



# Windows on Arm

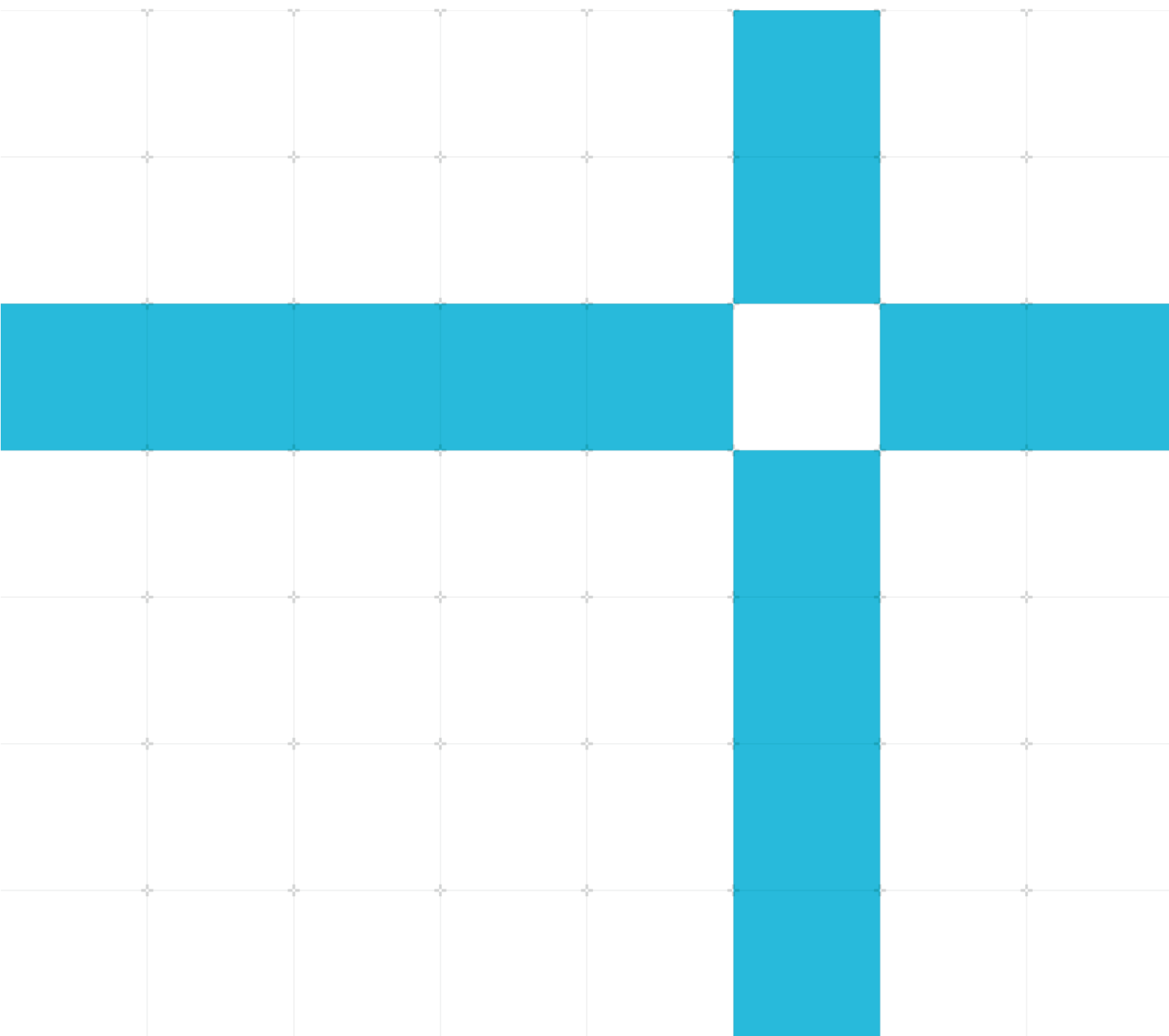
## Building a Native Windows on Arm App with WinUI 3

Non-Confidential

Copyright © 2022 Arm Limited (or its affiliates).  
All rights reserved.

**Issue 1.0**

102767



## Windows on Arm

### Building a Native Windows on Arm App with WinUI 3

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.

#### Release information

#### Document history

Issue	Date	Confidentiality	Change
01	06 January 2022	Non-Confidential	

### Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2022 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

## Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

## Product Status

The information in this document is Final, that is for a developed product.

## Web Address

[www.arm.com](http://www.arm.com)

# Contents

<b>1 Overview .....</b>	<b>5</b>
1.1 Before you begin.....	5
1.2 Prepare the development PC .....	5
1.3 Prepare the target device .....	6
<b>2 Create a WinUI 3.0 application.....</b>	<b>7</b>
2.1 Create the WinUI 3.0 desktop project .....	8
2.2 Add controls to the main window .....	10
<b>3 Deploy the application on the two in one laptop.....</b>	<b>14</b>
3.1 Publishing the binaries for each mode .....	14
3.2 Create the app packages.....	16
3.3 Install the package on the two in one laptop.....	19
<b>4 Benchmark the app.....</b>	<b>20</b>
4.1 Benchmark X86 emulation.....	20
4.2 Benchmark AArch64 native.....	20
<b>5 Summary.....</b>	<b>21</b>
<b>6 Related Information .....</b>	<b>22</b>
<b>7 Next steps.....</b>	<b>23</b>

# 1 Overview

In this guide, you can learn how to create a simple but complete WoA-native WinUI 3 application.

Microsoft recently released WinUI 3.0, which runs on .NET 5.0, on machines with AArch64 processors (Arm 64), Intel, and AMD. WinUI 3.0 also runs on .NET 5.0.8, Windows forms, UWP, and WPF **already have native Arm support**. As seen in the **Best Practices for Migrating Windows apps to Windows on Arm**, applications on an AArch64 device can run in x86 emulation mode. Running in emulation mode is fine for applications that do not require high performance. Sometimes, it can be more than ten times slower than running in native mode. Therefore, to really take advantage of the power of the new platform, you want to create native apps.

This guide describes how to develop a simple graphical application for a device with an AArch64 processor. The device we are using to test the app is a Microsoft surface pro X device with an AArch64 processor. I call this device the two in one laptop.

## 1.1 Before you begin

To work through in this guide, you need visual studio 2019 (VS2019) as an Integrated Development Environment (IDE), with C#. Developing WinUI 3.0 applications involves minimal knowledge of XAML and WPF.

You can find the code for this article on [GitHub](#).

## 1.2 Prepare the development PC

Detailed instructions to install the right components can be found **Windows app development site**. If you have not installed VS2019, you can use the download link the page provides. While you can complete the exercise on VS2022, it is still in preview, so proceed with caution.

If you already have VS2019, open the visual studio installer to add any necessary workloads.

## 1.3 Prepare the target device

To prepare your AArch64 PC, follow the instructions to [enable your device for development](#). Then:

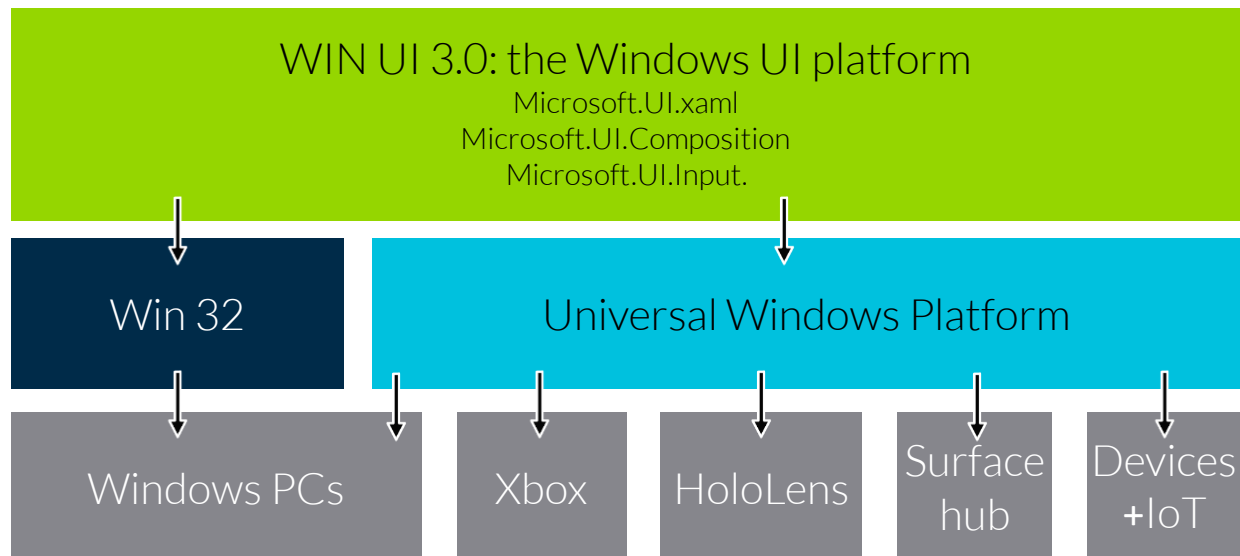
- Download [.NET 5.0 \(Linux, macOS, and Windows\)](#)
- Install [.NET 5.0 x86](#)
- Install [.NET 5.0 Arm64](#).

Install the x86 version to contrast the x86 emulated mode with the AArch64 native mode.

## 2 Create a WinUI 3.0 application

The **Windows UI library** (WinUI) contains the documentation for WinUI 3.0, and the following diagram. The diagram highlights the purpose of WinUI 3, which enables you to use one framework to create applications for all possible Windows platforms.

The following diagram shows the support for the AArch64 processor:



**Figure 1: Creating applications with WinUi3 for Windows platforms**

First, we create a small application that allows you to reduce the resolution of a user-loaded image. The resolution is downgraded by selecting squares of 20 x 20 pixels and replacing the color of each selected square with the average color of its pixels.

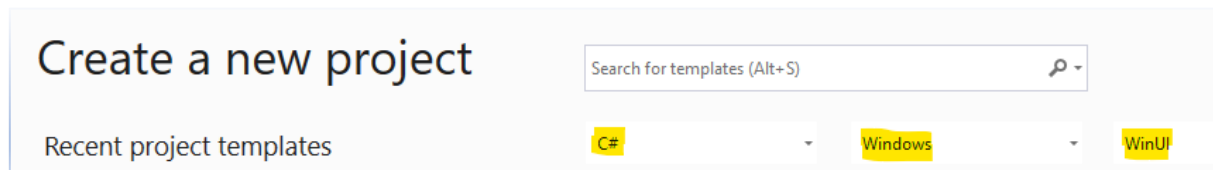
This example is just a demo application, so there are not many options. But it is enough to perform some benchmarks. Detailed instructions for creating Windows app SDK projects can be found on the [Windows app development site](#).

In the next chapter, we create the downgrade picture application.

## 2.1 Create the WinUI 3.0 desktop project

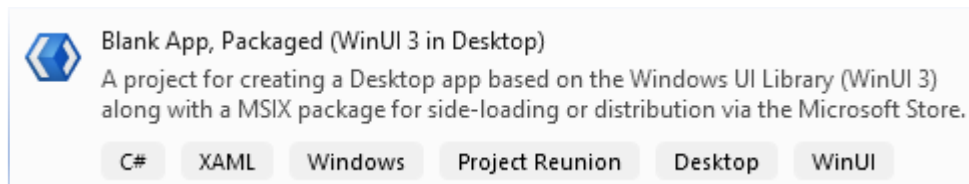
Open VS2019 and follow the instructions:

1. Click **File > New > Project**, to create a new project and then select **C#**, **Windows**, and **WinUI**:



**Figure: Creating new project**

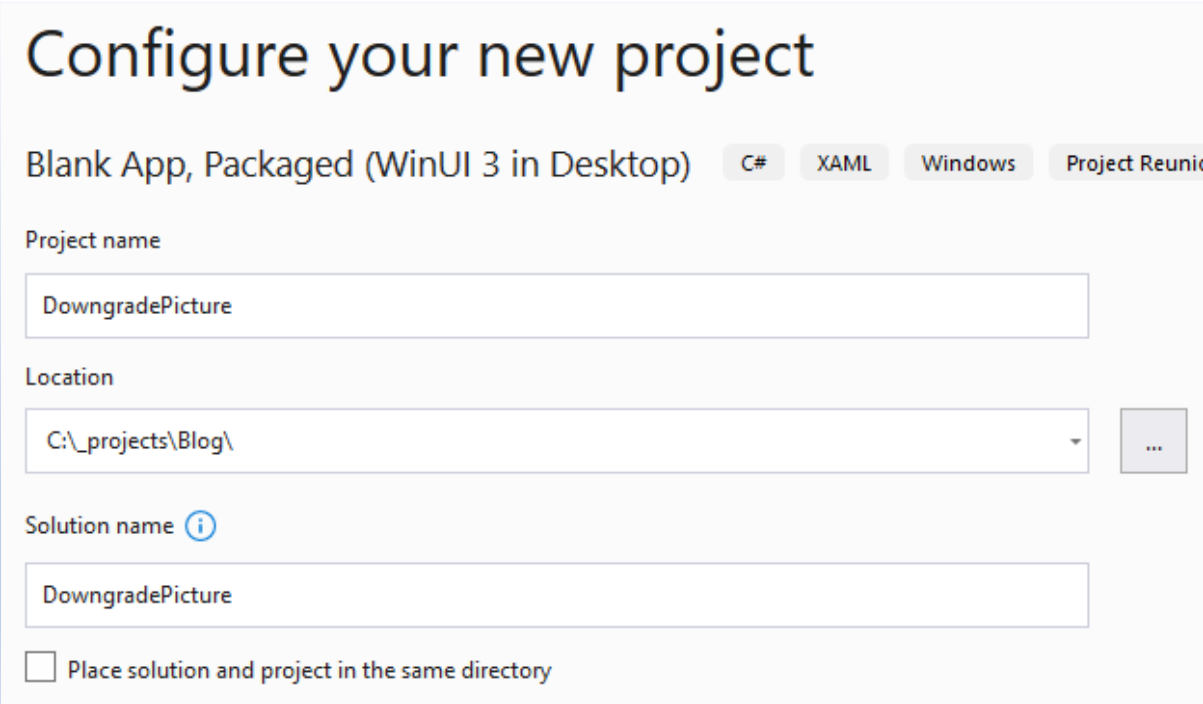
2. Select **Blank App, Packaged (WinUI 3 in Desktop)**:



**Figure: Selecting blank app, packaged**

3. Name the app to **DowngradePicture**, then choose a location, and click create:





**Configure your new project**

Blank App, Packaged (WinUI 3 in Desktop) C# XAML Windows Project Reunion

Project name  
DowngradePicture

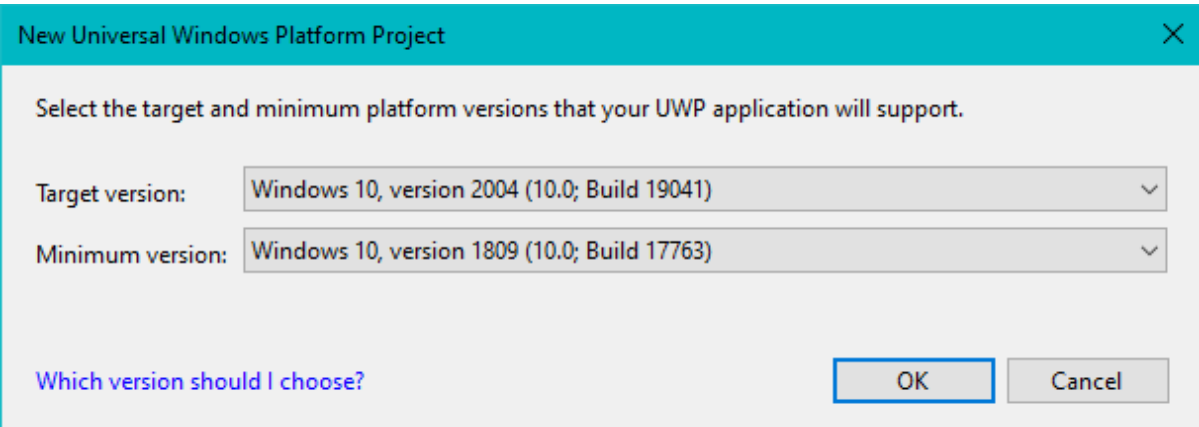
Location  
C:\\_projects\Blog\

Solution name ⓘ  
DowngradePicture

☐ Place solution and project in the same directory

Figure: Naming the app, selecting the location

4. Select the target and minimum versions of Windows on the appearing dialog box. Leave the default values and click **OK**:



**New Universal Windows Platform Project**

Select the target and minimum platform versions that your UWP application will support.

Target version: Windows 10, version 2004 (10.0; Build 19041)

Minimum version: Windows 10, version 1809 (10.0; Build 17763)

Which version should I choose?

OK Cancel

Figure: Platform version dialog box



The settings seen on the previous figure, also work in Windows 11.

After a few seconds, the solution explorer and an overview page will appear. We now have an application that can run on our machine. It does not do much yet, but it is the beginning of your first WinUI 3.0 application.

In the Solution Explorer, there are two projects:

- **DowngradePicture** contains the code and resources for the project. Here we publish the application for different versions (x86 and AArch64).
- **DowngradePicture (Package)** contains the manifest file of the application. This file contains all the information that our application must run. Notice that this is a startup project. The manifest file allows for setting the icons and capabilities of the application.

If you are familiar with Windows Presentation Foundation, you can recognize the MainWindow.xaml file. This file is where we add the code to load and display the image file.

## 2.2 Add controls to the main window

If you prefer not to type from blank, you can find the necessary code on [GitHub](#).

On the main window, we want to show the original and downgraded pictures with some controls. Also, we would like a text block to display the time taken to downgrade the image.

The following image shows how the reduced image must look like compared to the original:

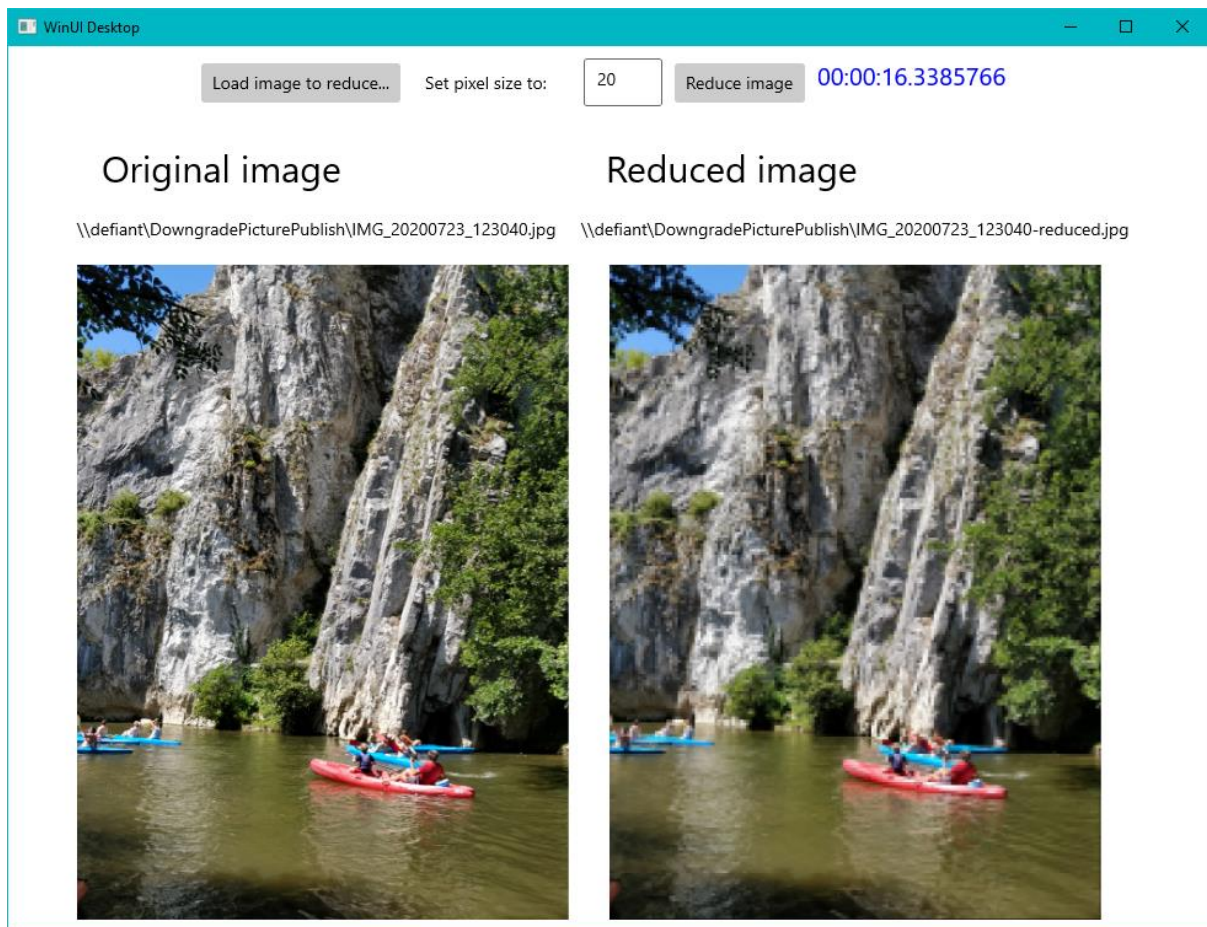


Figure: Original and downgraded image comparison

The following XAML code to accomplish the task, can be found on [GitHub](#):

```
<Window
    x:Class="Downgrade Aarch64 Picture.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:DowngradePicture"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <StackPanel Orientation="Vertical" HorizontalAlignment="Center" >
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center" >
            <Button Click="OnLoadClicked">Load image to reduce...</Button>
            <TextBlock Margin="20" Text="Set pixel size to: " />
        </StackPanel>
    </StackPanel>
```

```

        <TextBox Margin="10" Text="20" x:Name="pixelSize" TextChanged="PixelSize_TextChanged" />

        <Button x:Name="reduceButton" Click="OnReduceClicked" IsEnabled="False">Reduce image</Button>

        <TextBlock Margin="10" FontSize="20" x:Name="txtDuration" Foreground="Blue" />
    />

    </StackPanel>

    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center" >
        <StackPanel Orientation="Vertical" HorizontalAlignment="Center" >
            <TextBlock Margin="20" Text="Original image" FontSize="30" />
            <TextBlock x:Name="txtPath" Text="" />
            <Image x:Name="originalImage" Width="400" ImageOpened="OnImageOpened" Margin="0,20" />
        </StackPanel>
        <StackPanel Orientation="Vertical" HorizontalAlignment="Center" Margin="10,0" >
            <TextBlock Margin="20" Text="Reduced image" FontSize="30" />
            <TextBlock x:Name="txtReducedPath" Text="" />
            <Image x:Name="reducedImage" Width="400" Margin="0,20" />
        </StackPanel>
    </StackPanel>
</StackPanel>
</Window>

```

The code to handle the events, and to reduce the bitmap is in the [MainWindow.xaml.cs](#) file.

The `OnLoadClicked` function allows you to select an image file and load it into `originalImage`.

The `OnReduceClicked` function calls the `ReduceBitmap` function and saves the result on the file system.

The `ReduceBitmap` function uses the value of `pixelSize` to reduce the bitmap quality.

The following function shows that the parameter `step` corresponds to the value of `pixelSize`:

```

private static Bitmap ReduceBitmap(Bitmap bitmap, int step)
{
    for (int x = 0; x < bitmap.Width; x += step)
    {
        for (int y = 0; y < bitmap.Height; y += step)
        {
            int r = 0;
            int g = 0;
            int b = 0;

```

```
        for (int i = 0; i < step && x + i < bitmap.Width; i++)
        {
            for (int j = 0; j < step && y + j < bitmap.Height; j++)
            {
                Color c = bitmap.GetPixel(x + i, y + j);
                r += c.R;
                g += c.G;
                b += c.B;
            }
        }
        Color avg = Color.FromArgb(r / (step * step), g / (step * step), b / (step
* step));

        for (int i = 0; i < step && x + i < bitmap.Width; i++)
        {
            for (int j = 0; j < step && y + j < bitmap.Height; j++)
            {
                bitmap.SetPixel(x + i, y + j, avg);
            }
        }
    }

    return bitmap;
}
```

The code uses `GetPixel` to get all pixel colors for each block. The code then calculates the average color for that block and uses `SetPixel` to set all the pixels to that color. Setting all pixels takes longer with large pictures. This difference allows us to contrast the performance between x86 and AArch64 modes.

## 3 Deploy the application on the two in one laptop

Now that the application runs on our development PC, we can deploy it to the two in one laptop. This process is less straightforward than needed, but the following walkthrough must help. The official documentation is on [package and deploy page](#) for Windows apps.

### 3.1 Publishing the binaries for each mode

With the following instructions publish the binaries for each mode:

1. Hit right-click **DowngradePicture project** in Solution Explorer and select **Publish** from the context menu. The publish window opens. In this window we can create multiple profiles. In our case, we need two profiles, one for x86 and one for arm64.

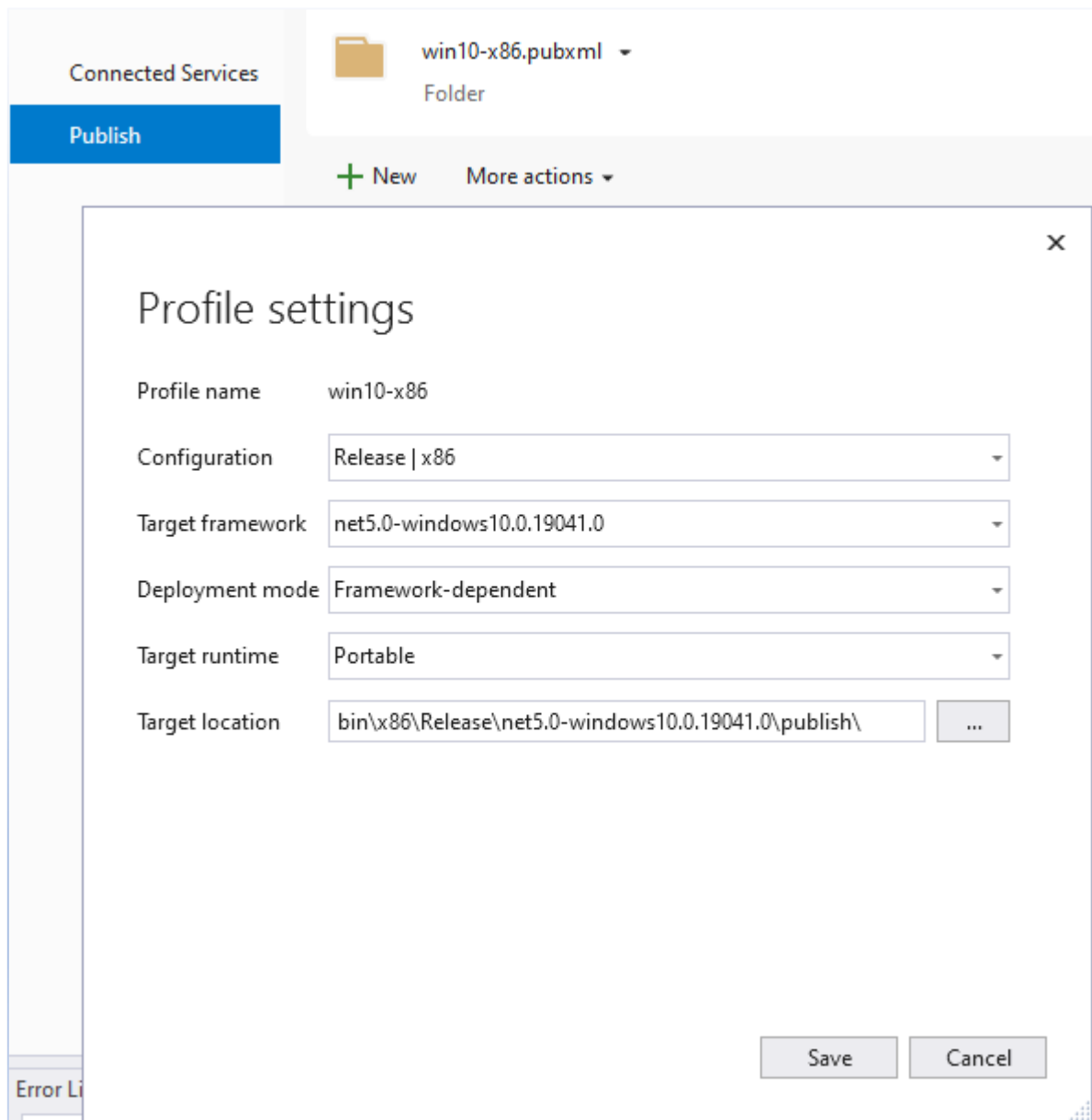


Note

Visual Studio refers to AArch64 as arm64.

---

2. Click the **New** button to create a profile.  
The following profile settings appear for win10-x86:



**Figure: Win 10 x86 profile.**

3. Click **Save**.

The following profile settings appear for win10-arm64:

## Profile settings

Profile name	win10-arm64	
Configuration	Release   arm64	
Target framework	net5.0-windows10.0.19041.0	
Deployment mode	Framework-dependent	
Target runtime	win10-arm64	
Target location	bin\arm64\Release\net5.0-windows10.0.19041.0\publish\	...
<input checked="" type="radio"/> File publish options		

Figure: Win 10 x64 profile.

4. Click **Save**.
5. Click **Publish**, once both profiles are created.

## 3.2 Create the app packages

Use the following instructions to create the app packages:

1. Right-click **DowngradePicture (Package)** project and select **Publish** from the context menu.
2. Finally, click **Create app package**.

A wizard opens with options for distribution as you can see in the following screenshot:

### Select distribution method

How will you distribute this application?

☐ Microsoft Store under a new app name

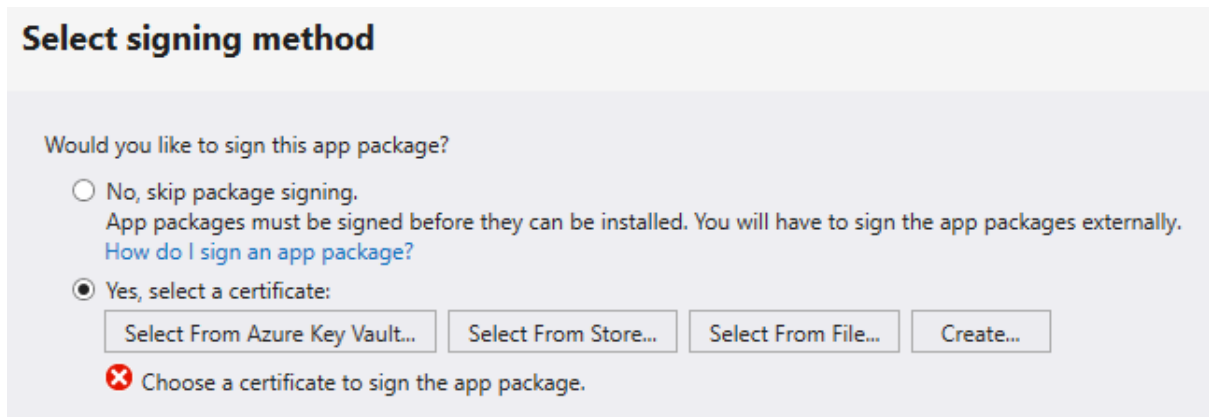
☒ Sideload [What is sideloading?](#)

☒ Enable automatic updates

Figure: App distribution methods



3. Choose **Sideload** to enable package deployment without passing through the Microsoft store. This selection is preferable for testing purposes. Sideload is also a good option when the application is an in-house application that you do not want to make publicly available.
4. Select **Yes, select a certificate**:



**Select signing method**

Would you like to sign this app package?

☐ No, skip package signing.  
App packages must be signed before they can be installed. You will have to sign the app packages externally.  
[How do I sign an app package?](#)

☒ Yes, select a certificate:

Select From Azure Key Vault... Select From Store... Select From File... Create...


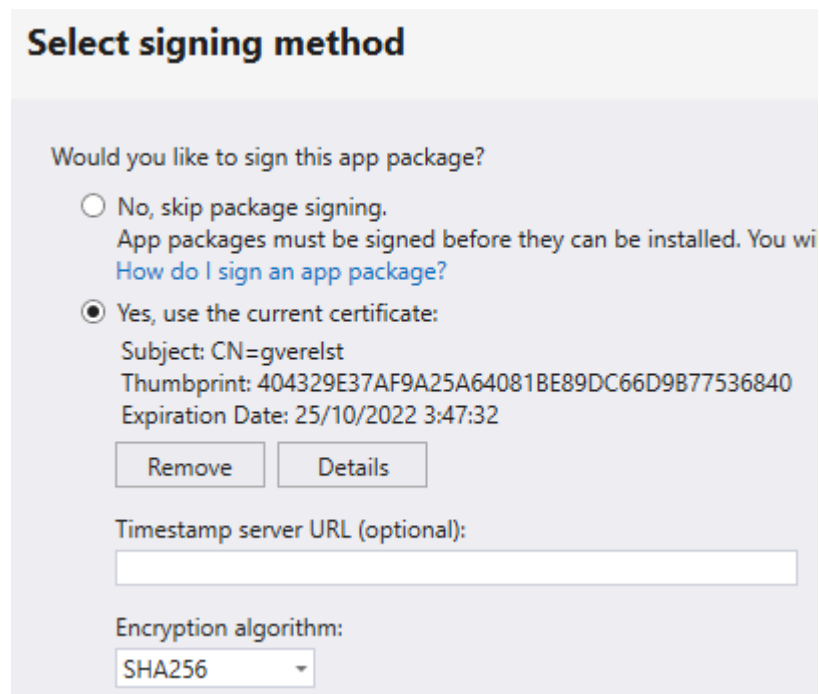
 Choose a certificate to sign the app package.

Figure: Select signing method.

If you already have a certificate, you can select it here. Otherwise, create a self-signed test certificate to use instead. Take note of the password that you use to create it for later use:



**Select signing method**

Would you like to sign this app package?

☐ No, skip package signing.  
App packages must be signed before they can be installed. You will have to sign the app packages externally.  
[How do I sign an app package?](#)

☒ Yes, use the current certificate:

Subject: CN=gverelst  
Thumbprint: 404329E37AF9A25A64081BE89DC66D9B77536840  
Expiration Date: 25/10/2022 3:47:32

Remove Details

Timestamp server URL (optional):

Encryption algorithm:  
SHA256

Figure: Certificate selection

On the next page, select the packages to create. Here, we can package the profiles we have published:

### Select and configure packages

Output location:  
 ...

Version:  
 .  .  .   
☒ Automatically increment  
[How do I use package version numbering?](#)

Generate app bundle:  
   
[What is an app bundle?](#)

Select the packages to create and the solution configuration mappings:

	Architecture	Solution Configuration
<input type="checkbox"/>	Neutral	None
<input checked="" type="checkbox"/>	x86	<input type="text" value="Release (x86)"/>
<input type="checkbox"/>	x64	<input type="text" value="Release (x64)"/>
<input type="checkbox"/>	ARM	None
<input checked="" type="checkbox"/>	ARM64	<input type="text" value="Release (arm64)"/>

☒ Include public symbol files, if any, to enable crash analysis for the app. [How do I use debugging symbols?](#)

**i** Select all architectures for your app to run on the most devices possible. [Which devices support each architecture?](#)

Figure: Select and configure packages.

We cannot select **Release (x64)** because we did not publish an x64 profile. Choosing the x64 wizard results in a Visual Studio error message. For the output location, we have used a shared folder on the development PC. The two in one laptop can access this folder as well, so we do not need to copy and distribute files.

### Configure update settings

Installer location:

[How do I publish my application?](#)

How often should this application check for updates?

☒ Every time the application runs

☐ Every

Figure: Configure update settings.

Click **Create** to create the packages at the installer location. Ignore warnings about files. Everything works if the previous steps have been followed correctly.

### 3.3 Install the package on the two in one laptop

Once you have created a developer certificate, you must export it to the target device. You only must do this export the first time:

1. Copy the **.pfx** file to the shared folder and double-click it.
2. Choose **Local machine** as the store location and click **Next**.
3. Click **Next** after the security confirmation and type the password that you set for the certificate. Leave the other options in their default states.
4. Click **Next** again and select **Trusted Root Certification Authorities**.

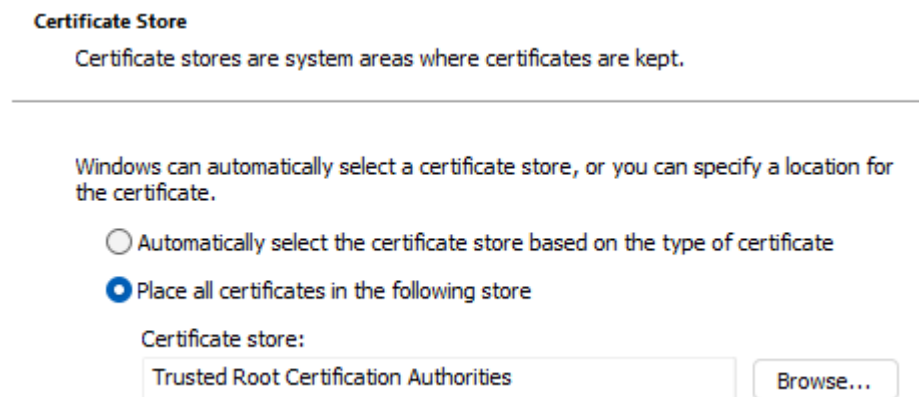


Figure: Certificate store

5. Click **Next** once more to see the overview and click **Finish** to import the certificate.

Now we are ready to install the application on the target PC.

6. Open the **index.html** file in the **publish** folder and click **Get the app** to install and run it.

We have now successfully created a WinUI 3.0 application and installed it on the target device for testing.

## 4 Benchmark the app

We created the application to run in either x86 (emulation) mode or AArch64 (native) mode on the two in one laptop. Having used the same picture in both modes, we can compare the results.

### 4.1 Benchmark X86 emulation

Downgrading the picture in x86 emulation mode took 47.39 seconds. Note that the left-hand side of the following image shows that this mode is running as a 32-bit process (x86):

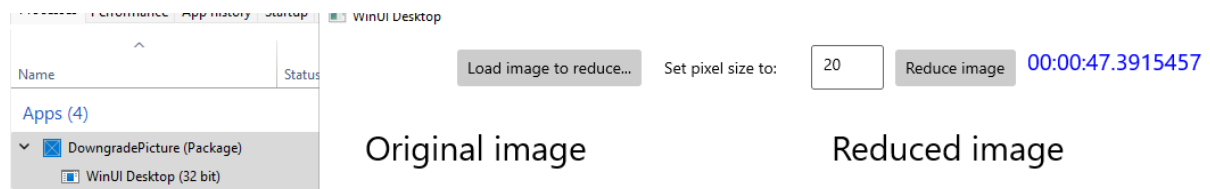


Figure: Downgrading time in x86

### 4.2 Benchmark AArch64 native

In native mode, downgrading the picture takes approximately 24 seconds, nearly twice as fast as in x86 emulation mode. Although this application uses integer-based calculations, we can still appreciate this boost in performance. When **working with floating-point operations**, the improvements are even better, up to 10 times faster as the following screenshot shows:

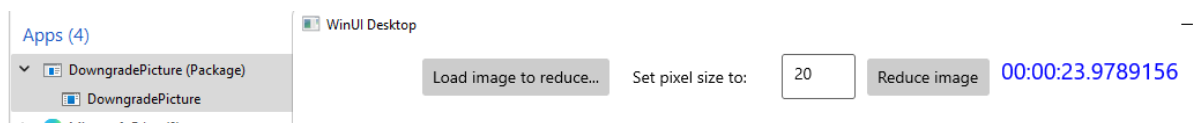


Figure: Downgrading time in x64

## 5 Summary

WinUI 3 has solid support for the AArch64 processor. Visual Studio allows for easy creation of the AArch64 package, which we can deploy on a two in one laptop. Achieving a 100 percent performance gain is especially impressive, considering that we only had to create a win10-arm64 profile in Visual Studio to accomplish it.

As a bonus, this building works on Windows 11 too, without any modification.

## 6 Related Information

Here are some resources related to material discussed in this guide:

Microsoft docs:

- [Build desktop Windows apps with the Windows App SDK - Windows apps | Microsoft docs](#)
- [Create your first WinUI 3 app - Windows apps | Microsoft docs](#)
- [Install tools for Windows app development - Windows apps | Microsoft docs](#)
- [Stable release channel for the Windows App SDK - Windows apps | Microsoft docs](#)
- [Windows UI library \(WinUI\) - Windows apps | Microsoft docs](#)

GitHub.com:

- [Microsoft/WinUI-3-Demos \(github.com\)](#)

## 7 Next steps

This guide has introduced how to build a native Windows on Arm app with WinUI 3. With this knowledge you can fork this application and make something out of it. Let us know what you have done in the comment section on [CodeProject](#). You can also publish your application on the Microsoft store.