Mech-DLK SDK User Manual

v2.0.0

Table of Contents

1. Welcome

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. Currently, development in C language is supported.

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

Contents

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

No.	Chapter	Content
1	*Installation Guide*	View system requirements and environment that should be configured and get Mech-DLK SDK.
2	*Getting Started*	Learn to use Mech-DLK SDK for Defect Segmentation model inference.
3	Samples	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 language.
5	*FAQs*	View the frequently asked questions.

2. Installation Guide

System Requirements

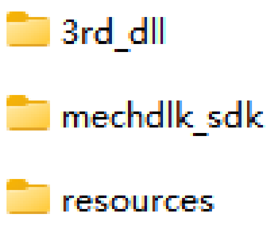
It is recommended that the device where model inference using Mech-DLK SDK is performed should meet the following requirements.

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 1650 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 Downloads.
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.

Configure Environment

Please ensure you have configured the environment required to directly run C samples in Mech-DLK SDK on your device. Confirm the requirements by the following:

1. Installed CMake and Visual Studio.
2. Added related environment variable.

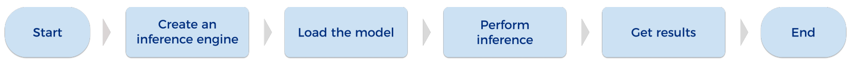
3. Getting Started

This chapter introduces how to apply Mech-DLK SDK to achieve inference of the Defect Segmentation model exported from Mech-DLK.

Prerequisites

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

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

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

Call the function `createImage` to create an input image.

Create an Inference Engine

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

Call the function `createPackInferEngine` to create an inference engine.

- 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 1 to 5 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

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

Call the function `infer` for deep learning engine inference.

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

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

Call the function `getDefectSegmentataionResult` to obtain the result of the defect segmentation model.

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

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

Call the function `resultVisualization` to plot the model inference result on the image(s) in `input`.

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

Release Memory

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

Release the memory of model inference results, input image(s), and the model engine to prevent memory leaks.

4. Samples

4.1. C (Windows)

In this chapter, you will learn to configure the samples of C APIs in Mech-DLK SDK with CMake and Visual Studio on a Windows operating system.

Sample List

We provide two categories of samples: **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 the required software.
- Add related environment variable.

Install Required Software

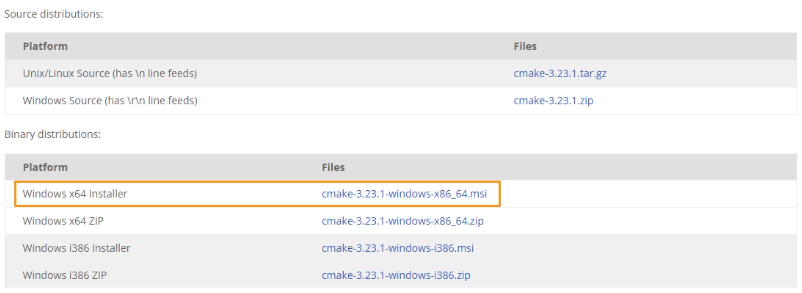
Before using the C samples in Mech-DLK SDK, you should install Mech-DLK SDK, CMake, and Visual Studio.

Install the Latest Version of Mech-DLK SDK

Please install the Mech-DLK SDK and the third-party libraries and resources it depend upon according to *Installation Guide*.

Install CMake (Version 3.2 or Above)

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



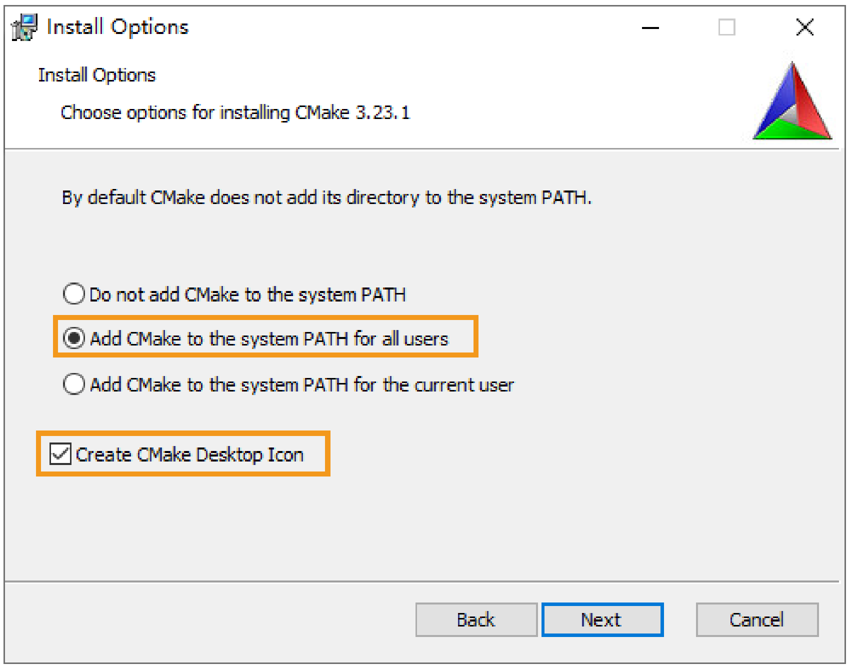
Source distributions:

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 Options

Install Options

Choose options for installing CMake 3.23.1

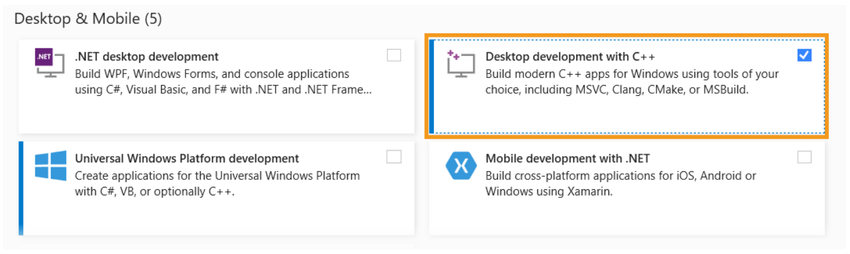
By default CMake does not add its directory to the system PATH.

- Do not add CMake to the system PATH
- [✓ Add CMake to the system PATH for all users]
- Add CMake to the system PATH for the current user

☑ Create CMake Desktop Icon

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.



Desktop & Mobile (5)

- .NET desktop development: Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...
- [✓ Desktop development with C++: Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.]
- Universal Windows Platform development: Create applications for the Universal Windows Platform with C#, VB, or optionally C++.
- Mobile development with .NET: Build cross-platform applications for iOS, Android or Windows using Xamarin.

Add Environment Variable

You can add the related environment variable 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, scroll to **Path** and double-click it to show the **Edit System Variable** dialog box.
4. Click **[New]** in the upper-right corner, add the path to Mech-DLK SDK, i.e., *xxx\mechdlk_sdk*, and then click **[Move Up]** to move the variable to the top. After that, click **[OK]**.

It is recommended to restart your computer after the installation; otherwise, the environment variable added may not take effect.

Build and Run Samples

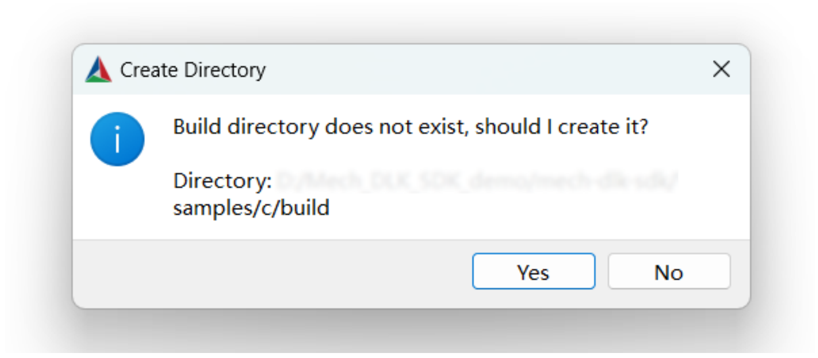
You can build all samples at a time.

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

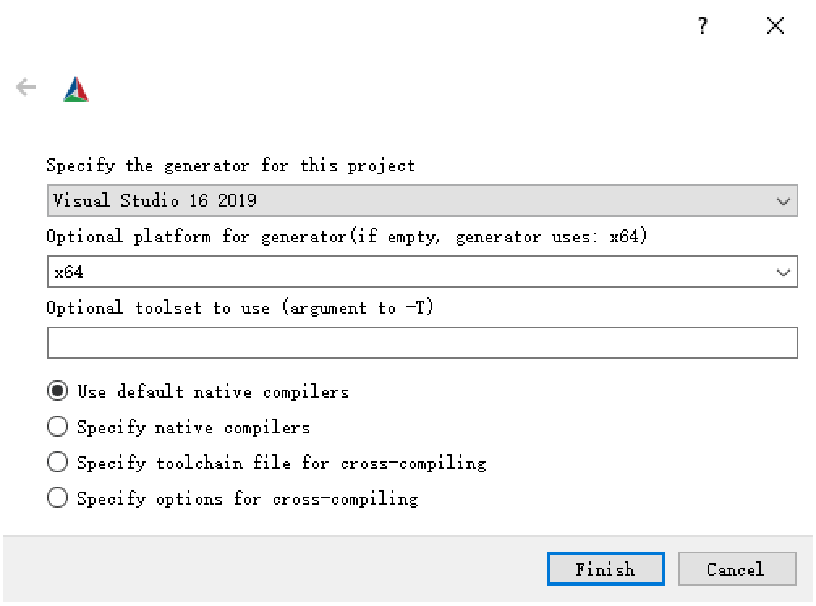
3. Click **[Configure]**. When you first click the **Configure** button, a window will pop up to confirm whether you want to create the **build** folder. Click **[Yes]** to create the folder.


Create Directory

Build directory does not exist, should I create it?

Directory: [redacted] samples/c/build

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



Specify the generator for this project

Visual Studio 16 2019

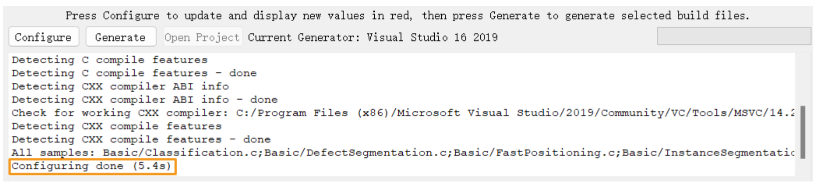
Optional platform for generator(if empty, generator uses: x64)

x64

Optional toolset to use (argument to -T)

● Use default native compilers
○ Specify native compilers
○ Specify toolchain file for cross-compiling
○ Specify options for cross-compiling

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

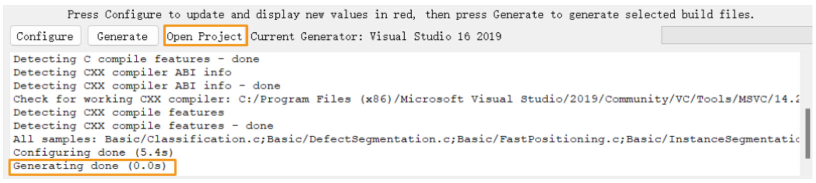


Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 16 2019

```
Detecting C compile features
Detecting C compile features - done
Detecting CXX compiler ABI info
Detecting CXX compiler ABI info - done
Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/Tools/MSVC/14.2
Detecting CXX compile features
Detecting CXX compile features - done
All samples: Basic/Classification.c;Basic/DefectSegmentation.c;Basic/FastPositioning.c;Basic/InstanceSegmentatic
Configuring done (5.4s)
```

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.



Press Configure to update and display new values in red, then press Generate to generate selected build files.

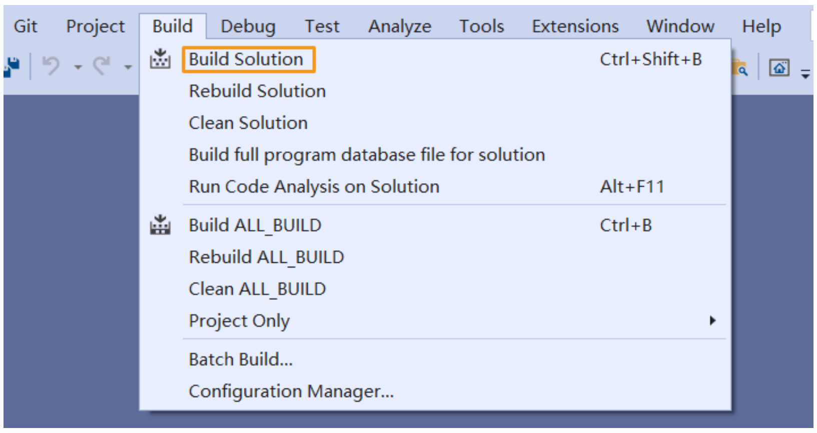
Current Generator: Visual Studio 16 2019

```
Detecting C compile features - done
Detecting CXX compiler ABI info
Detecting CXX compiler ABI info - done
Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/Tools/MSVC/14.2
Detecting CXX compile features
Detecting CXX compile features - done
All samples: Basic/Classification.c;Basic/DefectSegmentation.c;Basic/FastPositioning.c;Basic/InstanceSegmentatic
Configuring done (5.4s)
Generating done (0.0s)
```

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. Click Build › Build Solution. An executable file in the EXE format 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.



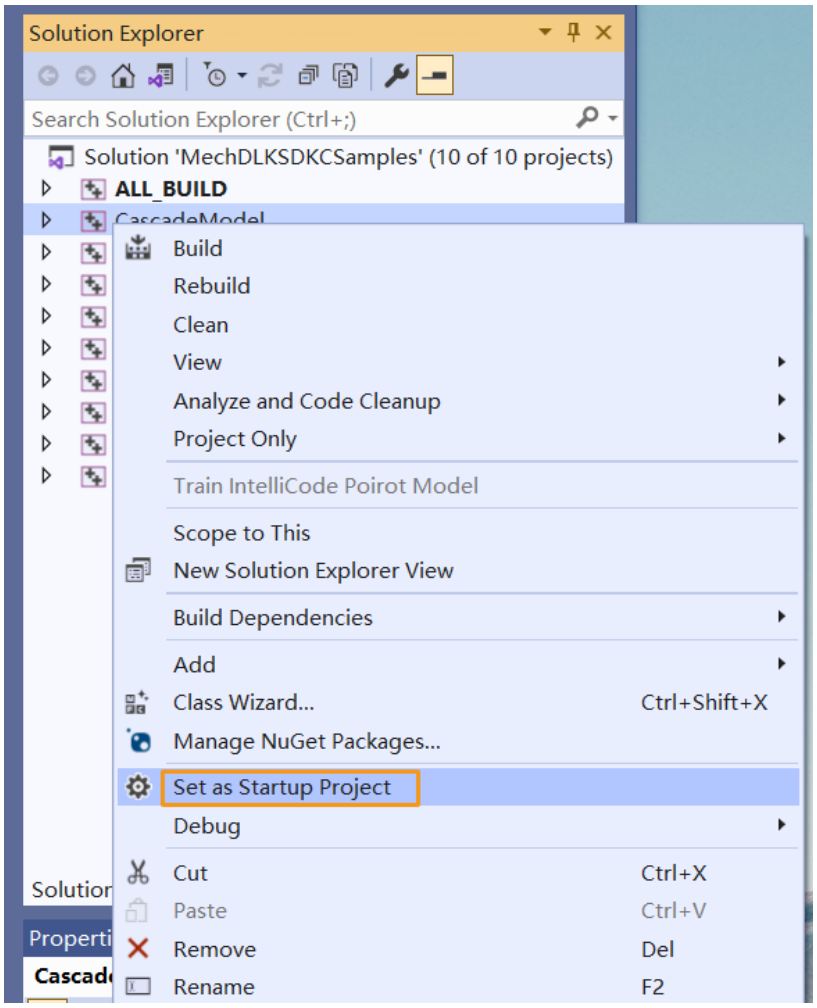
3. Copy and paste the **resources** folder to the following path:
   - *xxx\mechdlk_sdk\samples\c\build\Release*
4. Copy all files in the **3rd_dll** folder under the project folder and all files in the path *xxx/mechdlk_sdk/dll* and paste them to the path *xxx\mechdlk_sdk\samples\c\build\Release*.

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 EXE file named after the sample to run the sample and obtain results.

5. API Reference

Please click Mech-DLK SDK C API to view related C APIs.

6. FAQs

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.

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

It has been verified that LabVIEW can be integrated with Mech-DLK SDK by loading C DLLs and thus call functions or programs. 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?

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