Capture photos and video with the Windows built-in camera UI

02/08/2017 • 4 minutes to read • +2

In this article

This article describes how to use the **CameraCaptureUI** class to capture photos or videos by using the camera UI built into Windows. This feature is easy to use. It allows your app to get a user-captured photo or video with just a few lines of code.

If you want to provide your own camera UI, or if your scenario requires more robust, low-level control of the capture operation, then you should use the **MediaCapture** class, and implement your own capture experience. For more information, see Basic photo, video, and audio capture with MediaCapture.

ⓘ **Note**

You shouldn't specify the **webcam** nor **microphone** capabilities in your app manifest file if your app only uses **CameraCaptureUI**. If you do, your app will be displayed in the device's camera privacy settings, but even if the user denies camera access to your app, this won't prevent the **CameraCaptureUI** from capturing media.

This is because the Windows built-in camera app is a trusted first-party app that requires the user to initiate photo, audio, and video capture with a button press. Your app may fail Windows Application Certification Kit certification when submitted to Microsoft Store if you specify the webcam or microphone capabilities when using **CameraCaptureUI** as your only photo capture mechanism.

You must specify the **webcam** or **microphone** capabilities in your app manifest file if you're using **MediaCapture** to capture audio, photos, or video programmatically.

Capture a photo with CameraCaptureUI

To use the camera capture UI, include the **Windows.Media.Capture** namespace in your project. To do file operations with the returned image file, include **Windows.Storage**.

C# Copy

```
using Windows.Media.Capture;
using Windows.Storage;
```

To capture a photo, create a new CameraCaptureUI object. By using the object's PhotoSettings property, you can specify properties for the returned photo, such as the image format of the photo. By default, the camera capture UI supports cropping the photo before it's returned. This can be disabled with the AllowCropping property. This example sets the CroppedSizeInPixels to request that the returned image be 200 x 200 in pixels.

Note

Image cropping in the **CameraCaptureUI** isn't supported for devices in the Mobile device family. The value of the AllowCropping property is ignored when your app is running on these devices.

Call CaptureFileAsync and specify CameraCaptureUIMode.Photo to specify that a photo should be captured. The method returns a StorageFile instance containing the image if the capture is successful. If the user cancels the capture, the returned object is null.

C# Copy

```
CameraCaptureUI captureUI = new CameraCaptureUI();
captureUI.PhotoSettings.Format = CameraCaptureUIPhotoFormat.Jpeg;
captureUI.PhotoSettings.CroppedSizeInPixels = new Size(200, 200);

StorageFile photo = await
captureUI.CaptureFileAsync(CameraCaptureUIMode.Photo);

if (photo == null)
{
    // User cancelled photo capture
    return;
}
```

The **StorageFile** containing the captured photo is given a dynamically generated name and saved in your app's local folder. To better organize your captured photos, you can move the file to a different folder.

C# Copy

```
StorageFolder destinationFolder =
    await
ApplicationData.Current.LocalFolder.CreateFolderAsync("ProfilePhotoFolder
",
        CreationCollisionOption.OpenIfExists);
```

```
await photo.CopyAsync(destinationFolder, "ProfilePhoto.jpg",
NameCollisionOption.ReplaceExisting);
await photo.DeleteAsync();
```

To use your photo in your app, you may want to create a SoftwareBitmap object that can be used with several different Universal Windows app features.

First, include the Windows.Graphics.Imaging namespace in your project.

C# Copy

```
using Windows.Storage.Streams;
using Windows.Graphics.Imaging;
```

Call OpenAsync to get a stream from the image file. Call BitmapDecoder.CreateAsync to get a bitmap decoder for the stream. Then, call GetSoftwareBitmap to get a **SoftwareBitmap** representation of the image.

C# Copy

```
IRandomAccessStream stream = await photo.OpenAsync(FileAccessMode.Read);
BitmapDecoder decoder = await BitmapDecoder.CreateAsync(stream);
SoftwareBitmap softwareBitmap = await decoder.GetSoftwareBitmapAsync();
```

To display the image in your UI, declare an Image control in your XAML page.

XML Copy

```
<Image x:Name="imageControl" Width="200" Height="200"/>
```

XML Copy

```
<Image x:Name="imageControl" Width="200" Height="200"/>
```

To use the software bitmap in your XAML page, include the using Windows.UI.Xaml.Media.Imaging namespace in your project.

C# Copy

```
using Windows.UI.Xaml.Media.Imaging;
```

The **Image** control requires that the image source be in BGRA8 format with premultiplied alpha or no alpha. Call the static method SoftwareBitmap.Convert to create a new software bitmap with the desired format. Next, create a new SoftwareBitmapSource object and call it SetBitmapAsync to assign the software bitmap to the source. Finally, set the **Image** control's Source property to display the captured photo in the UI.

C# Copy

```
SoftwareBitmap softwareBitmapBGR8 =
SoftwareBitmap.Convert(softwareBitmap,
        BitmapPixelFormat.Bgra8,
        BitmapAlphaMode.Premultiplied);

SoftwareBitmapSource bitmapSource = new SoftwareBitmapSource();
await bitmapSource.SetBitmapAsync(softwareBitmapBGR8);

imageControl.Source = bitmapSource;
```

Capture a video with CameraCaptureUI

To capture a video, create a new CameraCaptureUI object. By using the object's VideoSettings property, you can specify properties for the returned video, such as the format of the video.

Call CaptureFileAsync and specify Video to capture a video. The method returns a StorageFile instance containing the video if the capture is successful. If you cancel the capture, the returned object is null.

C# Copy

```
CameraCaptureUI captureUI = new CameraCaptureUI();
captureUI.VideoSettings.Format = CameraCaptureUIVideoFormat.Mp4;

StorageFile videoFile = await
captureUI.CaptureFileAsync(CameraCaptureUIMode.Video);

if (videoFile == null)
{
    // User cancelled photo capture
    return;
}
```

What you do with the captured video file depends on the scenario for your app. The rest of this article shows you how to quickly create a media composition from one or more captured videos and show it in your UI.

First, add a MediaPlayerElement control in which the video composition will display on your XAML page.

XML Copy

```
<MediaPlayerElement x:Name="mediaPlayerElement" Width="320" Height="240"
AreTransportControlsEnabled="True"/>
```

When the video file returns from the camera capture UI, create a new **MediaSource** by calling **CreateFromStorageFile**. Call the **Play** method of the default **MediaPlayer** associated with the **MediaPlayerElement** to play the video.

C# Copy

```csharp
mediaPlayerElement.Source = MediaSource.CreateFromStorageFile(videoFile);
mediaPlayerElement.MediaPlayer.Play();
```

Related topics

- Camera
- Basic photo, video, and audio capture with MediaCapture
- CameraCaptureUI

Is this page helpful?

Yes No