# Mech-DLK SDK User Manual

v2.0.1

# Table of Contents

# 1. Welcome

Welcome to Mech-DLK SDK User Manual. Let's get started!

## Overview

Mech-DLK SDK is a secondary development software kit specifically designed to be used with Mech-DLK. Its main purpose is to help developers easily do deep learning inference in their software systems. With Mech-DLK SDK, developers can rapidly deploy deep learning models and flexibly integrate deep learning functionality into their own applications without reliance on Mech-Vision. You can use the provided APIs to build applications in C and C# languages.

You can apply Mech-DLK SDK for the inference based on models exported from Mech-DLK (version 2.4.2 or above).

## Release notes

### Mech-DLK SDK 2.0.1

#### New features

- Added C# APIs for secondary development in C# language. See C# API Reference.
- Provided samples using C# APIs and realized collaborative development with OpenCV and HALCON. See the sample list and running prerequisites.

#### Configuration adjustment

- Changed the ways of adding environment variables for running samples provided in Mech-DLK SDK. For more details, see the **Add environment variables** parts in the sample usage guide.

#### Bug fix

- Fixed the result parsing problem that occurred during inference using the defect segmentation model.

## Contents

This manual consists of the following chapters. Click to view the details according to your needs:

| No. | Chapter | Content |
| --- | --- | --- |
| 1 | *Installation guide* | View the system requirements and obtain Mech-DLK SDK and the third-party libraries and resources it depend upon. |
| 2 | *Get started* | Learn to use Mech-DLK SDK for inference with a defect segmentation model. |
| 3 | *Sample usage guide* | Learn about the types of samples and prerequisites to run these samples and build and run these samples. |
| 4 | *API reference* | View APIs and structures in C and C# languages. |

| No. | Chapter | Content |
|-----|---------|---------|
| 5 | *FAQs* | View the frequently asked questions. |

# 2. Installation guide

## System requirements

It is recommended that the device where model inference using Mech-DLK SDK is performed should meet the following requirements.

| Authorized dongle version | Pro-Run | Pro-Train |
| --- | --- | --- |
| Operating system | Windows 10 or above | |
| CPU | Intel® Core™ i7-6700 or above | |
| Memory | 8GB or above | 16GB or above |
| Graphics card | GeForce GTX 1660 or above | GeForce RTX 3060 or above |
| Graphics card driver | Version 472.50 or above | |

> The Pro-Run version features Mech-DLK SDK, labeling, and Operation Mode. The Pro-Train version supports all features, including module cascading, labeling, training, validation, and Mech-DLK SDK.

## Get Mech-DLK SDK

1. Create a local project folder on your device, such as *dlk_sdk*.

2. Clone the repository of Mech-DLK SDK from GitHub to the project folder.

3. Download the third-party libraries (3rd_dll.zip) and resources (resources.zip) that the Mech-DLK SDK relies on to the project folder from Mech-Mind Download Center.

4. Unzip the downloaded packages of third-party libraries and resources. At this point, the project folder should contain the following contents:

📁 3rd_dll

📁 mechdlk_sdk

📁 resources
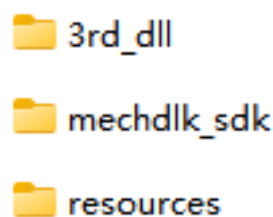
> Do NOT change any of these files and note down the directory for subsequent use.

# 3. Get started

This chapter introduces how to apply Mech-DLK SDK to achieve inference using a defect segmentation model exported from Mech-DLK.

## Prerequisites

- Install Mech-DLK SDK.

- Connect the license dongle provided by Mech-Mind to your device.

- Make sure that CodeMeter is running: In the system tray, check if the CodeMeter icon 🜨 is displayed in the Windows tray.

> ℹ️ If you have installed Mech-DLK on your device, you don't have to install CodeMeter again because it's already in place. Check the Windows tray to make sure that CodeMeter is running.

## Inference flow

Start ▸ Create an inference engine ▸ Load the model ▸ Perform inference ▸ Get results ▸ End

## Function description

In this section, we take the Defect Segmentation model exported from Mech-DLK as an example to show the functions you need to use when using Mech-DLK SDK for model inference.

### Create an input image

Call the following function to create an input image.

```C#
MMindImage image = new MMindImage();
image.CreateFromPath("path/to/image.png");
List<MMindImage> images = new List<MMindImage> { image };
```

```C
MMindImage input;
createImage("path/to/image.png", &input);
```

### Create an inference engine

Call the following function to create an inference engine.

C#

```
InferEngine inferEngine = new InferEngine();
inferEngine.Create("path/to/xxx.dlkpack", BackendType.GpuDefault, 0);
```

ⓘ
- If NVIDIA discrete graphics cards are available on your device, you can set the inference backend, i.e., the second parameter in the function, to GpuDefault or GpuOptimization.
  - When the inference backend is set to GpuOptimization, you need to wait for one to five minutes for model optimization.
- If NVIDIA discrete graphics cards are unavailable on your device, you can only set the inference backend to CPU.
- In this function, the third parameter represents the ID of the used NVIDIA graphics cards, which is 0 when there is only one graphics card. When the inference backend is set to CPU, this parameter is invalid.

C

```
Engine engine;
createPackInferEngine(&engine, "path/to/xxx.dlkpack", GpuDefault, 0);
```

ⓘ
- If NVIDIA discrete graphics cards are available on your device, you can set the inference backend, i.e., the third parameter in the function, to GpuDefault or GpuOptimization.
  - When the inference backend is set to GpuOptimization, you need to wait for one to five minutes for model optimization.
- If NVIDIA discrete graphics cards are unavailable on your device, you can only set the inference backend to CPU.
- In this function, the fourth parameter represents the ID of the used NVIDIA graphics cards, which is 0 when there is only one graphics card. When the inference backend is set to CPU, this parameter is invalid.

## Deep learning engine inference

Call the function below for deep learning engine inference.

C#

```
inferEngine.Infer(images);
```

C

```
infer(&engine, &input, 1);
```

ⓘ In this function, the parameter 1 denotes the number of images for inference, which should equal the number of images in input.

## Obtain the defect segmentation result

Call the function below to obtain the defect segmentation result.

C#

```
List<Result> results;
inferEngine.GetResults(out results);
```

C

```
DefectAndEdgeResult* defectAndEdgeResult = NULL;
unsigned int resultNum = 0;
getDefectSegmentataionResult(&engine, 0, &defectAndEdgeResult,
&resultNum);
```

> ℹ️ In this function, the second parameter 0 denotes the model index in the deep learning model inference package.
>
> - If the inference package is of a single model, the parameter can only be set to 0.
> - If the inference package is of cascaded models, the parameter should be set according to the order of modules in the model inference package.

## Result visualization

Call the function below to visualize the model inference result.

C#

```
inferEngine.ResultVisualization(images);
image.Show("result");
```

C

```
resultVisualization(&engine, &input, 1);
showImage(&input, "result");
```

> ℹ️ In this function, the parameter 1 denotes the number of images for inference, which should equal the number of images in input.

## Release memory

Call the following function(s) to release memory and prevent memory leaks.

C#

```
inferEngine.Release();
```

C

```
releaseDefectSegmentationResult(&defectAndEdgeResult, resultNum);
releaseImage(&input);
releasePackInferEngine(&engine);
```

# 4. Sample usage guide

This chapter introduces you to the usage of C# and C samples.

## C# samples

Check the contents below to learn about the **running prerequisites** and **ways to build and run C# samples**.

Learn about running prerequisites

Run Basic samples

Run an Advanced sample with HALCON

Run an Advanced sample with OpenCV

## C samples

Check the contents below to learn about the **running prerequisites** and **ways to build and run C samples**.

Learn about running prerequisites

Run Basic and Advanced samples

## 4.1. C# (Windows)

Sample list

Two categories of samples are provided: **Basic** and **Advanced**.

- **Basic**: samples using models exported from Mech-DLK to do inference of single images and simultaneous inference of images, as well as obtain and visualize results.

    - ImageInfer

      A sample for inference of single images (both single models and cascaded models are supported)

    - MultiImageInfer

      A sample for simultaneous inference of images (both single models and cascaded models are supported)

- **Advanced**: samples demonstrating collaborative development of Mech-DLK SDK with HALCON/OpenCV.

    - ImageInferWithHalcon

      A sample that runs on the basis of Mech-DLK SDK and HALCON

- ImageInferWithOpenCV

  A sample that runs on the basis of Mech-DLK SDK and OpenCV

## Prerequisites

The usage of C# samples in Mech-DLK SDK is based on the following prerequisites:

- Install required software.
- Added related environment variables.
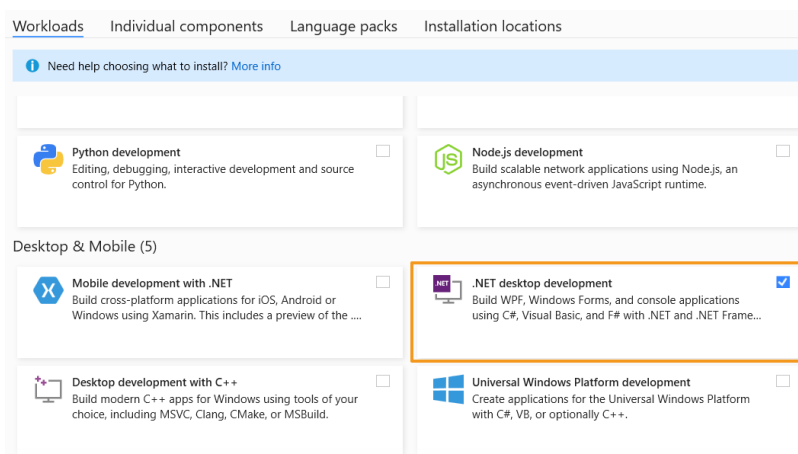
## Install required software

The usage of the C# samples in Mech-DLK SDK requires the installation of Mech-DLK SDK and Visual Studio.

### Install Mech-DLK SDK

Please obtain the latest Mech-DLK SDK and the third-party libraries and resources it depends upon according to the *Installation guide*.

### Install Visual Studio (version 2017 or above)

1. Download the Visual Studio installer.
2. During installation, please select .NET desktop development in the "Desktop & Mobile" category. Then, click [ Install ] in the lower-right corner.



## Add environment variables

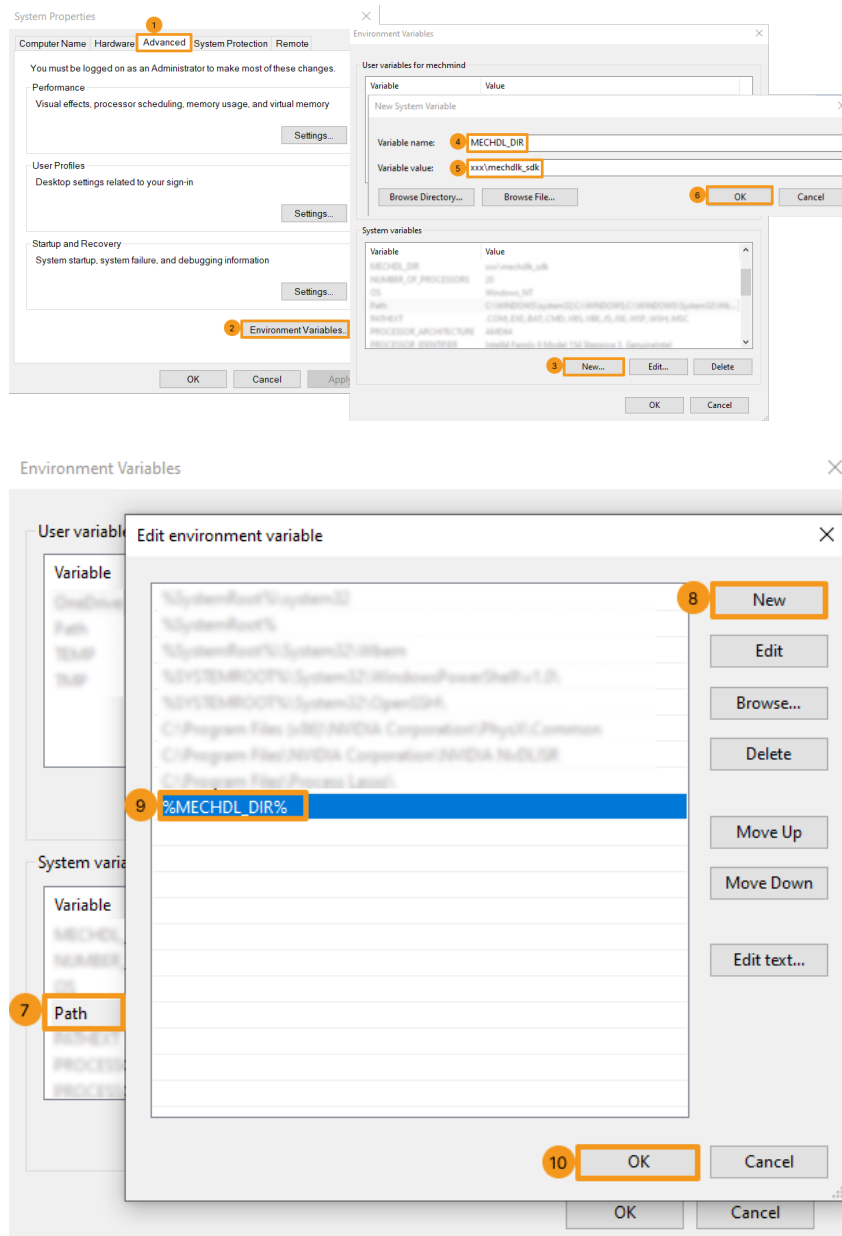You can add the related environment variables by the following steps.

1. Right-click This PC on the desktop and select Properties.
2. Click Advanced system settings and on the Advanced tab of the pop-up System Properties dialog box, click [ Environment Variables ] to open the Environment Variables dialog box.
3. In the System Properties box, click [ New ] and in the pop-up box of New System Variable, enter MECHDL_DIR in the text field of Variable name and *xxx/mechdlk_sdk* in the text field of Variable value. Then, click [ OK ].
4. In the System Properties box, scroll to Path and double-click it to show the Edit System

Variable dialog box.

5. Click [ New ] in the upper-right corner, and add %MECHDL_DIR%. Then, click [ OK ] in the lower-right corner.

> ⛔ If you can find **%MMIND_DLK%** in the **Edit System Variable** dialog box, select it and click [ Delete ] on the right to remove this variable.





## 4.1.1. Run Basic samples

You can build and run the provided samples by the following instructions if you have completed all operations required in the Prerequisites section.

### Build samples

1. Find the `MechDLCShapeSamples.sln` file under *xxx\mechdlk_sdk\samples\csharp* and double-click the file to open the solution in Visual Studio.

2. In the menu bar, select Build › Build Solution. An executable file will be generated and saved to the **bin** folder (path: *xxx\mechdlk_sdk\samples\csharp*).
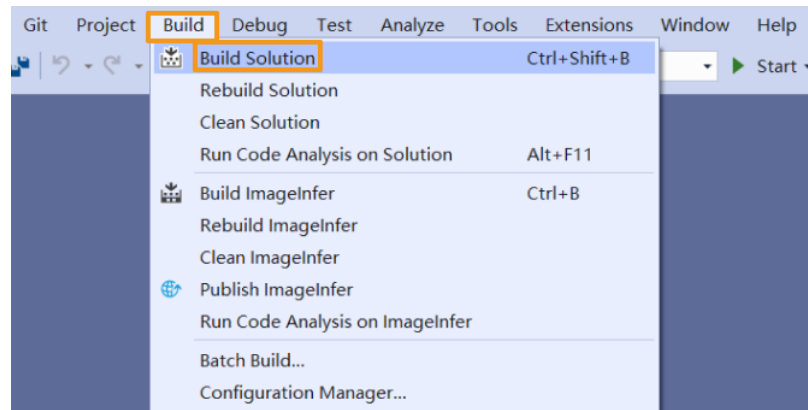


3. Copy and paste the **resources** folder under the project folder to the path *xxx\mechdlk_sdk\samples\csharp\bin*.

4. Copy and paste all files in the **3rd_dll** folder under the project folder to the path *xxx\mechdlk_sdk\samples\csharp\bin*.

5. Copy and paste all files in the **mechdlk_sdk\dll** directory under the project folder to the path *xxx\mechdlk_sdk\samples\csharp\bin*.

ℹ️ | To run any C# sample in Mech-DLK SDK, you only need to perform Steps 3, 4, and 5 once.

## Run samples

You can run the samples in Visual Studio after building them, or you can run the samples by double-clicking the executable files.

### Run a sample in Visual Studio

1. In the **Solution Explorer** panel, right-click a sample and select **Set as Startup Project**.
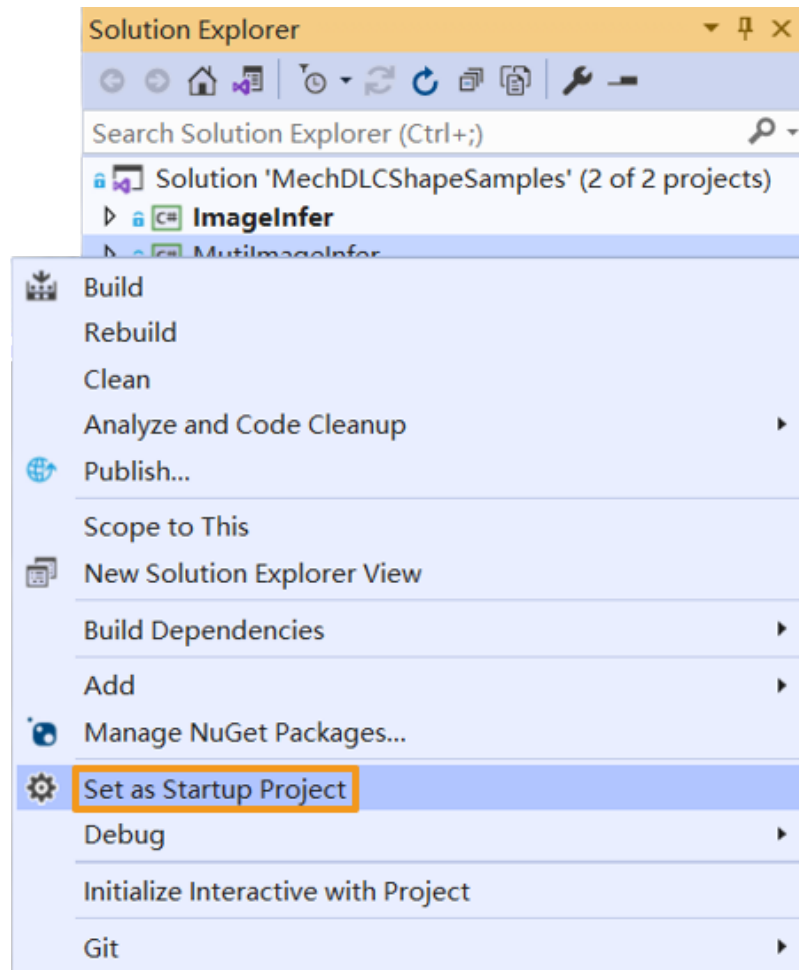
2. Click [ Start ] on the toolbar to run the sample.

**Run the executable file of a sample**

In the **bin** folder (path: *xxx\mechdlk_sdk\samples\csharp\bin*), double-click the executable file named after the sample to run the sample and obtain results.

### 4.1.2. Run Advanced sample with HALCON

You can build and run the provided sample by the following instructions if you have completed all operations required in the Prerequisites section.

**Install HALCON**

1. Download and install HALCON (C# version). Please note down the directory of HALCON.

2. Find the `MechDLCShapeWithHalconSamples.sln` file under *xxx\mechdlk_sdk\samples\csharp* and double-click the file to open the solution in Visual Studio.

3. In the **Solution Explorer** panel, select ImageInferWithHalcon › References. Then, right-click **References** and select **Add Reference**.

4. In the pop-up **Reference Manager** dialog box, click [ Browse ] in the lower-right corner.

5. In the directory of HALCON, find and select the `halcondotnet.dll` file from the path *xxx\bin\dotnet35* and click [ Add ].

6. In the **Reference Manager** dialog box, click **[ OK ]** to finish reference adding.
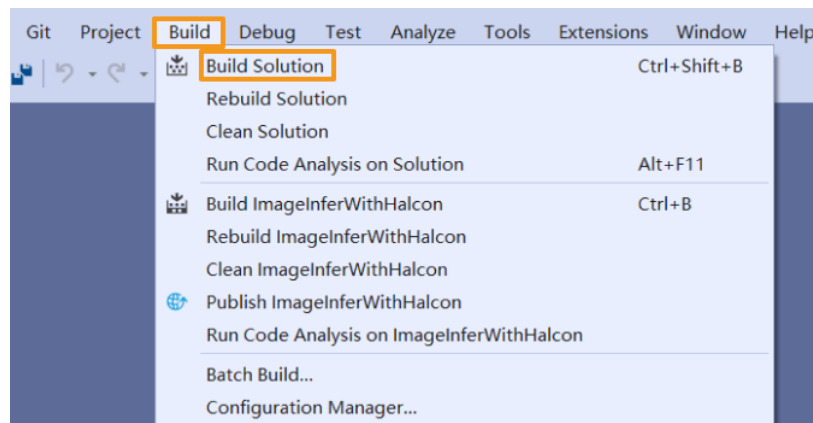
> ℹ️ The provided sample was tested with HALCON 20.11.1.2. The directory of the DLL file may differ for different HALCON versions. Please follow the above steps according to the actual situation.
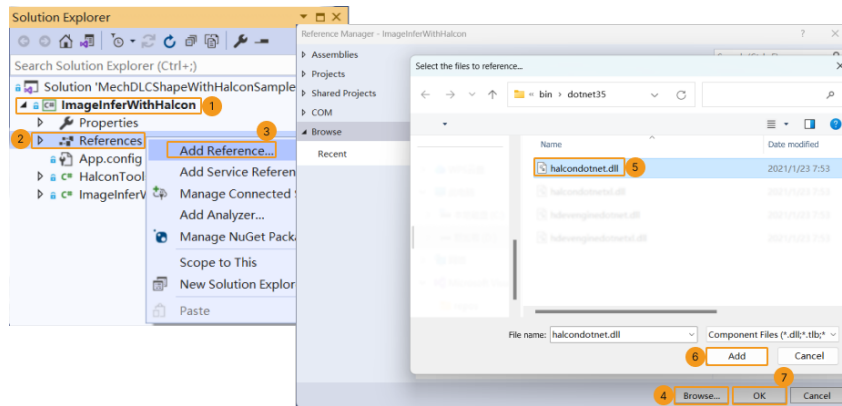


## Build the sample

1. In the Visual Studio menu bar, select Build › Build Solution. An executable file is generated and saved to the **bin** folder (*xxx\mechdlk_sdk\samples\csharp*).



2. Copy and paste the **resources** folder under the project folder to the path *xxx\mechdlk_sdk\samples\csharp\bin*.

3. Copy and paste all files in the **3rd_dll** folder under the project folder to the path *xxx\mechdlk_sdk\samples\csharp\bin*.

4. Copy and paste all files in the **mechdlk_sdk\dll** directory under the project folder to the path *xxx\mechdlk_sdk\samples\csharp\bin*.
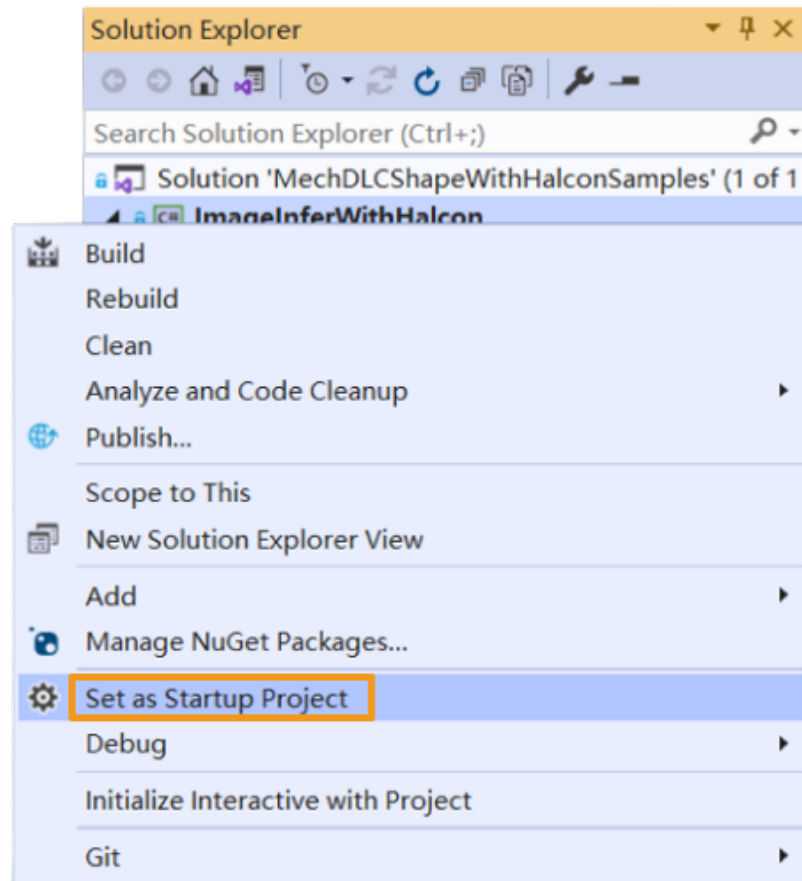
> ℹ️ To run any C# sample in Mech-DLK SDK, you only need to perform Steps 2, 3, and 4 once.

## Run the sample

You can run the sample in Visual Studio after building it, or you can run the sample by double-clicking the executable file.

### Run the sample in Visual Studio

1. In the **Solution Explorer** panel, right-click the sample and select **Set as Startup Project**.

2. Click **[ Start ]** on the toolbar to run the sample.

### Run the executable file of the sample

In the **bin** folder (path: *xxx\mechdlk_sdk\samples\csharp\bin*), double-click the executable file named after the sample to run the sample and obtain results.

### 4.1.3. Run Advanced sample with OpenCV

You can build and run the provided sample by the following instructions if you have completed all operations required in the Prerequisites section.

### Install OpenCV

1. Download and install OpenCV (C# version), namely, OpenCvSharp. Please note down the directory of OpenCvSharp.

2. Find the `MechDLCShapeWithOpenCVSamples.sln` file under *xxx\mechdlk_sdk\samples\csharp* and double-click the file to open the solution in Visual Studio.

3. In the **Solution Explorer** panel, select ImageInferWithOpenCV › References. Right-click **References** and select **Add Reference**.

4. In the pop-up **Reference Manager** dialog box, click **[ Browse ]** in the lower-right corner.

5. In the directory of OpenCvSharp, find and select the `OpenCvSharp.dll` file from the path *xxx\ManagedLib\net461* and then click **[ Add ]**.

6. In the **Reference Manager** dialog box, click **[ OK ]** to finish reference adding.
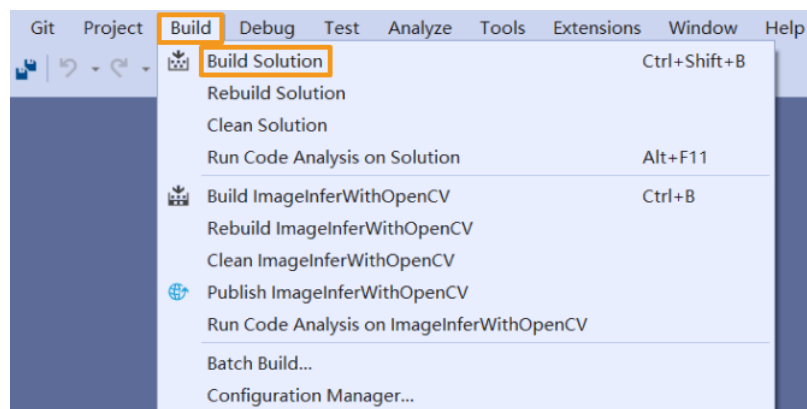
> ℹ️ The provided sample was tested with OpenCvSharp 4.5.3. The directory of the DLL file may differ for different OpenCvSharp versions. Please follow the above steps according to the actual situation.
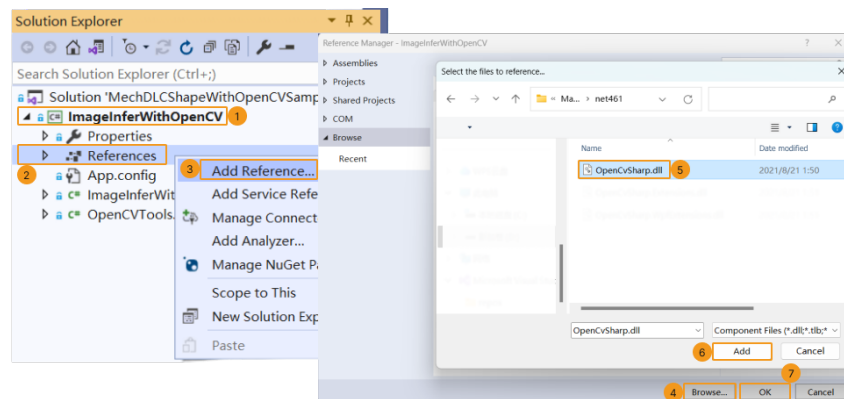


## Build the sample

1. In the Visual Studio menu bar, select Build › Build Solution. An executable file will be generated and saved to the **bin** folder (*xxx\mechdlk_sdk\samples\csharp*).



2. Copy and paste the **resources** folder under the project folder to the path *xxx\mechdlk_sdk\samples\csharp\bin*.

3. Copy and paste all files in the **3rd_dll** folder under the project folder to the path *xxx\mechdlk_sdk\samples\csharp\bin*.

4. Copy and paste all files in the **mechdlk_sdk\dll** directory under the project folder to the path *xxx\mechdlk_sdk\samples\csharp\bin*.

5. In the directory OpenCvSharp, copy and paste the `OpenCvSharpExtern.dll` file in the directory of **NativeLib\win\x64** to the path *xxx\mechdlk_sdk\samples\csharp\bin*.
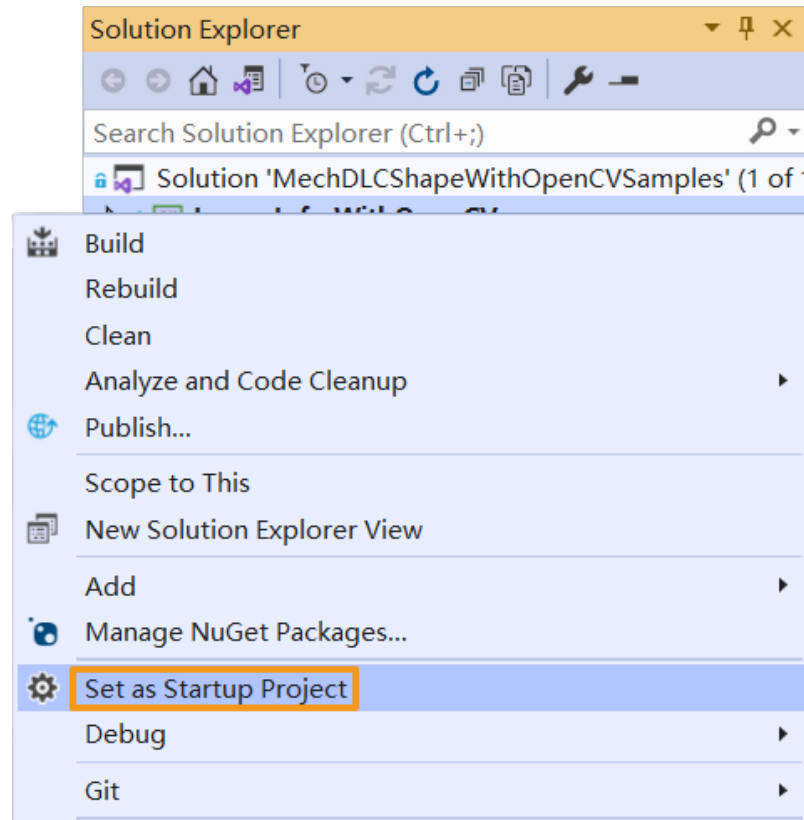
> ℹ️ To run any C# sample in Mech-DLK SDK, you only need to perform Steps 2, 3, and 4 once.

## Run the sample

You can run the sample in Visual Studio after building it, or you can run the sample by double-clicking the executable file.

Run the sample in Visual Studio

1. In the **Solution Explorer** panel, right-click the sample and select **Set as Startup Project**.



2. Click [ Start ] on the toolbar to run the sample.

Run the executable file of the sample

In the **bin** folder (path: *xxx\mechdlk_sdk\samples\csharp\bin*), double-click the executable file named after the sample to run the sample and obtain results.

# 4.2. C (Windows)

On this page, you will learn to run the C samples in Mech-DLK SDK with CMake and Visual Studio on a Windows operating system.

## Sample list

Two categories of samples are provided: **Basic** and **Advanced**.

- **Basic:** samples using single models exported from Mech-DLK to do inference of single images as well as obtain and visualize results.
- **Advanced:** samples for simultaneous inference of multiple images and inference of cascaded models.

## Basic

- Classification: a sample for inference based on the Classification model.

- DefectSegmentation: a sample for inference based on the Defect Segmentation model.
- FastPositioning: a sample for inference based on the Fast Positioning model.
- InstanceSegmentation: a sample for inference based on the Instance Segmentation model.
- ObjectDetection: a sample for inference based on the Object Detection model.

### Advanced

- CascadeModel: a sample for inference based on cascaded models.
- FolderImagesInfer: a sample used to show the inference of images in a folder one by one.
- MultiImageInfer: a sample for simultaneous inference of images.

### Prerequisites

The prerequisites for the use of C samples in Mech-DLK SDK are as follows:

- Install required software.
- Added related environment variables.

### Install required software

Before using the C samples in Mech-DLK SDK, you should install Mech-DLK SDK, CMake, and Visual Studio.

### Install Mech-DLK SDK

Please obtain the latest Mech-DLK SDK and the third-party libraries and resources it depends upon according to the *Installation guide*.

### Install CMake (version 3.2 or above)

1. Download CMake: Select the installer to the right of **Windows x64 Installer**.

Source distributions:

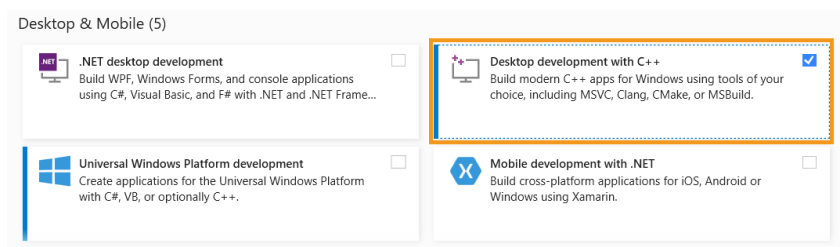| Platform | Files |
|---|---|
| Unix/Linux Source (has \n line feeds) | cmake-3.23.1.tar.gz |
| Windows Source (has \r\n line feeds) | cmake-3.23.1.zip |

Binary distributions:

| Platform | Files |
|---|---|
| Windows x64 Installer | cmake-3.23.1-windows-x86_64.msi |
| Windows x64 ZIP | cmake-3.23.1-windows-x86_64.zip |
| Windows i386 Installer | cmake-3.23.1-windows-i386.msi |
| Windows i386 ZIP | cmake-3.23.1-windows-i386.zip |

2. During installation, select the following two options so that CMake can be added to environment variables, and a desktop icon can be created for CMake.
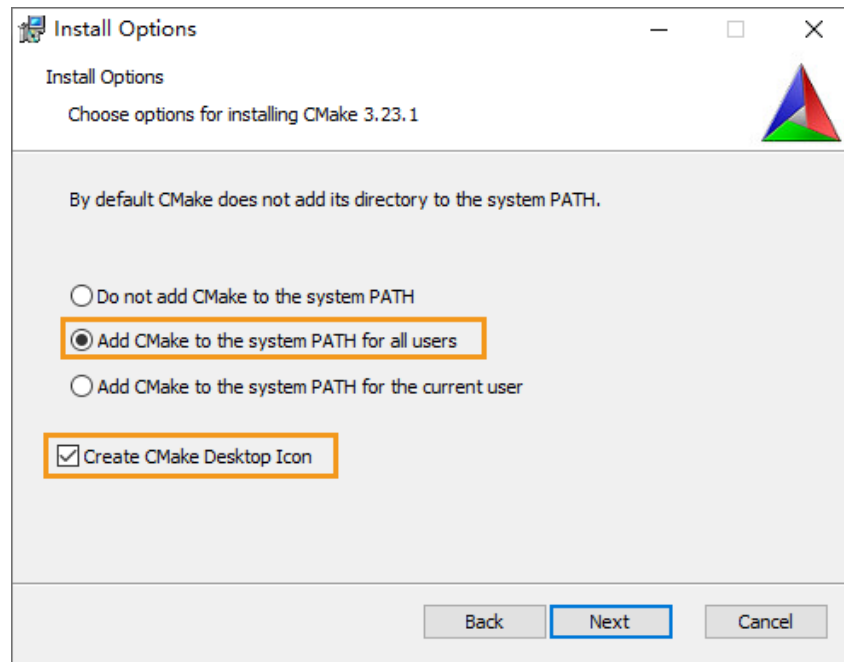
### Install Visual Studio (version 2017 or above)

1. Download the Visual Studio installer.

2. During installation, please select "Desktop development with C++" in the **Desktop & Mobile** category. Then, click **[ Install ]** in the lower-right corner.
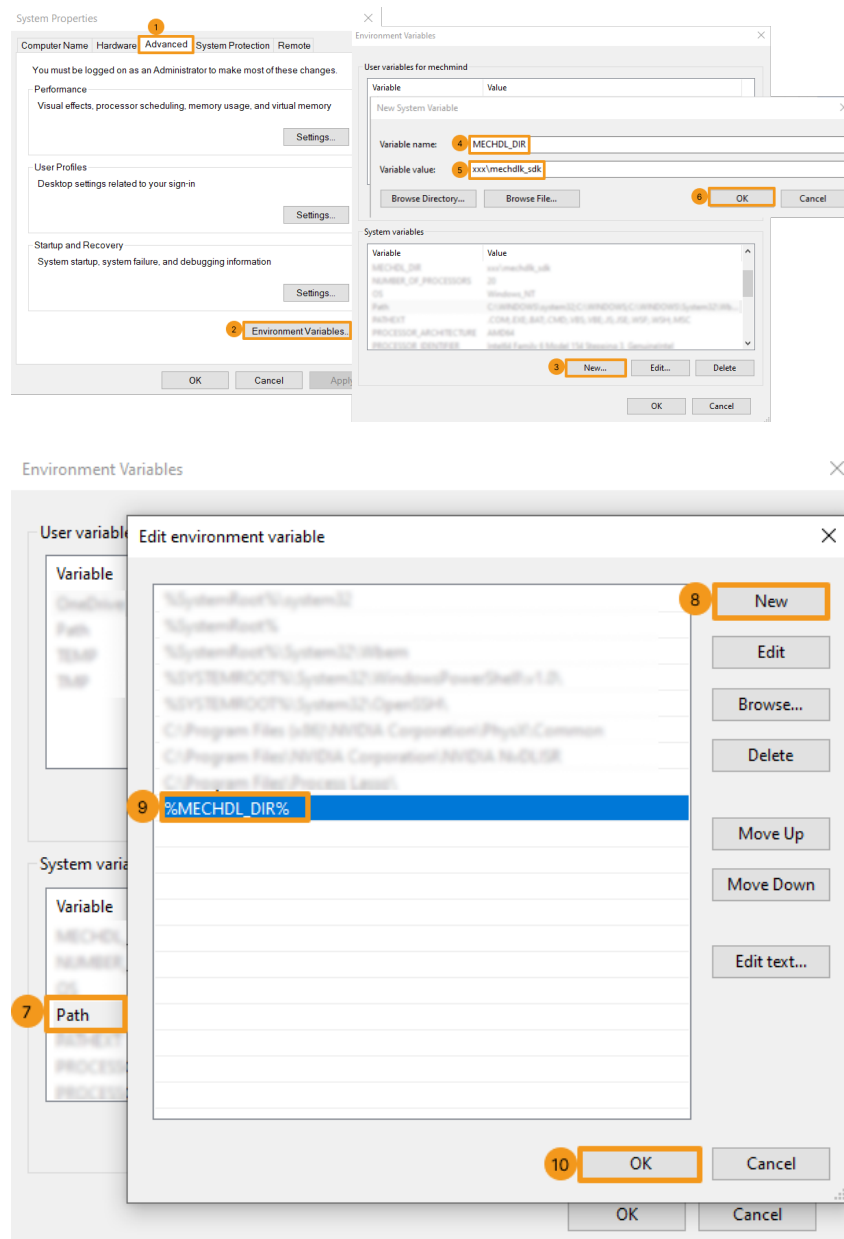


### Add environment variables

You can add the related environment variables by the following steps:

1. Right-click **This PC** on the desktop and select **Properties**.

2. Click **Advanced system settings** and on the **Advanced** tab of the pop-up **System Properties** dialog box, click **[ Environment Variables ]** to open the **Environment Variables** dialog box.

3. In the **System Properties** box, click **[ New ]** and in the pop-up box of **New System Variable**, enter **MECHDL_DIR** in the text field of **Variable name** and *xxx/mechdlk_sdk* in the text field of **Variable value**. Then, click **[ OK ]**.

4. In the **System Properties** box, scroll to **Path** and double-click it to show the **Edit System Variable** dialog box.

5. Click **[ New ]** in the upper-right corner, and add **%MECHDL_DIR%**. Then, click **[ OK ]** in the lower-right corner.

> ⛔ If you can find **%MMIND_DLK%** in the **Edit System Variable** dialog box, select it and click **[ Delete ]** on the right to remove this variable.

## Build and run samples

You can build all samples at a time.

### Configure samples in CMake

1. Double-click the CMake desktop icon to open the software.

2. Enter or select the path to the source code and the path to build the binaries. For the path to build the binaries, you can add \*build* right after the path to the source code if there isn't one and continue the configuration.

| Where is the source code | xxx\mechdlk_sdk\samples\c |
|---|---|
| Where to build the binaries | xxx\mechdlk_sdk\samples\c\build |

3. Click [ Configure ]. When you first click the **Configure** button, a window will pop up to confirm whether you want to create the **build** folder. Click [ Yes ] to create the folder.
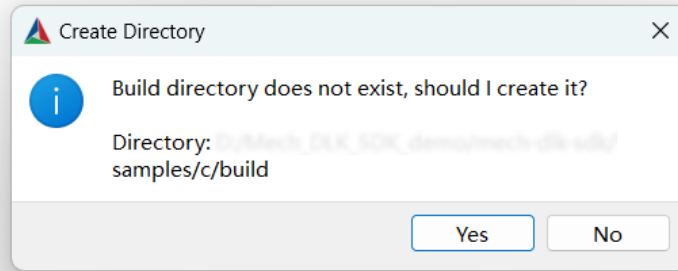
Then, you will enter the configuration window. Select the Visual Studio version and click [ Finish ].
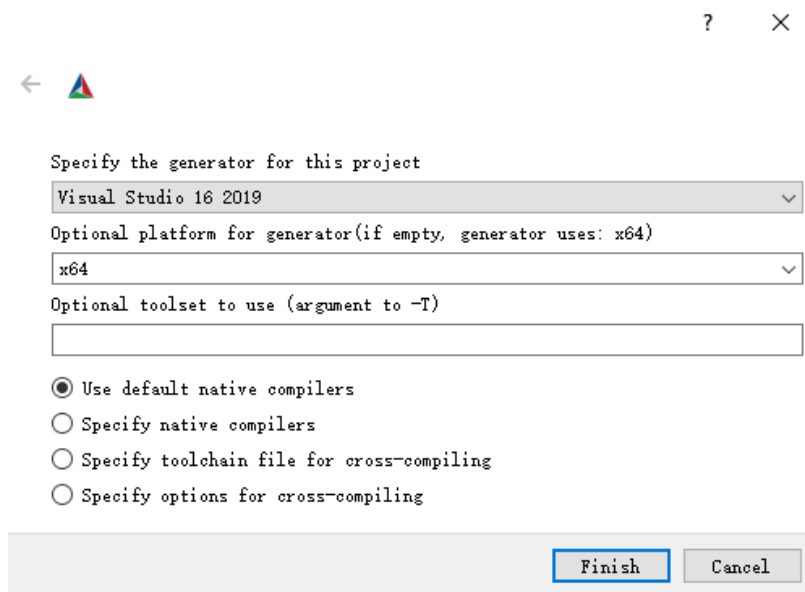


When the configuration completes, the statement **Configuring done** will appear in the last line of the log.
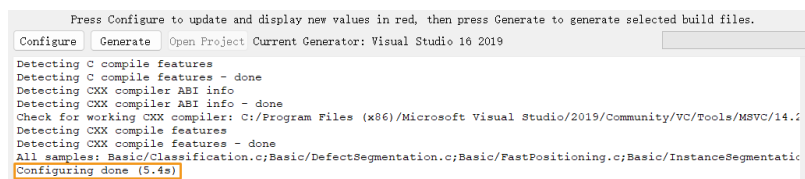


4. Click [ Generate ] to generate Visual Studio solutions. When the generation completes, the statement **Generating done** will appear in the last line of the log. Then, click [ Open Project ] to open the solutions in Visual Studio.
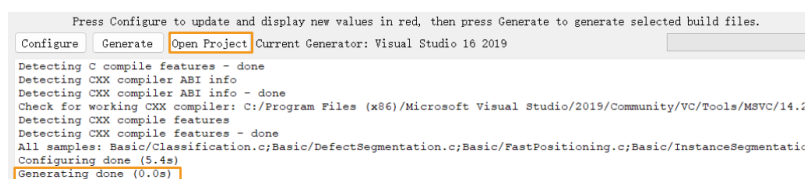


You can also open the created **build** folder (path: *xxx\mechdlk_sdk\samples\c\build*), find **MechDLKSDKCSamples.sln**, and double-click it to open the solutions in Visual Studio.

## Build samples in Visual Studio

1. On the Visual Studio toolbar, confirm that the solution configuration is **Release**. Currently, DLL files for **Debug** are unavailable.

2. In the menu bar, select Build › Build Solution. An executable file (.exe) is generated for each sample. The executable files are saved to the **Release** folder, located in the *Where to build the binaries path* that you entered in CMake.
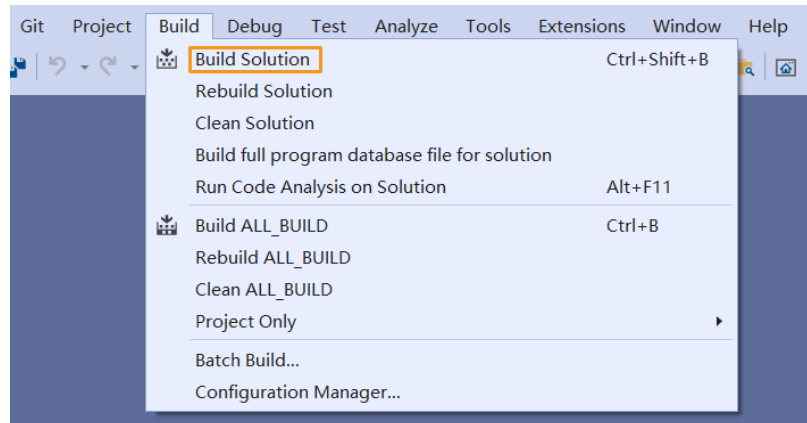


3. Copy and paste the **resources** folder to the following path:
   - *xxx\mechdlk_sdk\samples\c\build\Release*

4. Copy and paste all files in the **3rd_dll** folder under the project folder to the path *xxx\mechdlk_sdk\samples\c\build\Release.*

5. Copy and paste all files in the **mechdlk_sdk\dll** folder under the project folder to the path *xxx\mechdlk_sdk\samples\c\build\Release.*

## Run samples

You can run the samples in Visual Studio after building them, or you can run the samples by double-clicking the executable files.

## Run a sample in Visual Studio

1. In the **Solution Explorer** panel, right-click a sample and select **Set as Startup Project**.
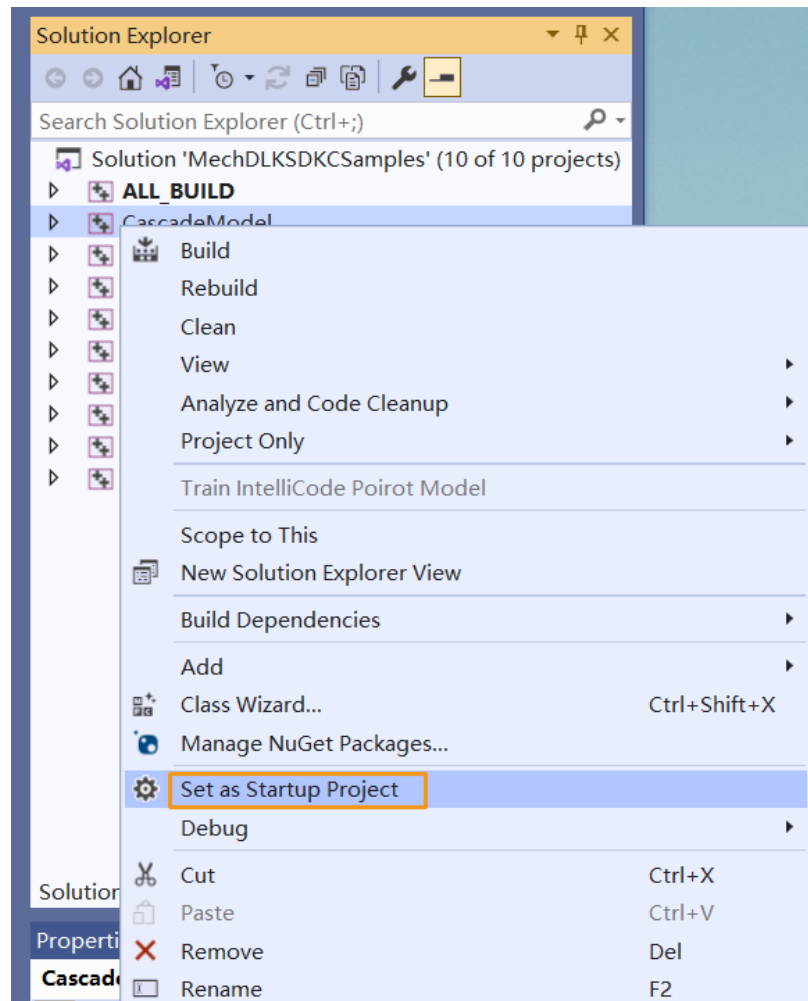
2. Click [ Local Windows Debugger ] on the toolbar to run the sample.

## Run the executable file of a sample

Open the **Release** folder and double-click the executable file (.exe) named after the sample to run the sample and obtain results.

# 5. API reference

## Latest resources

C# API Reference

View details

C API Reference

View details

# 6. FAQs

The following are frequently asked questions and corresponding answers regarding Mech-DLK SDK. If you cannot find the answer you need, please go to the Q&A section of Mech-Mind Online Community to post your questions and exchange ideas.

## Model-related questions

### Can models trained with Mech-DLK be used in other software?

Yes, models trained with Mech-DLK can be used in other software as long as the following requirements are satisfied:

- The software can use Mech-DLK SDK to load the inference model.
- The software can call programs written with APIs in the corresponding language to use the model.

### What kind of models can Mech-Vision / Mech-DLK SDK support?

Mech-Vision / Mech-DLK SDK can only use models trained with Mech-DLK or super models provided by Mech-Mind.

### Can models trained with Mech-DLK be converted into models in other formats?

No, the model format cannot be converted.

## Development-related problems

### What third-party software can be integrated with Mech-DLK SDK?

It has been verified that Mech-DLK SDK supports the integration with LabVIEW to load and call DLLs in C. In addition, as long as third-party software can support the calling of program blocks in C, C#, or C++, it can be integrated with Mech-DLK SDK and thus call program blocks.

### What language of APIs does Mech-DLK SDK support?

You can use the provided APIs to build applications in C and C# languages. We are working on APIs in C++ and Python. If you need to use such APIs, please contact Mech-Mind Technical Support.