



Mech-DLK SDK User Manual

v2.0.2

Table of Contents

1. Welcome	1
2. Installation Guide	2
3. Get Started	3
4. Sample Usage Guide	8
4.1. C# (Windows)	8
4.1.1. Run Basic Samples	11
4.1.2. Run Advanced Sample with HALCON	12
4.1.3. Run Advanced Sample with OpenCV	14
4.2. C++ (Windows)	16
4.3. C (Windows)	23
5. API Reference	30
6. FAQs	31

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#, 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 2.0.2

New Features

- Added C++ APIs for secondary development in C++ language. See [C++ API Reference](#).
- Provided C++ samples and achieved collaborative development with OpenCV. See the [sample list and running prerequisites](#).

Release Notes of Previous Versions

[Mech-DLK SDK 2.0.1 Release Notes](#)

[Mech-DLK SDK 2.0.0 Release Notes](#)

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	See the API reference of multiple languages.
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.

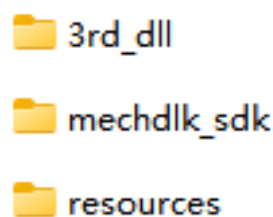
Authorized dongle version	Pro-Run	Pro-Train
Operating system	Windows 10 or above	
CPU	Intel® Core™ i7-6700 or above	
Memory	8GB or above	16GB or above
Graphics card	GeForce GTX 1660 or above	GeForce RTX 3060 or above
Graphics card driver	Version 472.50 or above	



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:




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.



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



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++

```

mmind::dl::MMindImage image;
image.createFromPath("path/to/image.png");
std::vector<mmind::dl::MMindImage> images = {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.dlcpack", 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 parameter is set to `GpuOptimization`, you need to wait for one to five minutes for model optimization. FP16 is valid only under this setting.
- 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++

```
mmind::dl::MMindInferEngine engine;
engine.create(kPackPath);
// engine.setInferDeviceType(mmind::dl::InferDeviceType::GpuDefault);
// engine.setBatchSize(1);
// engine.setFloatPrecision(mmind::dl::FloatPrecisionType::FP32);
// engine.setDeviceId(0);
engine.load();
```



In C++ interfaces, the model parameters can be set according to the actual situation:

- When the `setxxx` function is not called, by default, `BatchSize` is set to `1`; `FloatPrecision` is set to `FP32`; `DeviceId` is set to `0`.
- If NVIDIA discrete graphics cards are available on your device, `InferDeviceType` is set to `GpuDefault`; otherwise, `InferDeviceType` is set to CPU.
- If you need to change the parameters of the inference engine, the `setxxx` function must be placed ahead of the `load()` function.
- When `InferDeviceType` is set to `GpuOptimization`, you need to wait for one to five minutes for model optimization. FP16 is valid only under this setting.

C

```
Engine engine;
```

```
createPackInferEngine(&engine, "path/to/xxx.dlkipack", 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++

```
engine.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++

```
std::vector<mmind::dl::MMindResult> results;
```

```
engine.getResults(results);
```

C

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



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.

Visualize Result

Call the function below to visualize the model inference result.

C#

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

C++

```
engine.resultVisualization(images);
image.show("Result");
```

C

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



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++

```
engine.release();
```

C

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

4. Sample Usage Guide

This chapter introduces you to the usage of C#, 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](#)

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.](#)
- [Add related environment variables.](#)

Install Required Software

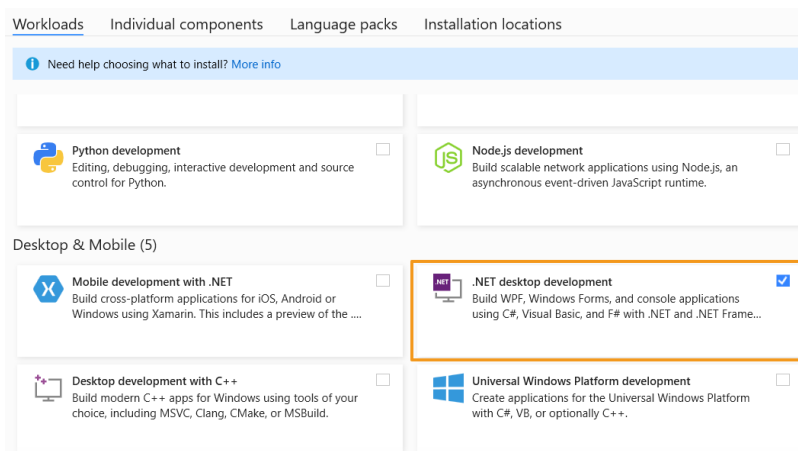
The usage of the C# samples in Mech-DLK SDK requires the installation of Mech-DLK SDK and Visual Studio.

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.



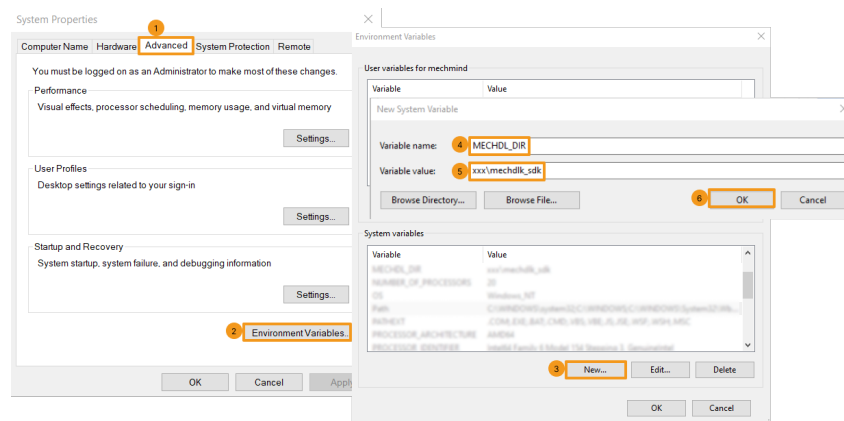
Add Environment Variables

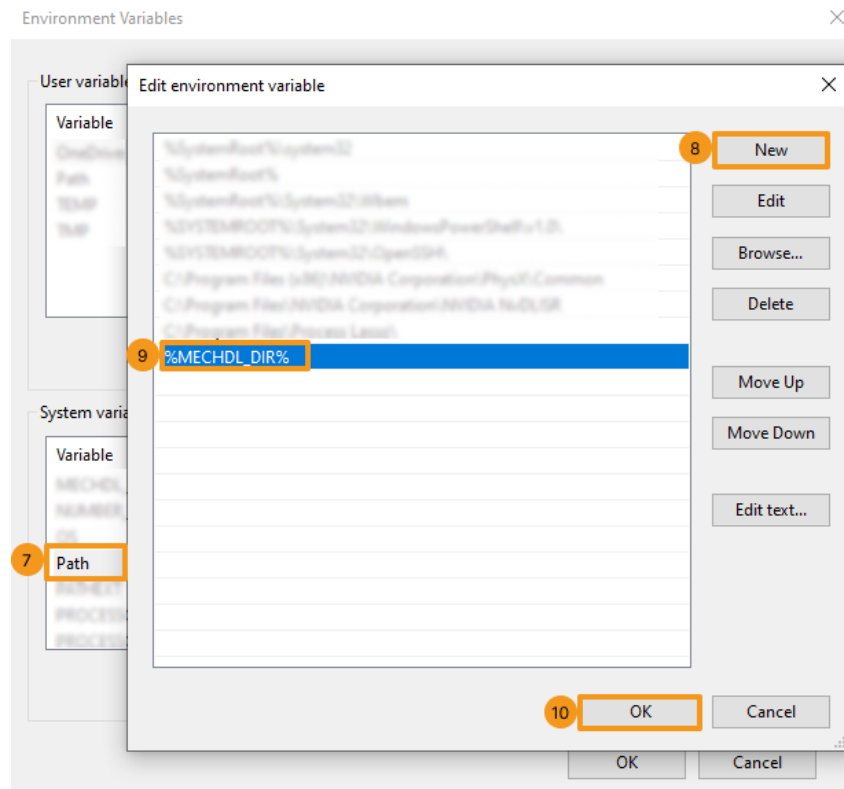
You can add the related environment variables by the following steps.

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.



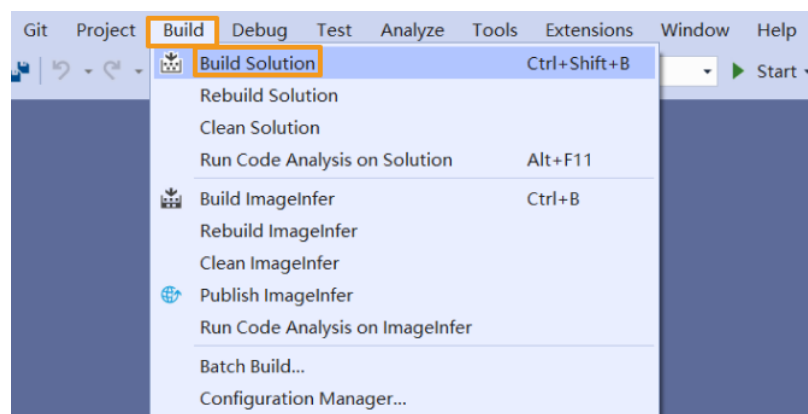


4.1.1. Run Basic Samples

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 `MechDLCSampleSamples.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`.

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

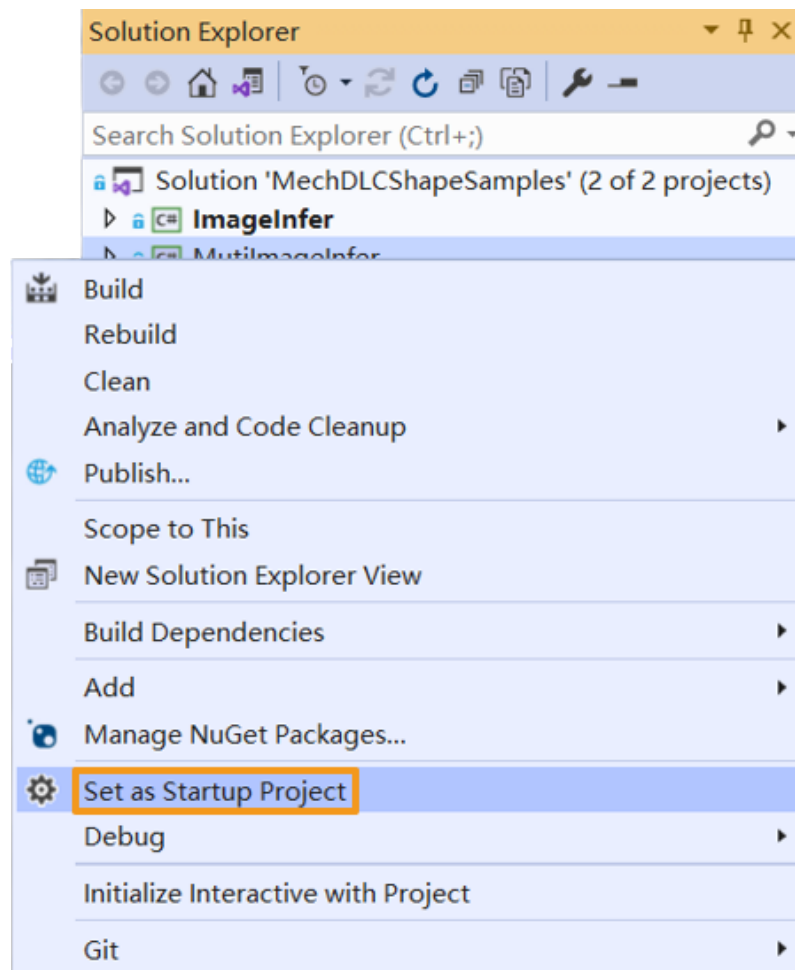
 To run any C# sample in Mech-DLK SDK, you only need to perform Steps 3, 4, and 5 once.

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

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



- 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

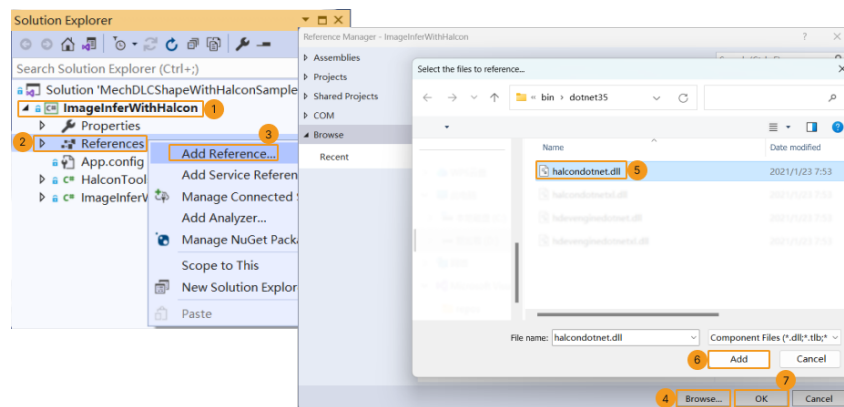
You can build and run the provided sample by the following instructions if you have completed all operations required in the [Prerequisites](#) section.

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.

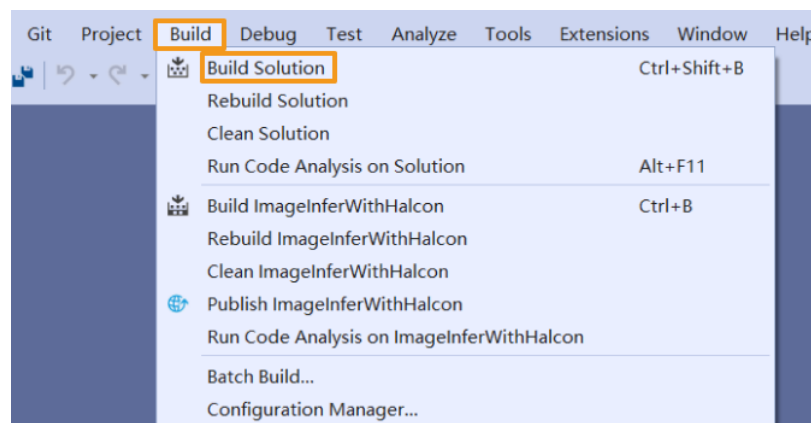


The provided sample was tested with HALCON 20.11.1.2. The directory of the DLL file may differ for different HALCON versions. Please follow the above steps according to the actual situation.



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`.

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

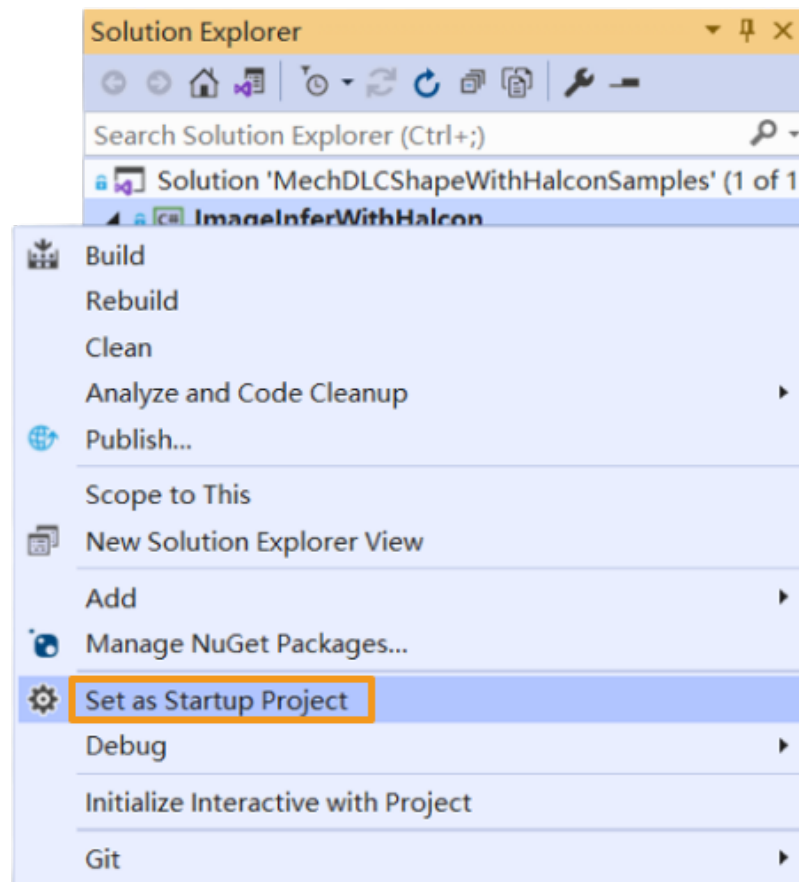
 To run any C# sample in Mech-DLK SDK, you only need to perform Steps 2, 3, and 4 once.

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

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



- 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

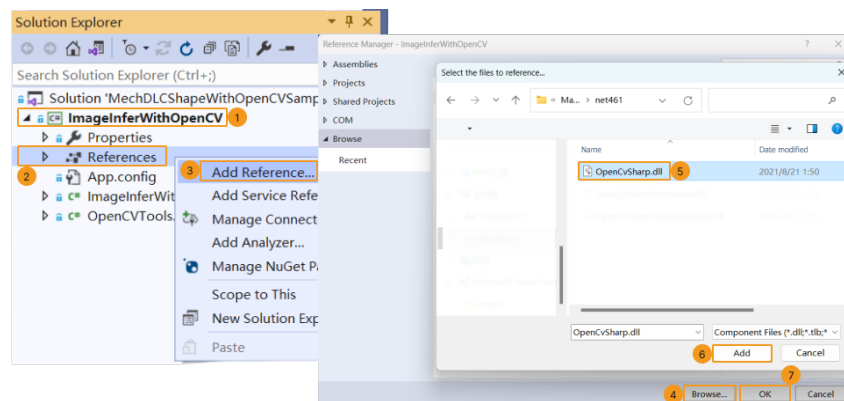
You can build and run the provided sample by the following instructions if you have completed all operations required in the [Prerequisites](#) section.

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.

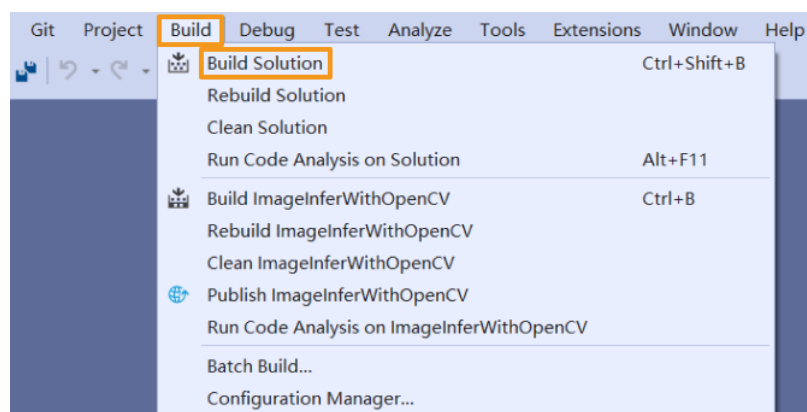


The provided sample was tested with OpenCvSharp 4.5.3. The directory of the DLL file may differ for different OpenCvSharp versions. Please follow the above steps according to the actual situation.



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.`

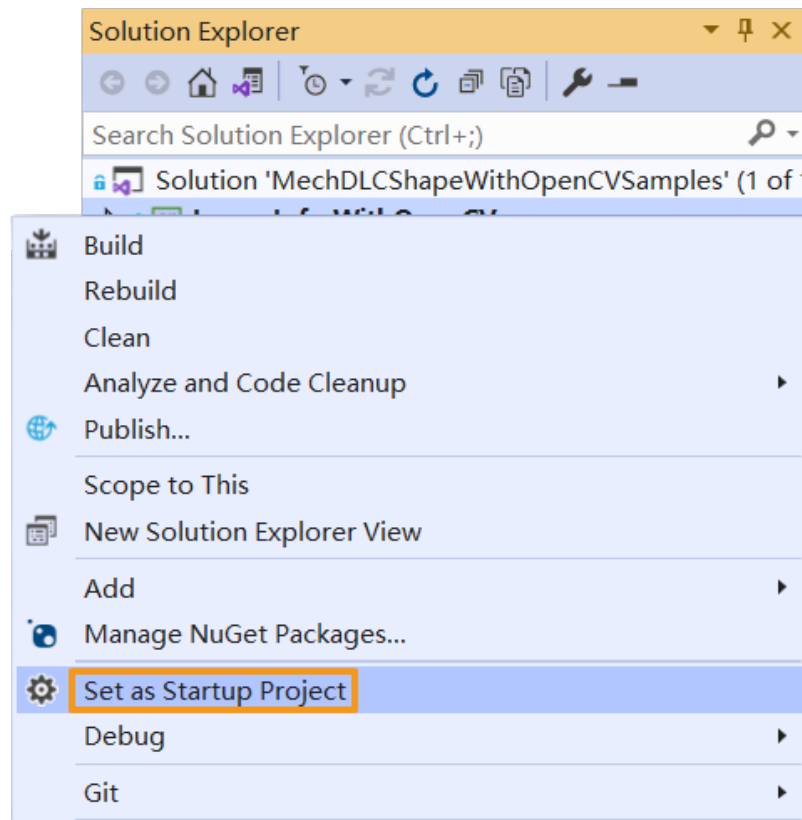
 To run any C# sample in Mech-DLK SDK, you only need to perform Steps 2, 3, and 4 once.

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 and simultaneous inference of multiple images as well as obtain and visualize results.
- **Advanced:** a sample demonstrating collaborative development of Mech-DLK SDK with OpenCV.

Basic

- [ImageInfer](#): a sample for inference of single images.
- [MultiImageInfer](#): a sample for simultaneous inference of images.

Advanced

- [ImageInferWithOpenCV](#): a sample running on the basis of Mech-DLK SDK and OpenCV.

Prerequisites

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

- [Install required software](#).
- [\(Optional\) Install OpenCV](#).
- [Add related environment variables](#).

Install Required Software

In order to use the C++ samples of Mech-DLK SDK, Mech-DLK SDK, CMake, and Visual Studio must be installed.

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**.

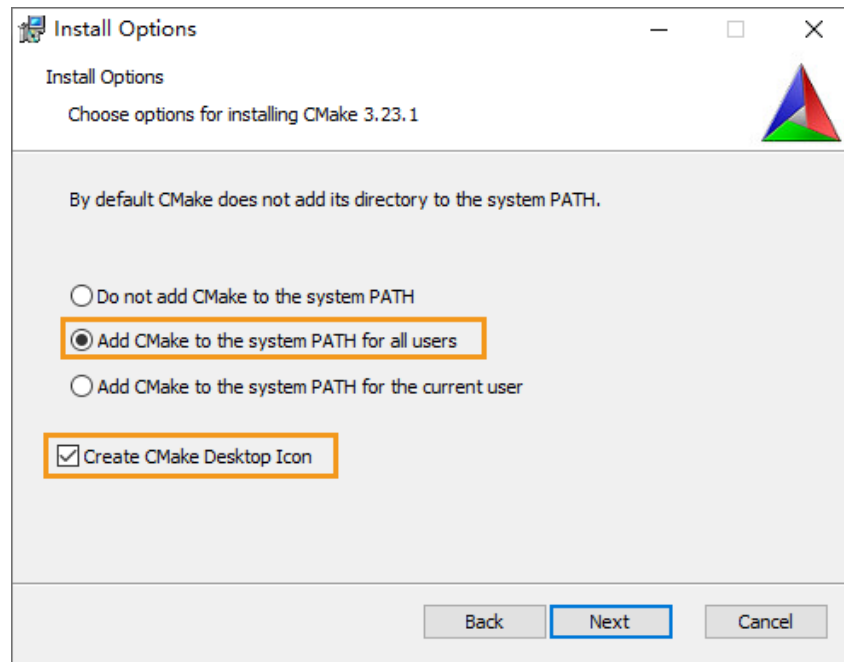
Source distributions:

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:

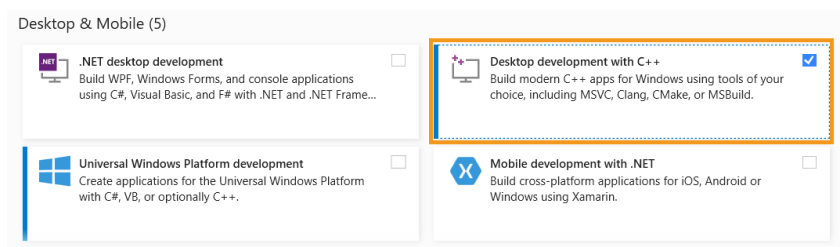
Platform	Files
Windows x64 Installer	cmake-3.23.1-windows-x86_64.msi
Windows x64 ZIP	cmake-3.23.1-windows-x86_64.zip
Windows I386 Installer	cmake-3.23.1-windows-i386.msi
Windows I386 ZIP	cmake-3.23.1-windows-i386.zip

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.



Install OpenCV (Optional)

If you need to run the **Advanced** sample ([ImageInferWithOpenCV](#)), please install OpenCV by the following steps and add it to the system variables.

1. Download and install [OpenCV](#). Please note down the installation directory of OpenCV.
2. After installation, you can find the following files in the directory of OpenCV:

 **build**
 **sources**
 **LICENSE.txt**
 **LICENSE_FFmpeg.txt**
 **README.md.txt**


The sample `ImageInferWithOpenCV` was tested with OpenCV 4.8.0, and results can be obtained.

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**].

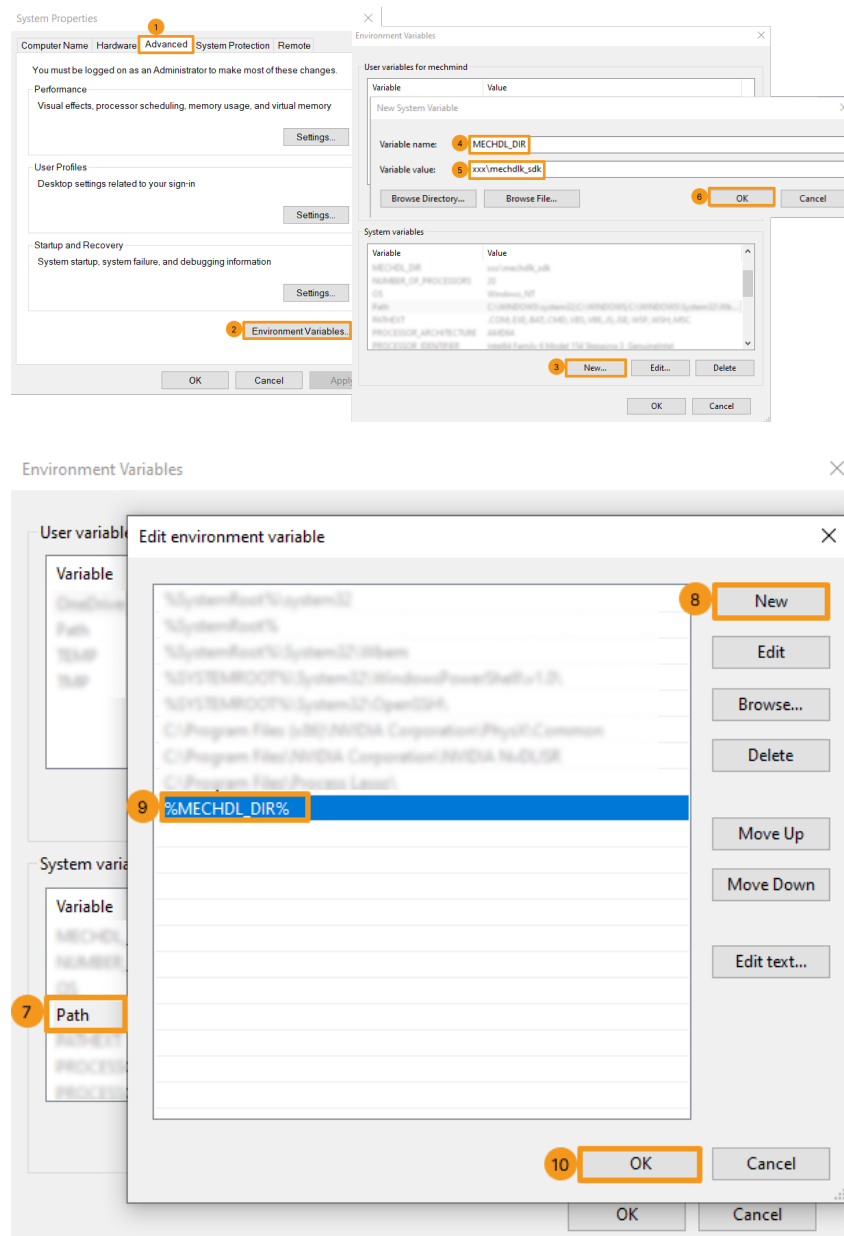


If you want to run the sample `ImageInferWithOpenCV`, please add OpenCV to the environment variables:

1. In the **System Properties** box, click [**New**].
2. In the pop-up box of **New System Variable**, enter **OPENCV_DIR** in the text field of **Variable name** and the path to the **build** folder of OpenCV (`xxx/opencv/build`) 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

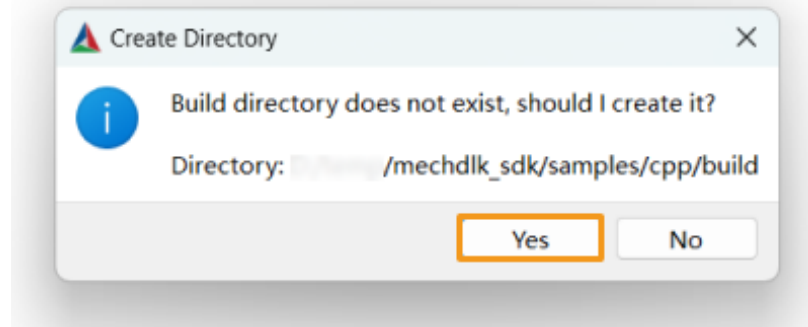
You can build all samples at a time.

Configure Samples in CMake

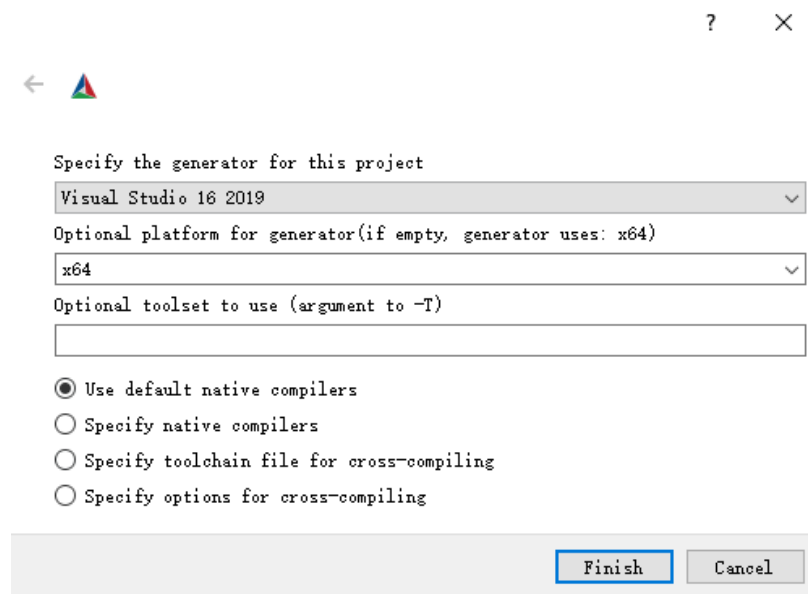
1. Double-click the CMake desktop icon to open the software.
2. Enter or select the path to the source code and the path to build the binaries. For the path to build the binaries, you can add `\build` right after the path to the source code if there isn't one and continue the configuration.

Where is the source code	xxx\mechdlk_sdk\samples\cpp
Where to build the binaries	xxx\mechdlk_sdk\samples\cpp\build

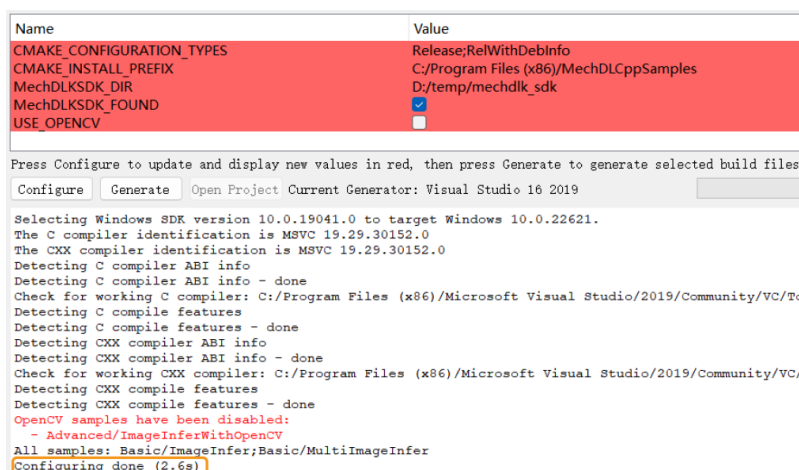
3. Click [**Configure**]. When you first click the **Configure** button, a window will pop up to confirm whether you want to create the **build** folder. Click [**Yes**] to create the folder.



Then, you will enter the configuration page. 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.



Do not select the **USE_OPENCV** option if you want to run **Basic** samples. Select the **USE_OPENCV** option if you want to run the **Advanced** sample.

- 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

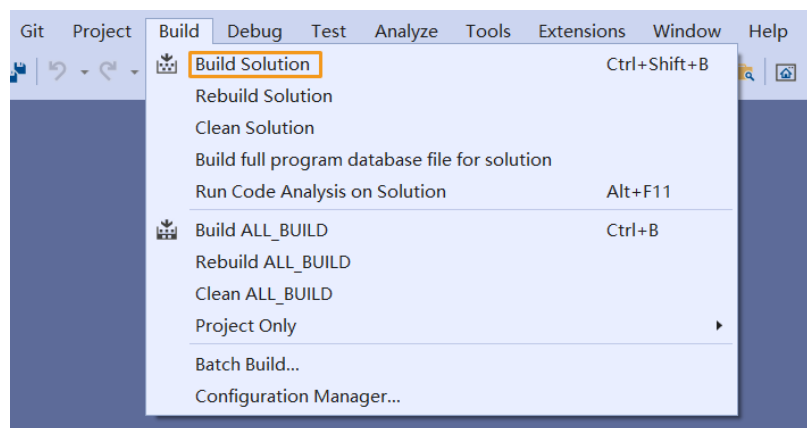
open the solution in Visual Studio.

```

Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/1
Detecting CXX compile features
Detecting CXX compile features - done
OpenCV samples have been disabled:
- Advanced/ImageInferWithOpenCV
All samples: Basic/ImageInfer;Basic/MultiImageInfer
Configuring done (2.6s)
Generating done (0.0s)
    
```

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 is generated for each sample. The executable files are saved to the **Release** folder (path: `xxx\mechdlk_sdk\samples\cpp\build`).



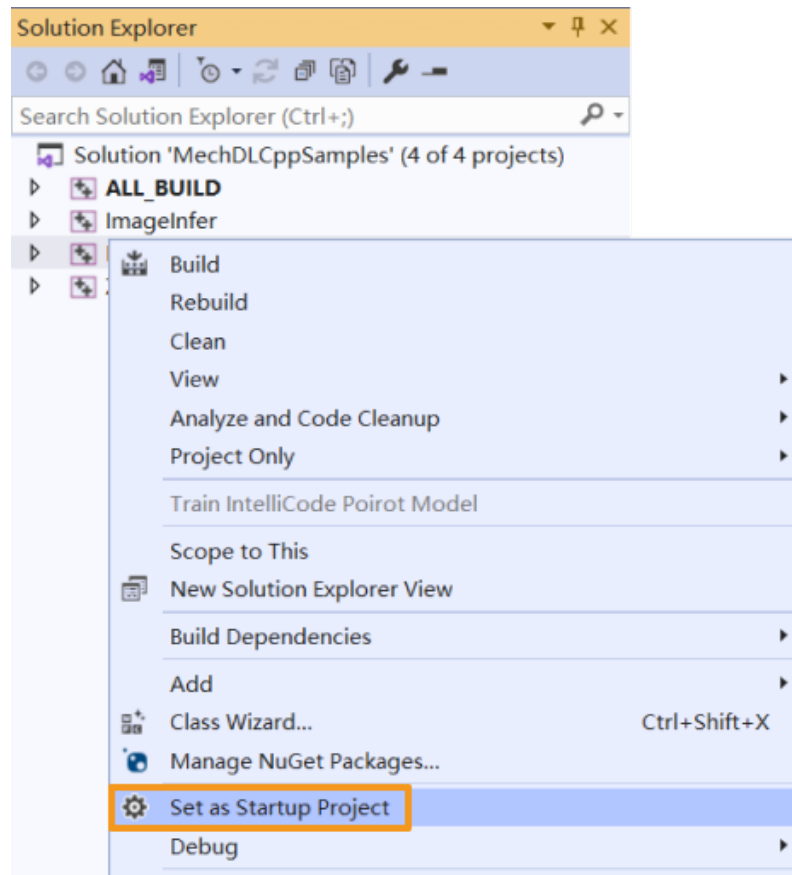
3. Copy and paste the **resources** folder under the project folder to the path `xxx\mechdlk_sdk\samples\cpp\build\Release`.
4. Copy and paste all files in the **3rd_dll** folder under the project folder to the path `xxx\mechdlk_sdk\samples\cpp\build\Release`.
5. Copy and paste all files in the **mechdlk_sdk\dll** folder under the project folder to the path `xxx\mechdlk_sdk\samples\cpp\build\Release`.
6. (Optional) If you need to run the **Advanced** sample, please copy and paste the `opencv_world480.dll` file in the OpenCV directory (`xxx\opencv\build\x64\vc16\bin`) to the path `xxx\mechdlk_sdk\samples\cpp\build\Release`.

Run Samples

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

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 the Sample

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

4.3. C (Windows)

In this section, 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**.

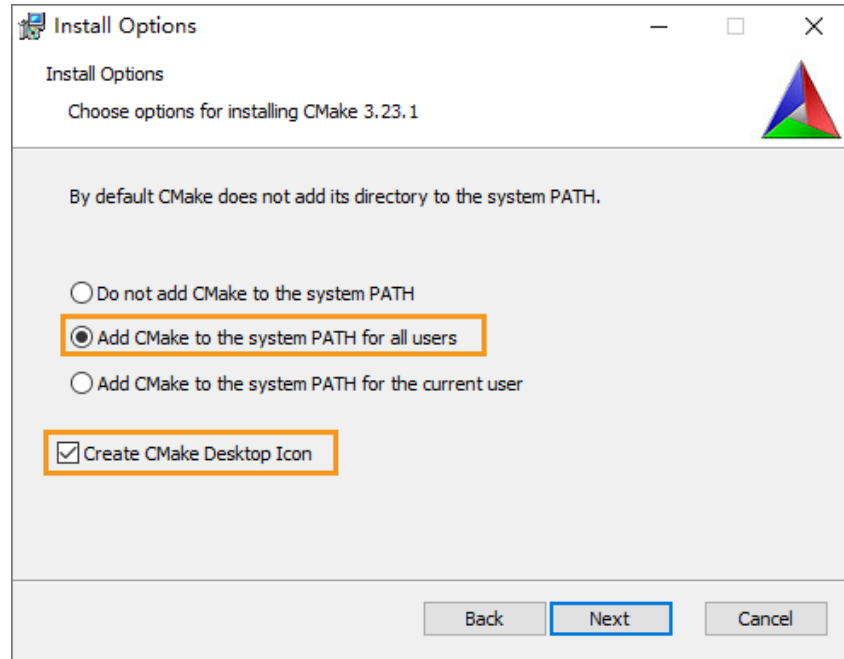
Source distributions:

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:

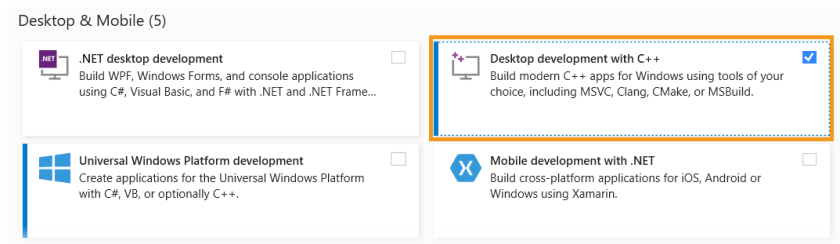
Platform	Files
Windows x64 Installer	cmake-3.23.1-windows-x86_64.msi
Windows x64 ZIP	cmake-3.23.1-windows-x86_64.zip
Windows i386 Installer	cmake-3.23.1-windows-i386.msi
Windows i386 ZIP	cmake-3.23.1-windows-i386.zip

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.



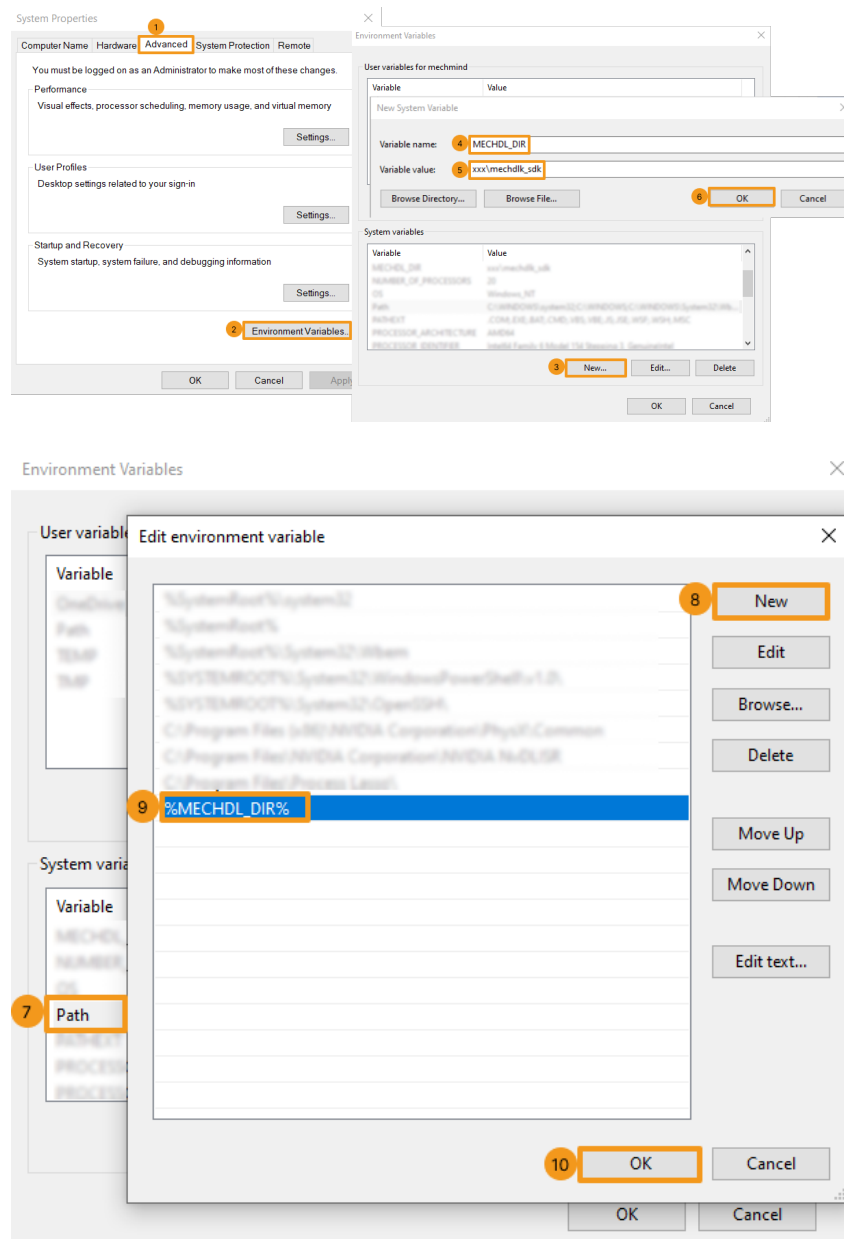
Add Environment Variables

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

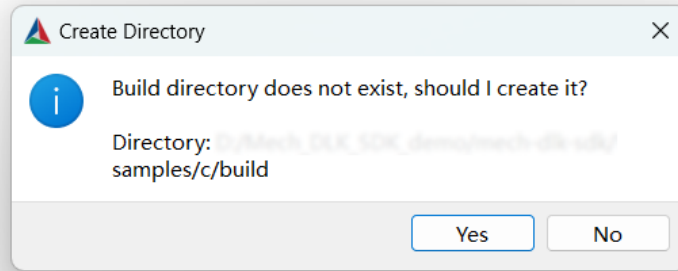
You can build all samples at a time.

Configure Samples in CMake

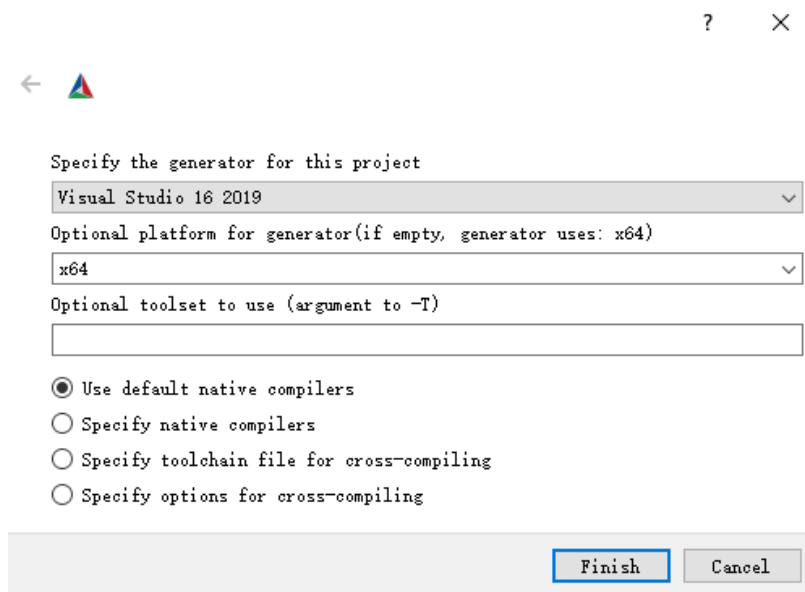
1. Double-click the CMake desktop icon to open the software.
2. Enter or select the path to the source code and the path to build the binaries. For the path to build the binaries, you can add `\build` right after the path to the source code if there isn't one and continue the configuration.

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

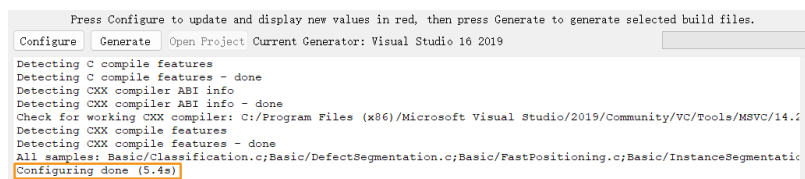
3. Click [**Configure**]. When you first click the **Configure** button, a window will pop up to confirm whether you want to create the **build** folder. Click [**Yes**] to create the folder.



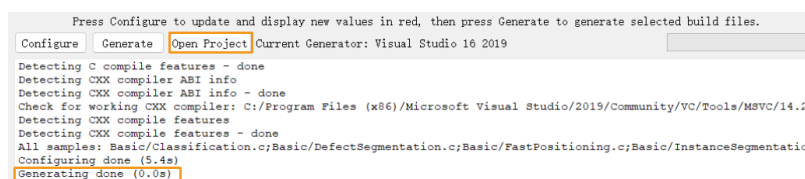
Then, you will enter the configuration window. Select the Visual Studio version and click [Finish].



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



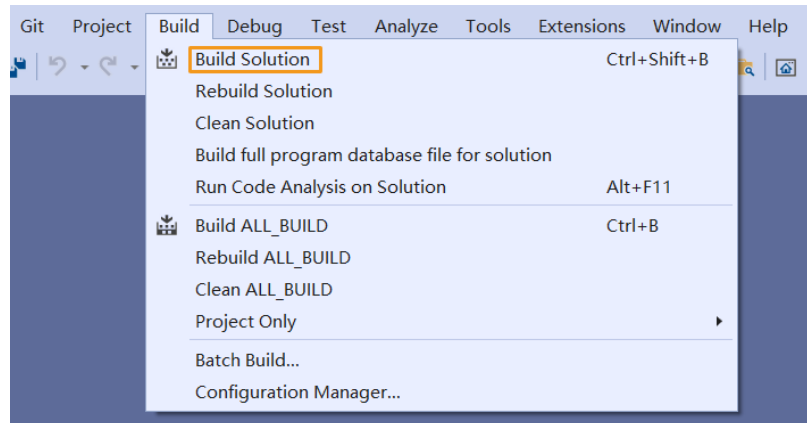
- Click [Generate] to generate Visual Studio solutions. When the generation completes, the statement **Generating done** will appear in the last line of the log. Then, click [Open Project] to open the solutions in Visual Studio.



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.



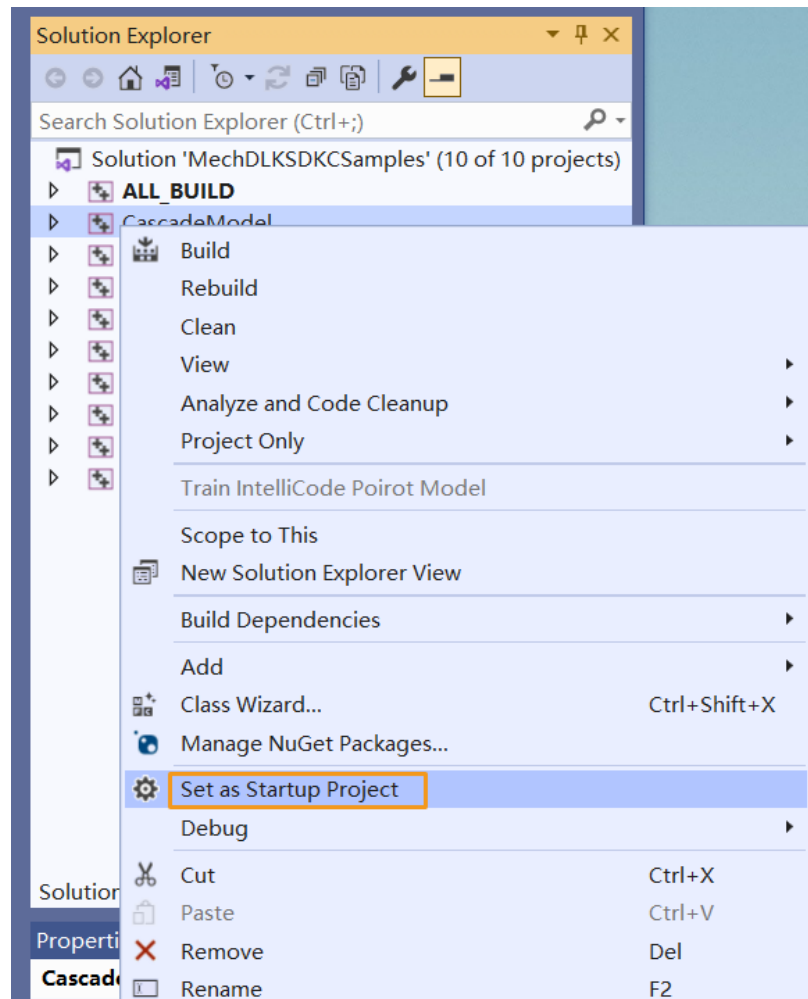
3. Copy and paste the **resources** folder to the following path:
 - `xxx\mechdlk_sdk\samples\c\build\Release`
4. Copy and paste all files in the **3rd_dll** folder under the project folder to the path `xxx\mechdlk_sdk\samples\c\build\Release`.
5. Copy and paste all files in the **mechdlk_sdk\dll** folder 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](#)

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 Questions

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