Append(L".m4a");
  picker->FileTypeFilter->Append(L".wav");
  picker->FileTypeFilter->Append(L".wma");
  picker->SettingsIdentifier = L"BackgroundMusic";
  return co_await picker->PickSingleFileAsync();
}

// JavaScript
async function pickMovieClipAsync()
{
  var picker = new Windows.Storage.Pickers.FileOpenPicker();
  picker.suggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.videosLibrary;
  picker.fileTypeFilter.replaceAll([ ".mp4", ".wmv" ]);
  picker.settingsIdentifier = "MovieClip";
  return await picker.pickSingleFileAsync();
}

async function pickBackgroundMusicAsync()
{
  var picker = new Windows.Storage.Pickers.FileOpenPicker();
  picker.suggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.musicLibrary;
  picker.fileTypeFilter.replaceAll([ ".mp3", ".m4a", ".wav", ".wma" ]);
  picker.settingsIdentifier = "BackgroundMusic";
  return await picker.pickSingleFileAsync();
}
```

Assigning a name to each picker keeps their identities distinct. Now, when the user clicks *Add movie clip*, they default to the folder that they most recently used *to pick movie clips*. And similarly for background music.

You can use the same trick for Save pickers and Folder pickers. For example, you might have two ways of saving the output of the program. One is to save the project as a project that can be edited further, and another is to save the project as a finished movie file. You can give different names to those pickers so that saving a project defaults to the folder where the user keeps all their projects, and saving a finished movie defaults to the folder where the user keeps all their finished movies.

Bonus chatter: The provider classes also have a `SettingsIdentifier` property. This is the implementation side of the same workflow: If you are a file picker provider, you should cache the picker settings under that identifier. When you are asked to provide the UI for a picker, look up that identifier in your cache, and if you find a match, set the picker defaults to the values you had previously saved.

On the provider side, the system will do the work of assigning a default settings identifier if the calling application didn't provide one, and the system will also do the work of making sure that the settings identifiers from different applications do not collide. You can just treat it as an opaque string to be used as a key in your settings cache.

Raymond Chen

Follow