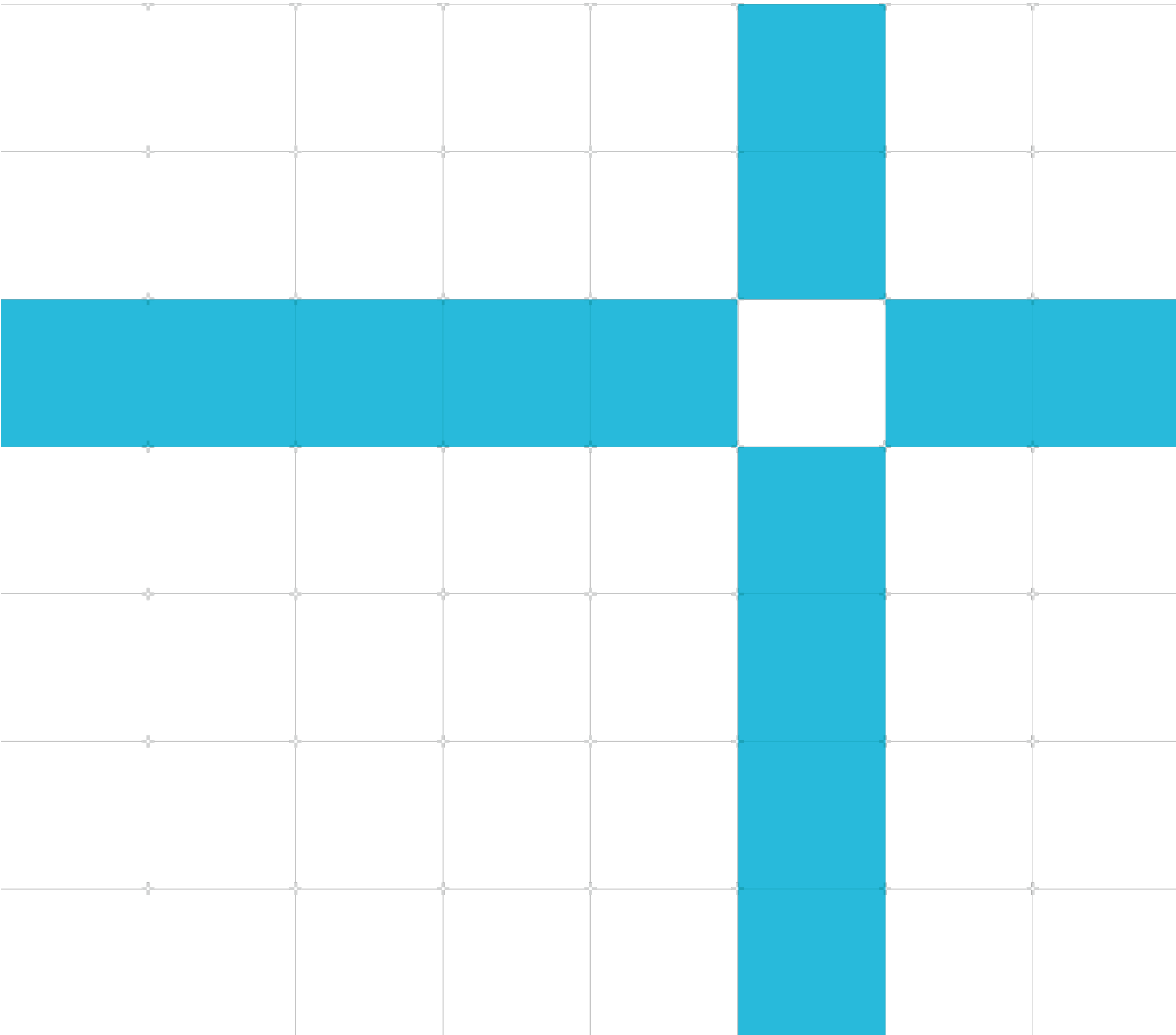
arm

# Building libraries for Windows on Arm

Non-Confidential



**Issue 1.0**
102528



## Building libraries for Windows on Arm



Release information

Document history

| Issue | Date | Confidentiality | Change |
|---|---|---|---|
| 1.0 | 21st June 2021 | Non-Confidential | First release |

# Non-Confidential Proprietary Notice

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

**developer.arm.com**

## Progressive terminology commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive terms. If you find offensive terms in this document, please email **terms@arm.com**.

# Contents

# 1 Overview

Many libraries written in C and C++ provide binary builds that make them easy to use right away. However, if a particular library does not provide a binary for your platform you must build the source yourself before you can use the library.

Libraries that provide binary builds include the following:

- Gaming libraries like SDL
- Encryption libraries like OpenSSL
- Networking libraries like Curl

This guide shows how to use Microsoft Visual Studio to build native Windows on Arm libraries. The guide uses the specific example of the popular zlib library, but you can apply the techniques to any library.

This guide is for:

- Library authors who want to provide Arm native builds
- Library users who need to build a library from source if a Windows on Arm binary is not available

## 1.1 Before you begin

This guide uses Microsoft Visual Studio to compile and build libraries. Visual Studio is only supported on devices using a x86-based or AMD64/x64-based processor. Although Windows on Arm provides x86 emulation, emulation severely impairs the performance of Visual Studio and Microsoft does not support running Visual Studio in this scenario. For more information, see Visual Studio on Arm-powered devices.

To follow this tutorial, you need the following hardware:

- An x86 or AMD64/x64 based host device to run Visual Studio and compile the library
- A Windows on Arm client device to test the library

The host device must have Visual Studio installed, including the Arm build tools. We show you how to install Visual Studio in Install Visual Studio.

In this guide, we use the zlib library as an example to show how to compile a library for Windows on Arm. You must download and unpack the zlib source from www.zlib.net. There are several packaging options to choose from. In this guide, we use zlib source code version 1.2.11, zip file format.

To use the library on your target Windows on Arm device, you must acquire and install the Microsoft C Runtime Library, version 14.00.24234.1 or later, called `vcruntime140.dll`. Microsoft distributes this runtime library as part of Microsoft Visual C++ Redistributable for Visual Studio

2015, 2017 and 2019, available as vc_redist.arm64.exe. For more information, see this information from Microsoft: The latest supported Visual C++ downloads.

# 2 Install Visual Studio

This section gives a brief overview of the installation process, especially the Arm-specific installation options.

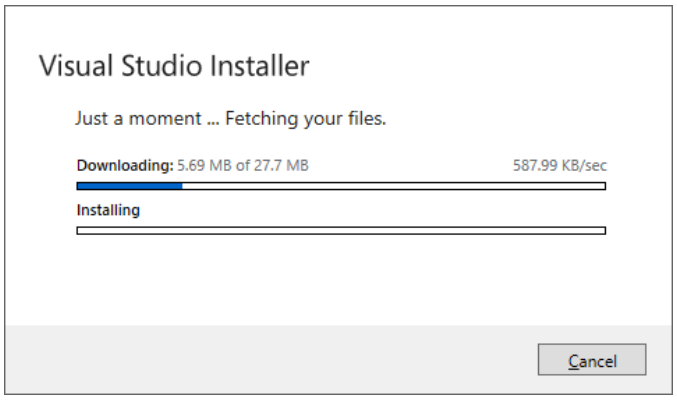
For complete installation instructions, see Install Visual Studio in the Microsoft Visual Studio documentation.

You can download Visual Studio from visualstudio.microsoft.com.

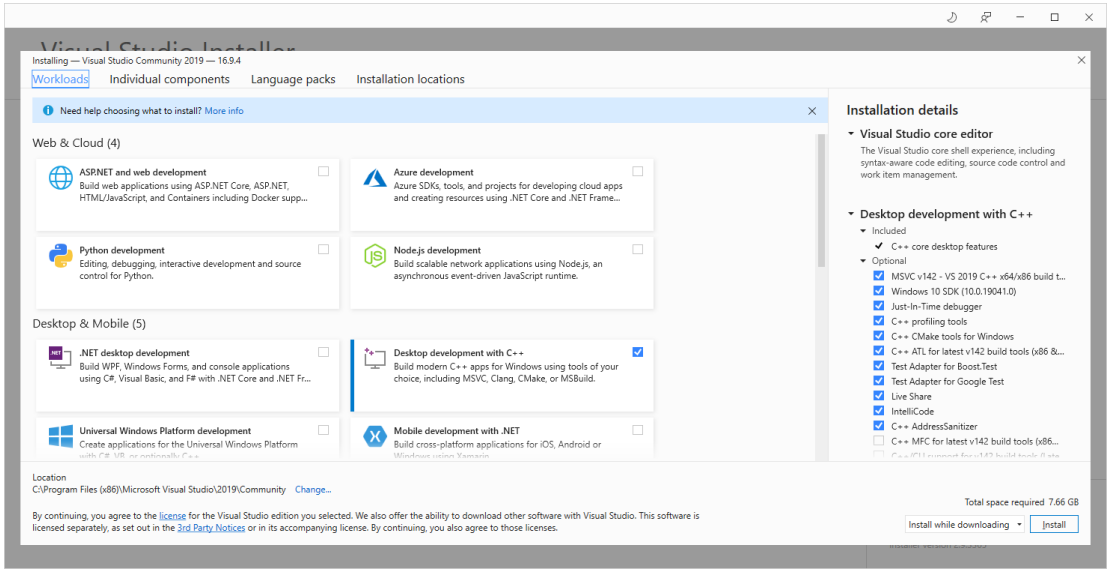
To install Visual Studio, including the Arm build tools, on your x86-based or AMD64/x64-based host device, follow these steps:

1. Run the downloaded installer by double-clicking `vs_community__719936627.1591007999.exe`.

   The installation process begins by downloading the required files, as shown in the following screenshot:

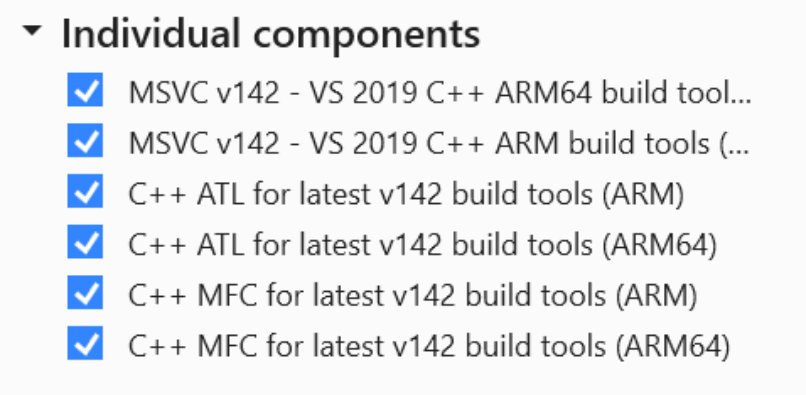



2. Click Desktop development with C++ on the Workloads tab in the Installing – Visual Studio window, as you can see in the following screenshot:



3. Select the following components on the **Individual components** tab:
   - Compilers, build tools, and runtimes section:
     - MSVC v142 – VS 2019 C++ ARM build tools (latest)
     - MSVC v142 – VS 2019 C++ ARM64 build tools (latest)
   - SDKs, libraries, and frameworks section:
     - C++ ATL for latest v142 build tools (ARM)
     - C++ ATL for latest v142 build tools (ARM64)
     - C++ MFC for latest v142 build tools (ARM)
     - C++ MFC for latest v142 build tools (ARM64)

   The **Installation details** > **Individual components** information panel on the right of the **Installing – Visual Studio** window looks like the following screenshot:




4. Click **Install** to complete the installation process.

# 3 Build the library

This section of the guide describes how to build the zlib library for Windows on Arm using Visual Studio on the host device.

We chose zlib to use as an example because it is well known, widely used, and freely available. The zlib library lets you create and extract zip archives.

To build the library, follow these steps:

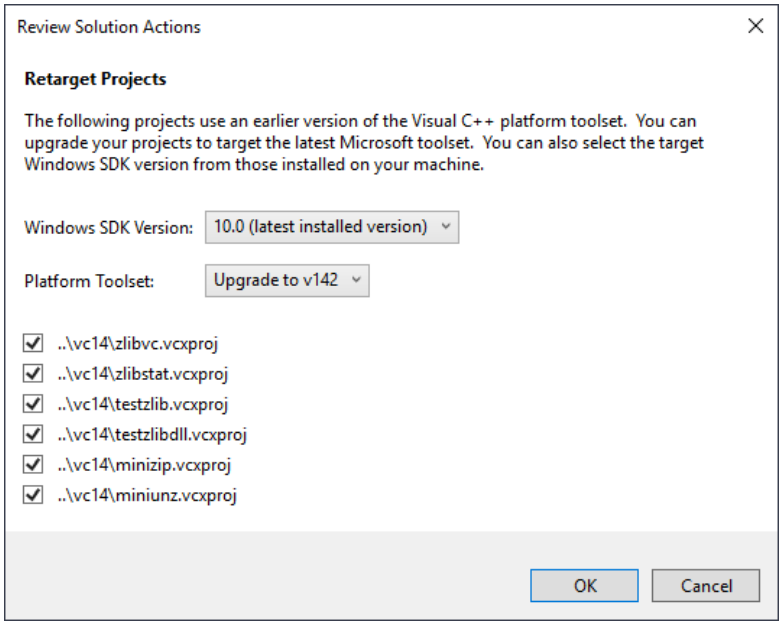
1. Locate and double-click the Visual Studio solution file `zlibvc.sln`. You downloaded and unpacked this file in Before you begin. The file is in the following folder:

   ```
   <install_dir>\zlib-1.2.11\contrib\vstudio\vc14
   ```

   This solution file loads several Visual Studio C/C++ project `.vcxproj` files, located in the same folder.

2. The first time you open `zlibvc.sln`, Visual Studio prompts you to upgrade the solution and the projects so that they build with the latest toolset. Click OK.

   The following screenshot shows this prompt:

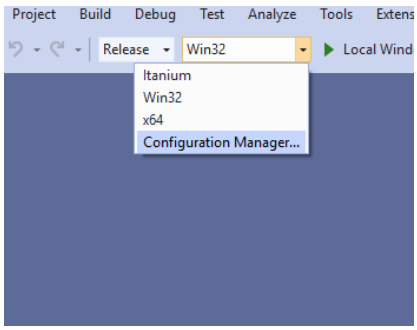



   When this short task finishes, you have a standard Visual C/C++ project that is ready for work.

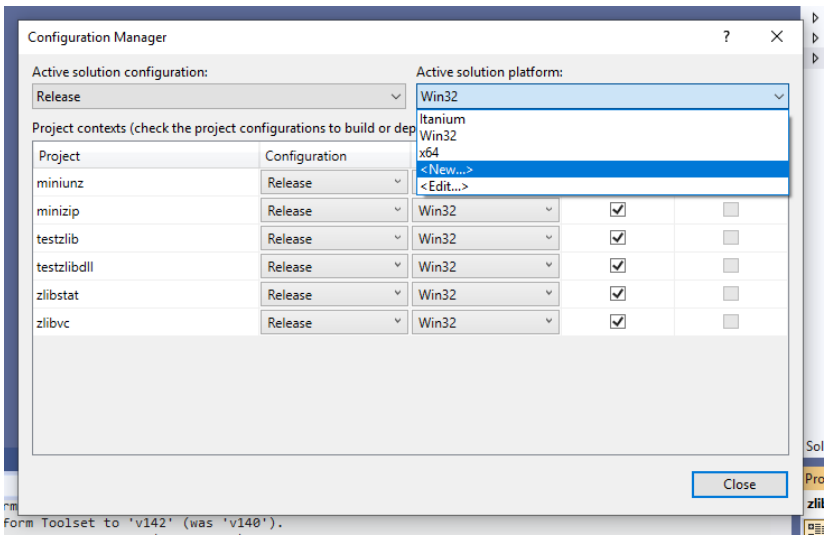
3. Add a new release configuration for the 64-bit Arm platform, ARM64.

   The easiest way to do this is to let the Configuration Manager copy one of the other project configurations.

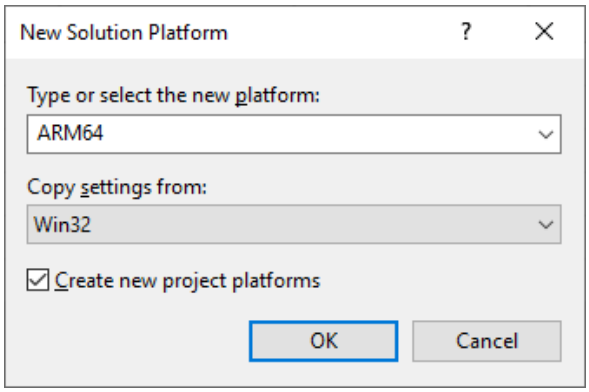
   a. Select **Release** > **Configuration Manager** as shown in the following screenshot:



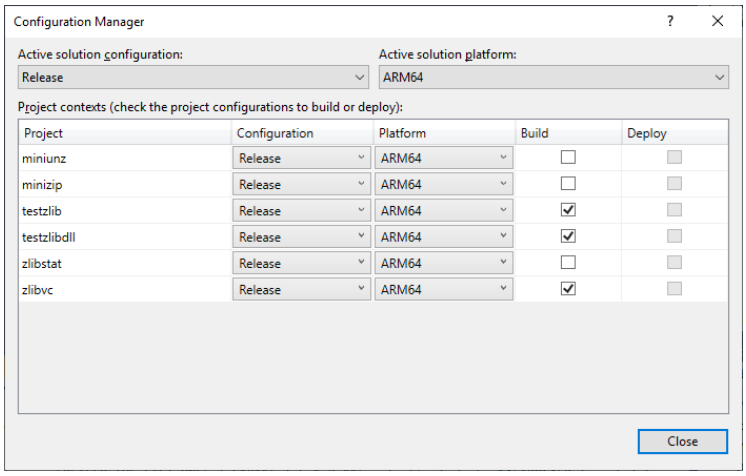

   b. In **Active solution platform**, click **<New...>** as shown in the following screenshot:




   c. Type `ARM64` for the new platform. Copy settings from the existing `Win32` platform. Ensure that the **Create new project platforms** option is enabled, then click **OK** as shown in the following screenshot:



d. Unselect all projects except zlibvc, testzlib, and testzlibdll as shown in the following screenshot:




Note

You can select more projects if you want, but you may need to modify other settings to build them successfully.

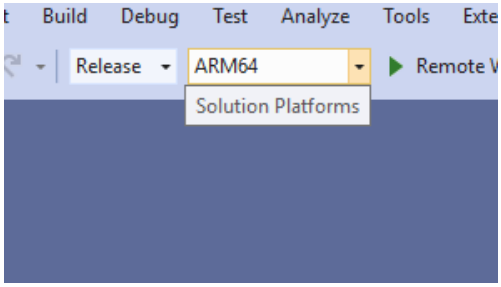
4. Modify the project properties for each of the buildable projects zlibvc, testzlib, and testzlibdll. Select the project, then click **Project** > **Properties**.

   Set the project properties shown in the following table:

| Project | Property | Value |
|---|---|---|
| All projects | Configuration Properties<br>> C/C++<br>> Preprocessor<br>> Preprocessor Definitions | `PLATFORM_IS_ARM64;WIN32;`<br>`_CRT_NONSTDC_NO_DEPRECATE;`<br>`_CRT_SECURE_NO_DEPRECATE;`<br>`_CRT_NONSTDC_NO_WARNINGS;`<br>`ZLIB_WINAPI;%(PreprocessorDefinitions)` |
| All projects | Configuration Properties<br>> C/C++<br>> All Options<br>> Program Database File Name | Give each project a unique file name value, for example zlibvc_db |
| All projects | Configuration Properties<br>> C/C++<br>> Code Generation<br>> Enable Floating Point Exceptions | No (/fp:except-) |
| All projects | Configuration Properties<br>> C/C++<br>> Code Generation<br>> Enable Enhanced Instruction Set | Not set |
| All projects | Configuration Properties<br>> Linker<br>> Command Line<br>> Additional Options | Remove `/MACHINE:I386` |

| | | |
|---|---|---|
| zlibvc and testzlib | Configuration Properties > Linker > Input > Additional Dependencies | `%(AdditionalDependencies)` |
| testzlibdll | Configuration Properties > Linker > Input > Additional Dependencies | `ARM64\Release\zlibwapi.lib; %(AdditionalDependencies)` |
| All projects | Configuration Properties > Linker > Advanced > Randomized Base Address | Yes (/DYNAMICBASE) |
| All projects | Configuration Properties > Linker > Advanced > Image Has Safe Exception Handlers | No (/SAFESEH:NO) |

5. Select the ARM64 Release build configuration by using the Solution Platforms option menus, as shown in the following screenshot:




6. Click Build > Build solution to build the library.
7. Check the console for success:

```
========== Build: 3 succeeded, 0 failed, 0 up-to-date, 3 skipped ==========
```

There are many similarities between libraries built for Intel, and those for Windows on Arm:

- Just like Intel binaries, Arm binaries have properties, version resources, and all the other attributes you would expect.
- Libraries may be either static or dynamic.
- The rules about where dynamic link libraries must go are the same on both platforms.

# 4 Test the library

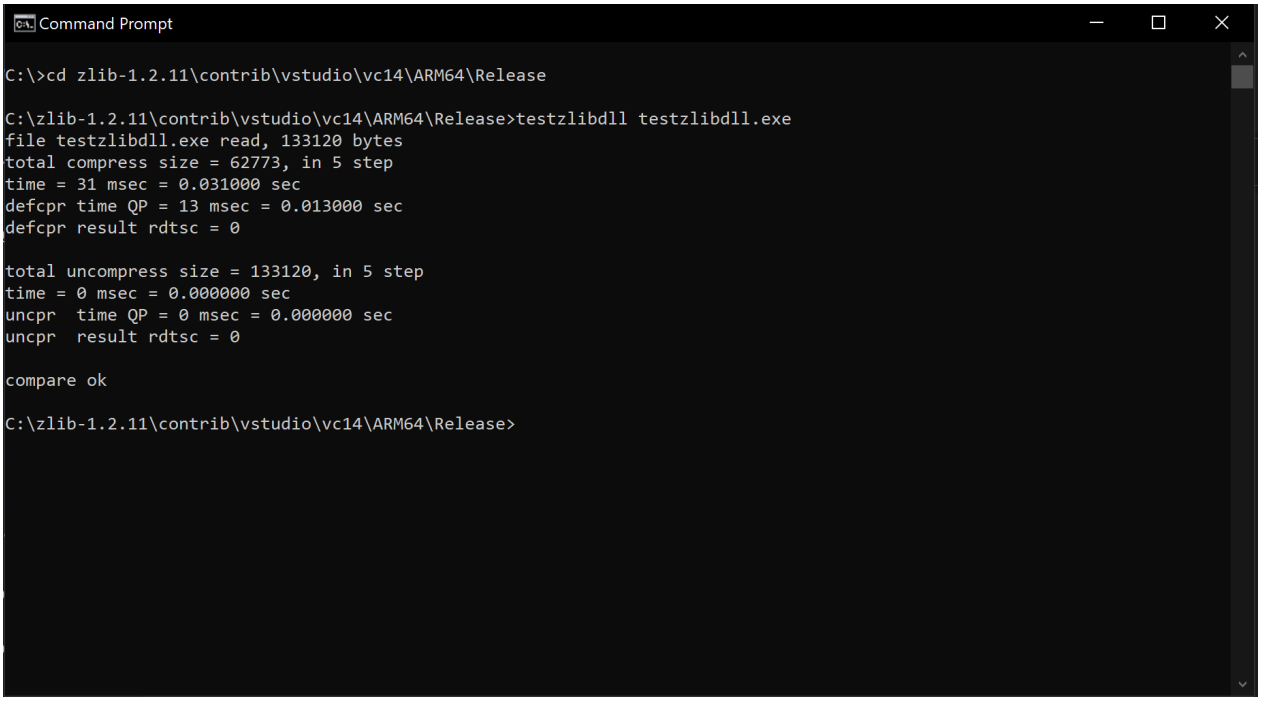
In this section of the guide, we run a test program on the Windows for Arm device to check that the library works.

One of the projects we built in Build the library is a test program, `testzlibdll.exe`. When you run this program, it reports information about its execution.

To run the test program, follow these steps:

1. On your Windows on Arm device, start a Windows command prompt with **Start** > **cmd**.
2. Move to the folder containing the program `testzlibdll.exe`. This will be of the form `<install_dir>\zlib-1.2.11\contrib\vstudio\vc14\ARM64\Release`:
   ```
   > cd C:\zlib-1.2.11\contrib\vstudio\vc14\ARM64\Release
   ```
3. Run the test program:
   ```
   > testzlibdll testzlibdll.exe
   ```

The test program reports information as shown in the following screenshot:




From the test program output, we can see that the Arm64 executable is working correctly.

# 5 Related information

Here are some resources related to material in this guide:

- Building ARM64 Win32 C++ Apps on YouTube
- Microsoft resources:
    - Install Visual Studio
    - Microsoft Visual Studio documentation
    - The latest supported Visual C++ downloads
    - Visual Studio on Arm-powered devices
    - visualstudio.microsoft.com
    - Windows 10 on Arm documentation
- Windows on Arm
- zlib.net

# 6 Next steps

This guide showed you how to use Microsoft Visual Studio to compile and build the zlib library for Windows on Arm devices.

As we have seen, building existing C and C++ code for Windows on Arm is not difficult. If you have any code that uses processor-specific features, you must rewrite that code before proceeding. Otherwise, porting to Arm is usually as simple as installing the Visual Studio tooling and adding new build targets.

You can apply the same process and techniques to build any library for Windows on Arm.

As a next step, you could try building a different library for Windows on Arm. You could also try an open-source library, or create a library of your own.

You can also examine other libraries that have been ported to Windows on Arm. Here are some examples:

- iconv
- libxml2
- LZMA