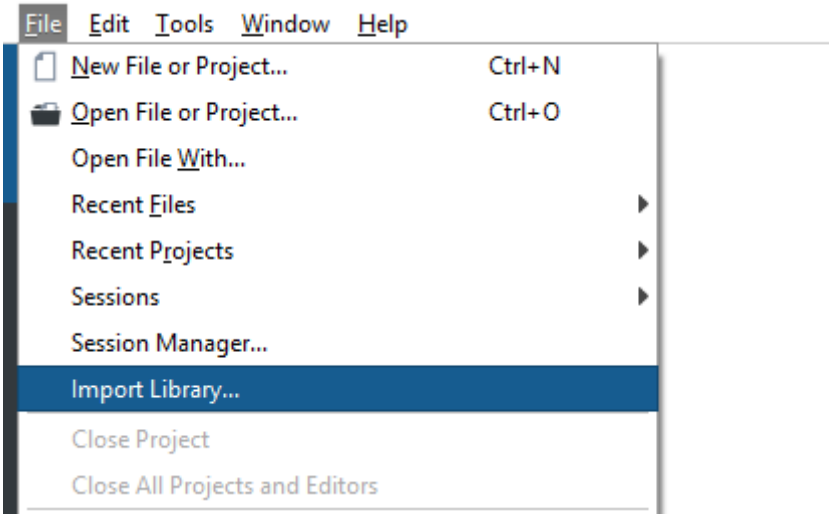
Using OpenCV with CDP Studio

CDP Studio supports importing and using third-party C++ libraries, like OpenCV.

Importing OpenCV

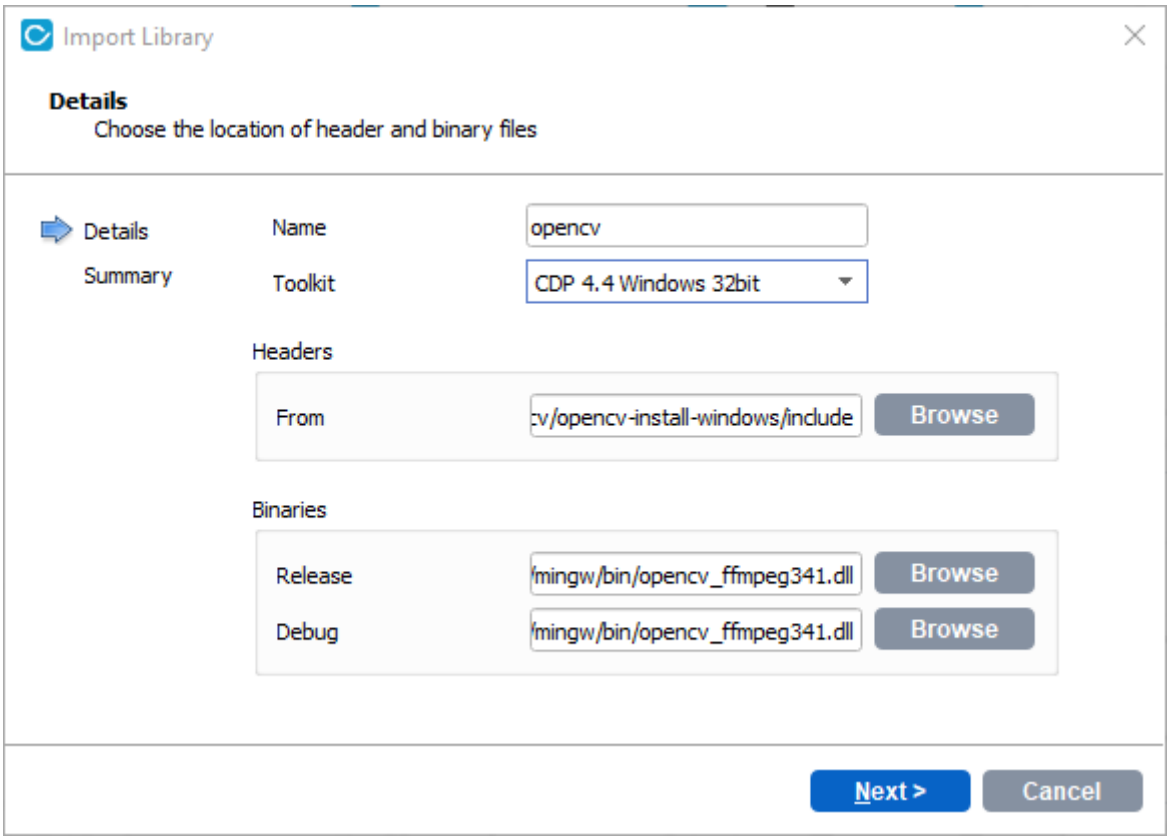
Select ***File -> Import Library…***



Next a dialog appears where one must specify headers directory and binaries. OpenCV header files are in *include* directory.

- For Windows release binaries are in *opencv-windows-release/x86/mingw/**bin*** folder and debug versions in *opencv-windows-debug/x86/mingw/**bin***. Select and add all *.dll* files.
- For other targets binaries are in ***lib*** directory. Select all *.so* files and add **same** files for both debug and release versions.

Note: make sure to **select a toolkit** that matches the CDP version of your project.

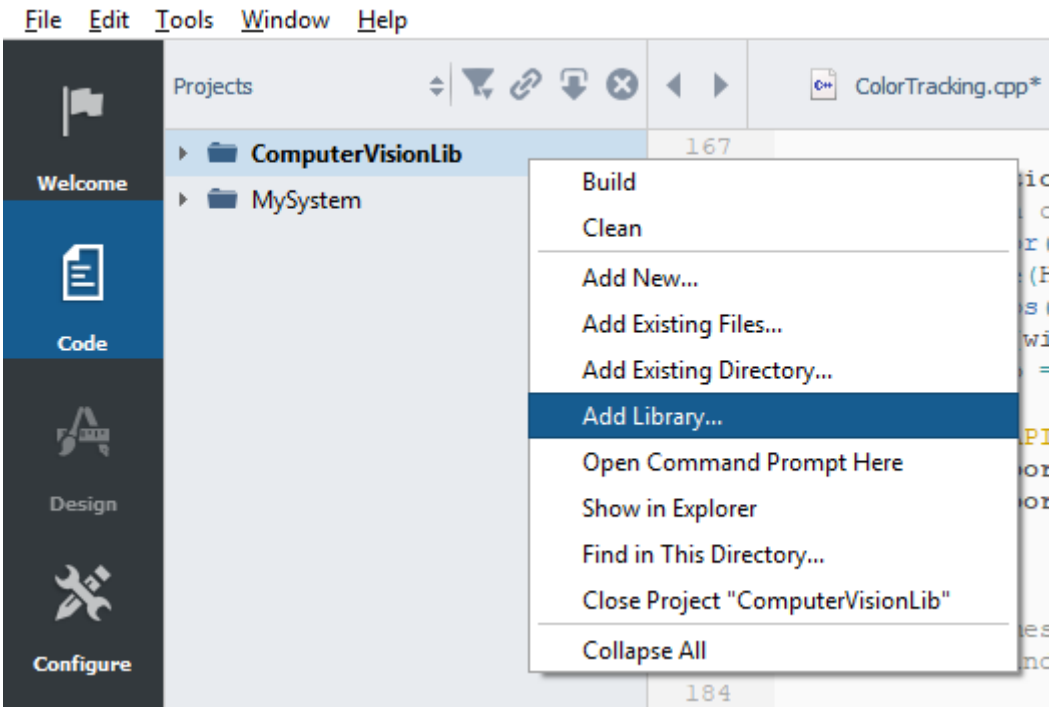


Note: ***Import Library…*** dialog must be run for both host and target platforms. For example, when developing a CDP system using OpenCV on Windows and deploying it on Raspberry Pi, the dialog must be run once for Windows binaries and once for Raspbian binaries. Make sure to set same *Name* each time (e.g. "**opencv**").

Using OpenCV in CDP library

Adding dependencies

To use OpenCV in your CDP application one should first create a CDP library project and add a CDP Component to it. Next one must add OpenCV dependency to this library project. For that go to Code Mode, right click on the library project and select ***Add Library…***



In the dialog select an ***External library*** and from the list the name you entered when running *Import Library…* wizard, e.g. "***opencv***".

Next open again the **Add Library…** dialog, select **CDP library** and from the list **CDP2QtLib**.

Code and multi-threading

The UI part of OpenCV must always be accessed from main thread that runs the UI event loop. Do not create windows from CDP thread (like the Process() function in CDP Component). One easy way to execute code on UI thread is to use Application::RunInMainThread() function that takes lambda as an argument.

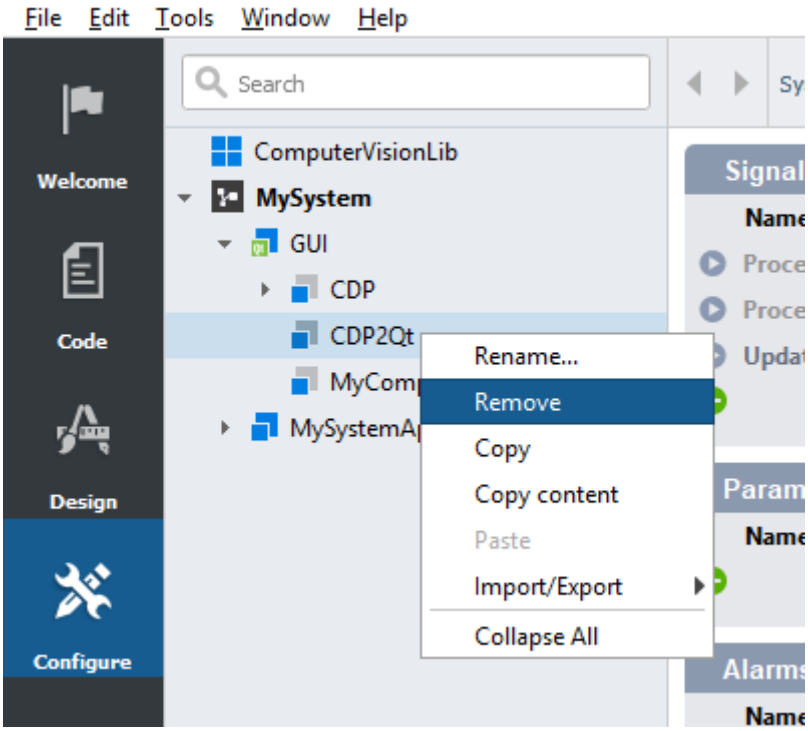
Also note that to access objects living in CDP Component thread (like CDPProperty, CDPSignal, CDPParameter) from UI thread, one should first lock component mutex or use RunInComponentThread() function.

```cpp
#include <opencv2/opencv.hpp>
#include <opencv2/highgui.hpp>
#include <CDPSystem/Application/Application.h>

void MyComponent::Activate()
{
  Application::RunInMainThread([&] { Loop(); });
}

void MyComponent::Loop()
{
  m_capture.read(m_currentFrame); // Read camera image

  {
    OSAPIMutexLocker(GetMemberAccessMutex()); // Locks component mutex
    for (auto filter : m_filters) // m_filters lives in component thread
      filter->applyFilter(m_currentFrame);
  }

  cv::imshow(windowName, cameraFeed);
  cv::waitKey(30);
  Application::RunInMainThread([&] { Loop(); });
}
```

Adding Component to CDP Application

The pre-built OpenCV binaries bundled with this guide have Qt dependency. Therefore, they work correctly only when dragged into a CDP GUI Application, which always bundles Qt. After creating a System project, go to Configure Mode, right click on the System and select ***Add GUI Application…***

After that, drag your CDP Component into from Resource pane into your CDP GUI Application. By default, CDP GUI Application includes an empty form called *mainwidget* for creating your own UI. If you have no need for it, remove the CDP2Qt subcomponent.



Example

An example project using OpenCV is available at https://github.com/CDPTechnologies/OpenCV-ColorTracking. Make sure to also read the project wiki.