

# Eseményvezérelt alkalmazások: 12. gyakorlat

A gyakorlat célkitűzése az előző gyakorlaton elkészített aszinkron kép letöltő alkalmazás (`ImageDownloader`) kiegészítése képmentés funkcióval, melyhez platformspecifikus kódot fogunk írni.

## 1 Nézet és nézetmodell kiegészítése

Induljunk ki a *11. gyakorlat* megoldásából, majd a *9. gyakorlat*-hoz nagyon hasonlóan, adjuk hozzá a kép letöltését kezdeményező gombot az `ImagePage.xaml`-höz, illetve kössük össze egy nézetmodellbeli `SaveImageCommand` paranccsal, mely egyetlen feladata kiváltani egy `SaveImage` eseményt, melynek paraméterül átadja az aktuálisan kiválasztott képet.

Az `AppShell.xaml.cs` fájlban iratkozzunk fel erre az eseményre.

## 2 Platformspecifikus perzisztencia

A képek letöltésekor fontos lenne, hogy azokat más alkalmazásokkal is (könnyen) elérhessük, ezért nem mentjük őket egyszerűen az adott platform alkalmazás adatkönyvtárába (`FileSystem.AppDataDirectory`). Mivel a platformfüggetlen fájl mentés dialógus ablak még fejlesztés alatt áll a MAUI keretrendszerhez, ezért más megoldást választunk: Windows operációs rendszeren a *Képek* könyvtárba, Android operációs rendszeren a *Letöltések* könyvtárba mentjük a képeket.

*Megjegyzés:* A file megnyitására szolgáló dialógus elérhető platformfüggetlen megvalósításban a MAUI keretrendszer jelenlegi változatában, a `FilePicker` osztály használatával. A `CommunityToolkit`-ben már folyamatban van a fájl mentés és a mappa kiválasztás dialógusok platformfüggetlenül hívható implementációja.

### 2.1 Az `ImageUtil` osztály

Hozzunk létre egy `ImageUtil` nevű statikus osztályt a projektben. Az osztály legyen parciális, ezáltal az esetleges közös kód kerülhetne a platformfüggetlen, megosztott változatba. Az osztály egyetlen statikus `SaveImage` metódust tartalmazzon melynek feladata a paraméterül kapott képet a megfelelő helyre menteni.

```
public static partial void SaveImage(ImageSource image, string path);
```

Hozzunk létre platform-specifikus implementációkat Windows és Android operációs rendszerekre. Ehhez egyszerűen a `Platforms` könyvtár megfelelő könyvtárában helyezünk el C# forrás fájlokat, és ezek az állományok csakis a [megfelelő platformra kerülnek fordításra](#).

#### 2.1.1 Windows

Majd hozzuk létre a Windows specifikus megvalósítást a `Platforms/Windows/ImageUtil.cs` fájlban. Hozzunk létre egy `FileStream`-et ahova szeretnénk kiírni a fájlt. A kapott `ImageSource` adatfolyamából (`stream`) pedig először dekódoljuk a kiírásra szánt képet a saját formátumából, majd kódoljuk `PNG`-ként és írjuk ki a kimeneti folyamra.

```
public static async partial void SaveImage(ImageSource image, string path)
{
    using (var outputStream = new FileStream(path, FileMode.Create).AsRandomAccessStream())
```

```

{
    var inStream = await (image as StreamImageSource)!.Stream(CancellationTokens.None);
    var decoder = await BitmapDecoder.CreateAsync(inStream.AsRandomAccessStream());
    var encoder = await BitmapEncoder.CreateAsync(BitmapEncoder.PngEncoderId, outStream);
    encoder.SetSoftwareBitmap(await decoder.GetSoftwareBitmapAsync());
    await encoder.FlushAsync();
}
}

```

*Megjegyzés:* Windowson elérhető a `FileSavePicker` osztály is, de ez némiképp megbonyolítaná az implementációt.

### 2.1.2 Android

Hozzuk létre az Android specifikus változatot is a `Platforms/Android/ImageUtil.cs` fájlban. Itt először győződjünk meg róla, hogy az alkalmazásunknak joga van írni a tárolóra, majd az előző pontban szereplőhöz nagyon hasonló folyamattal az `Android.Graphics.Bitmap.CompressAsync` segítségével írjuk ki PNG formátumban a képet a megadott útvonalra.

```

public static async partial void SaveImage(ImageSource image, string path)
{
    var status = await Permissions.RequestAsync<Permissions.StorageWrite>();
    if (status is not PermissionStatus.Granted)
    {
        throw new PermissionException("Storage permission is not granted");
    }

    using (var outputStream = new FileStream(path, FileMode.Create))
    {
        var handler = new StreamImageSourceHandler();
        Bitmap pic = await handler.LoadImageAsync(image, Android.App.Application.Context);
        await pic.CompressAsync(Bitmap.CompressFormat.Png, 100, outputStream);
    }
}

```

Ahhoz, hogy jelezzük, hogy alkalmazásunk bizonyos funkciói extra jogosultságokat igényelnek, írjunk hozzá a `Platforms/Android/AndroidManifest.xml`-hez az alábbiakat:

```

<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

```

## 2.2 Dialógus

Az `AppShell` osztályban a `SaveImage` eseménykezelőjében fogjuk majd meghívni az imént létrehozott metódust, de előtte még kérdezzük meg a felhasználót, hogy milyen néven szeretné elmenteni a képet. Ehhez használjuk a `DisplayPromptAsync` metódust.

Androidon alapesetben az alkalmazásunk privát területére tudnánk írni, de most a munkafüzet elején tárgyaltnak megfelelően jobb lenne egy publikus helyre menteni. Mivel ez is platformspecifikus, használjunk feltételes fordítást. Androidon mentsünk a *Letöltések* mappába, Windowson pedig a felhasználó *Képek* mappájába.

```

#if ANDROID
    string path = Android.OS.Environment.GetExternalStoragePublicDirectory(
        Android.OS.Environment.DirectoryDownloads).AbsolutePath;
#else
    string path = Environment.GetFolderPath(Environment.SpecialFolder.MyPictures);
#endif
    string filename = await DisplayPromptAsync("Kép letöltése és mentése",
        $"Mentés helye:\n{path}");

```

Ezekután már minden információ birtokában vagyunk, hogy meghívjuk az platformspecifikus képmentő metódust.

### 3 Fájlnev alapú feltételes fordítás (*megjegyzés*)

A forrásfájlok platformokhoz rendelve a könyvtárrendszer szerveződése helyett a fájlok elnevezési konvenciójával megadható. Ennek tipikus példája, hogy ha egy fájl a platformnak megfelelő suffix-al látunk el, akkor csak az adott platformon legyen a fordítási folyamat része. Ehhez adjuk hozzá a projektfájlhoz az alábbiakat:

```
<!-- Android -->
<ItemGroup Condition="$(TargetFramework.StartsWith('net6.0-android')) != true">
  <Compile Remove="**\**\*.Android.cs" />
  <None
    Include="**\**\*.Android.cs"
    Exclude="$(DefaultItemExcludes);$(DefaultExcludesInProjectFolder)" />
</ItemGroup>

<!-- Both iOS and Mac Catalyst -->
<ItemGroup
  Condition="$(TargetFramework.StartsWith('net6.0-ios')) != true AND
    $(TargetFramework.StartsWith('net6.0-maccatalyst')) != true">
  <Compile Remove="**\**\*.MaciOS.cs" />
  <None
    Include="**\**\*.MaciOS.cs"
    Exclude="$(DefaultItemExcludes);$(DefaultExcludesInProjectFolder)" />
</ItemGroup>

<!-- iOS -->
<ItemGroup Condition="$(TargetFramework.StartsWith('net6.0-ios')) != true">
  <Compile Remove="**\**\*.iOS.cs" />
  <None
    Include="**\**\*.iOS.cs"
    Exclude="$(DefaultItemExcludes);$(DefaultExcludesInProjectFolder)" />
</ItemGroup>

<!-- Mac Catalyst -->
<ItemGroup Condition="$(TargetFramework.StartsWith('net6.0-maccatalyst')) != true">
  <Compile Remove="**\**\*.MacCatalyst.cs" />
  <None
    Include="**\**\*.MacCatalyst.cs"
    Exclude="$(DefaultItemExcludes);$(DefaultExcludesInProjectFolder)" />
</ItemGroup>

<!-- Windows -->
<ItemGroup Condition="$(TargetFramework.Contains('-windows')) != true">
  <Compile Remove="**\*.Windows.cs" />
  <None
    Include="**\*.Windows.cs"
    Exclude="$(DefaultItemExcludes);$(DefaultExcludesInProjectFolder)" />
</ItemGroup>
```