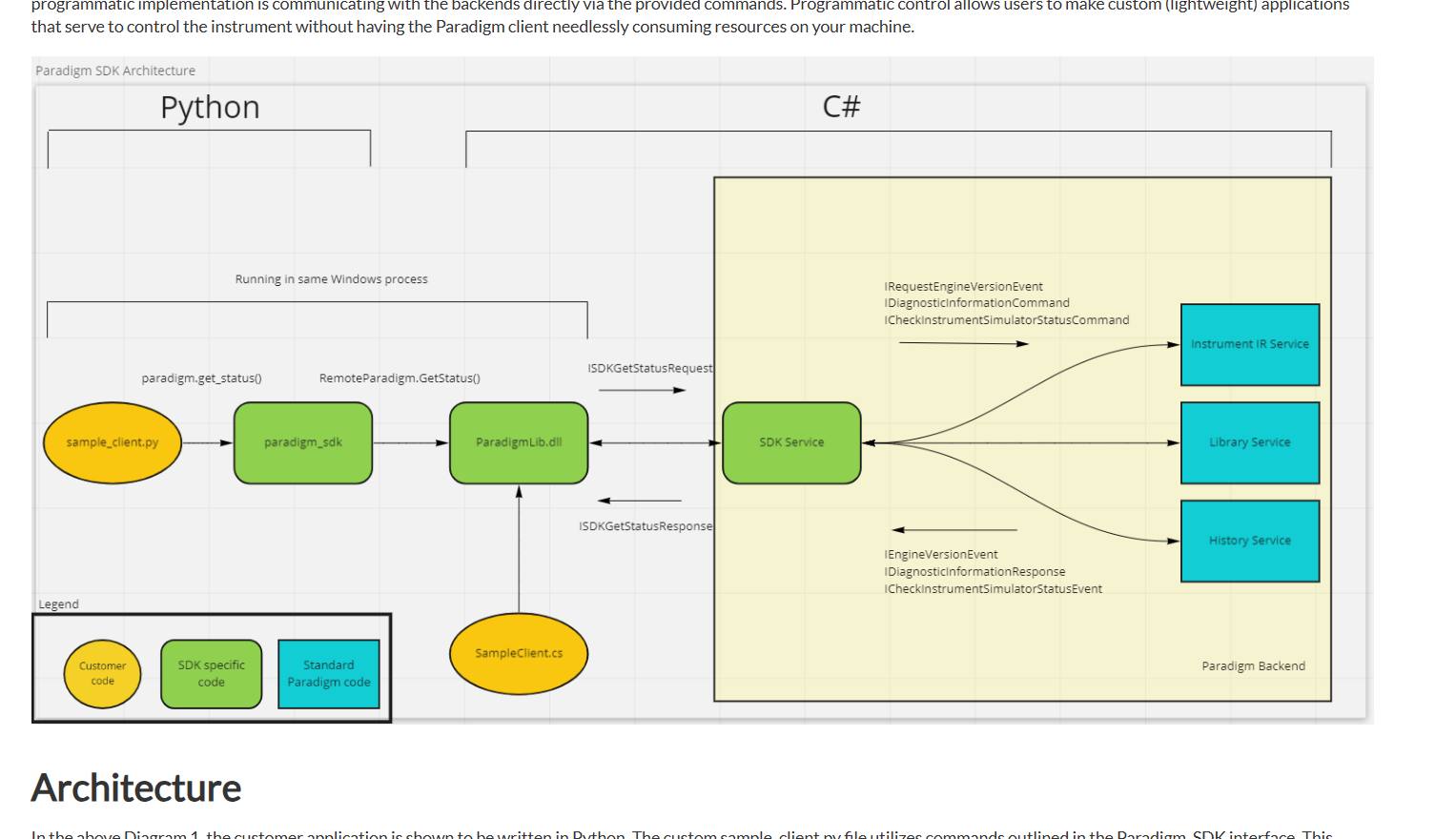
**Reflectometer**

**20 Jan, 2025**

1. [OMNIC Paradigm Software - Thermo Fisher Scientific](https://knowledge1.thermofisher.com/Molecular_Spectroscopy/Molecular_Spectroscopy_Software/OMNIC_Family/OMNIC_Paradigm_Software)
2. 
3. Need to create a wrapper that can call C# DLL functions
4. USB Connection
5. Here ParadigmLib.dll is the key
6. **How can we use the C# dll to our MFC app?**
7. **Way-01:** **C++/CLI**: A .NET extension for C++ that allows you to directly interact with .NET assemblies.
8. **Way-02:** **P/Invoke (Platform Invocation Services)**: Use a wrapper in C# to expose necessary methods, and call those from your C++ application.

If ParadigmLib exposes a COM (Component Object Model) interface, you can use C++ to directly interact with it.

1. **Way-03:** If working directly with ParadigmLib.dll is too complex, consider writing a lightweight C# application that acts as a mediator between your C++ code and the Paradigm backend. This mediator would handle SDK calls and communicate with C++ application via IPC.
2. First try to directly call the APIs
3. How to build C# dll?
4. How to use that dll in C++?
5. SECS/GEM documentation: [SECS/GEM Software Integration Tools for .NET, Java, VC++ and Linux or Windows (hume.com)](http://hume.com/)
6. [Consuming Your C# Library in MFC/C++ Project- CodeProject](https://www.codeproject.com/Articles/5358626/Consuming-Your-Csharp-Library-in-MFC-Cplusplus-Pro)
7. C++/CLI: [C++/CLI - Wikipedia](https://en.wikipedia.org/wiki/C%2B%2B/CLI)

**23 Jan, 2025**

1. Created a DLL file in C# and added to MFC C++, it is working
2. Copy the dll file to x64 folder, where .sln is present
3. Copy the dll to source folder where all .h and .cpp files are present
4. Add the dll to the project
5. **Now, Check the wrapper dll with MFC,** Not working!! May be issue with 02 dll linking

**27 Jan, 2025**

1. [Exposing native to managed - C++/CLI vs. P/Invoke- CodeProject](https://www.codeproject.com/Articles/651516/Exposing-native-to-managed-Cplusplus-CLI-vs-P-Invo#createCsConsoleProject)
2. .NET (mean .NET core) is a cross platform library
3. .NET Framework is only for Windows development
4. **It worked. Done**

**Documentation**

This article shows a basic introduction for calling a .NET DLL to a MFC project using C++/CLI class library.

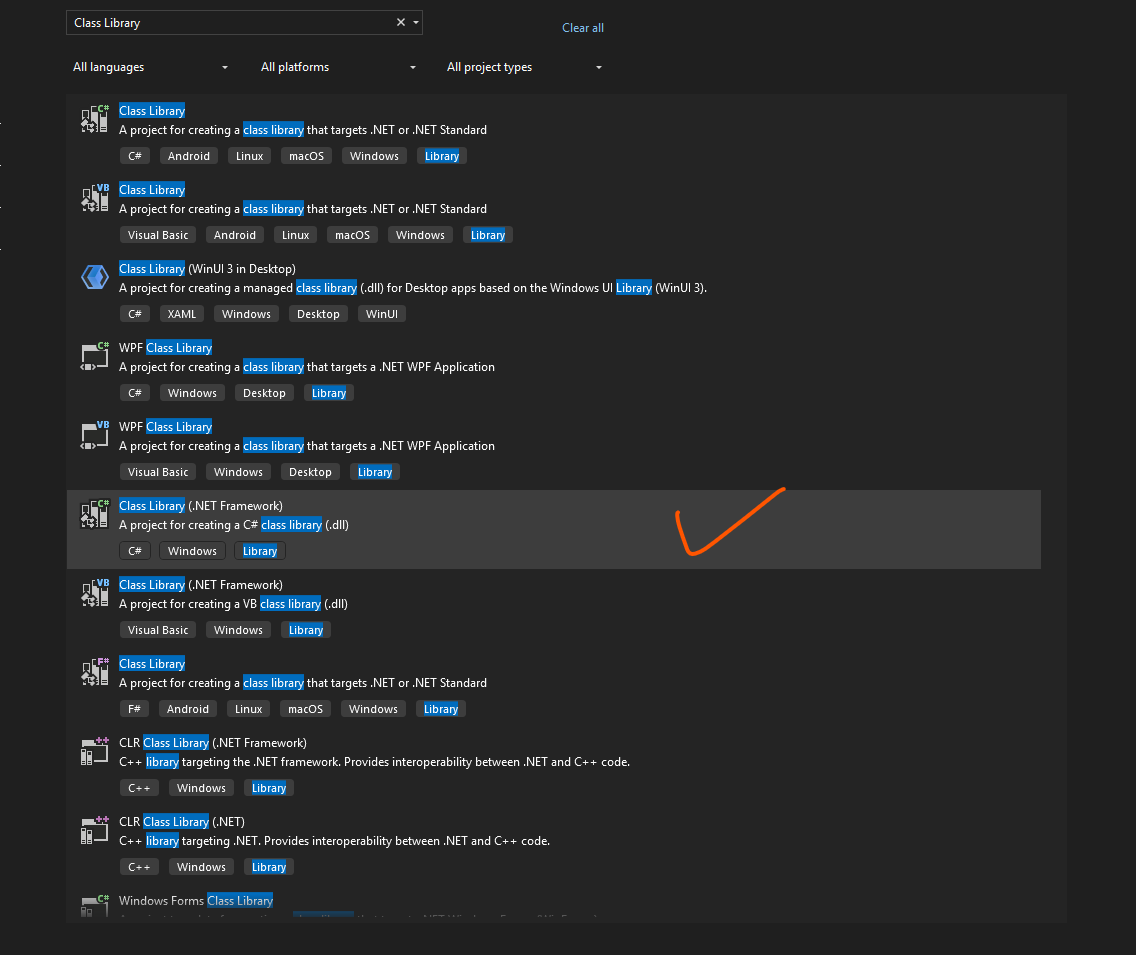
Summary of Phases:

1. Build a DLL through C# Class Library
2. Build another DLL through a **C++/CLI Class Library**
3. Integrate the DLL to MFC x86 and use its APIs

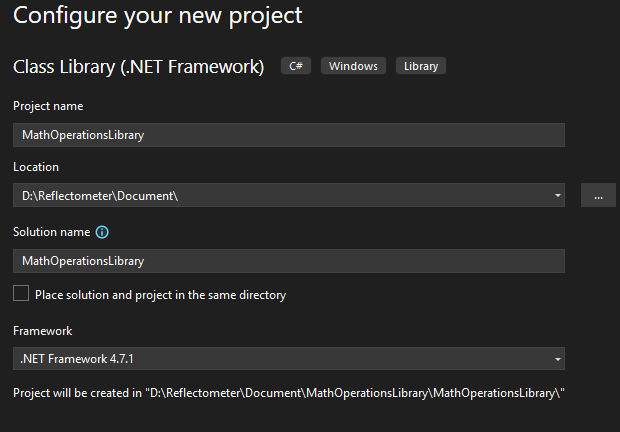
I’m describing it step by step below-

**Phase-01: Build a DLL through C# Class Library**

Step-1: Create the C# Class Library (.NET framework)



Step-2: Give name and select suitable version of framework\*



Step-03: Configure in x86

In top panel, Any CPU -> Configuration Manager -> Active solution platform -> New -> x86 -> Ok

Step-04: Rename the .cs file and class with MathOperations

Step-05: Put this code under the class in MathOperations.cs file-

public int Add(int a, int b)

{

return a + b;

}

public int Subtract(int a, int b)

{

return a - b;

}

Step-06: Build the solution

Step-07: A MathOperationsLibrary.dll file will generate inside D:\Reflectometer\Document\MathOperationsLibrary\MathOperationsLibrary\bin\x86\Debug

We will use this DLL in our wrapper

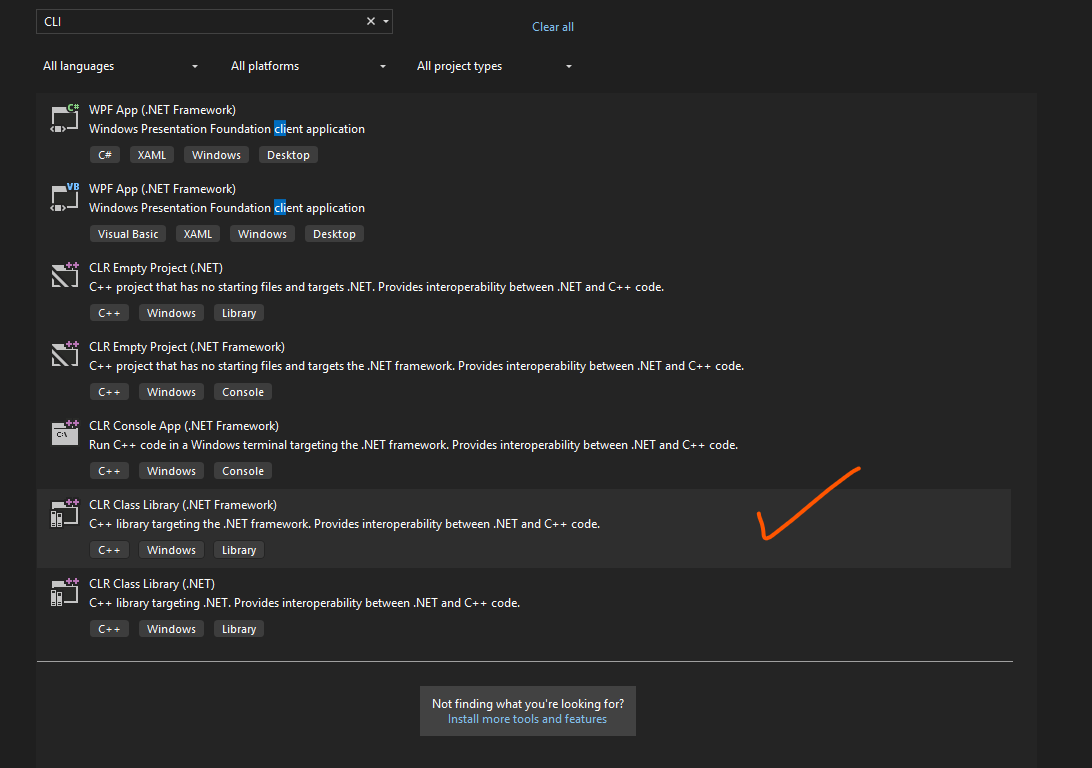
**Phase-02: Build another DLL through a C++/CLI Class Library (Wrapper DLL)**

**This works as a wrapper since we are assuming the first DLL can’t be directly called to the MFC app.**

**For this purpose, we need to use** C++/CLI Class Library. It provides interoperability between .NET and C++ code.

Step-01: Create a **C++/CLI Class Library** project (.NET Framework)

Remember to use same projects .NET or .NET Framework in both of the projects



Step-02: Rename the class as WrapperClass in WrapperDLL.h file

Step-03: In top panel, change to x86 Debug

Step-04: Under project, click References->Add Reference->Browse-> MathOperationsLibrary.dll->select and Ok

Step-05: Add these codes in WrapperDLL.h file

public:

int Add(int a, int b)

{

MathOperationsLibrary::MathOperations^ mathOps = gcnew MathOperationsLibrary::MathOperations();

return mathOps->Add(a, b);

}

int Sub(int a, int b)

{

MathOperationsLibrary::MathOperations^ mathOps = gcnew MathOperationsLibrary::MathOperations();

return mathOps->Subtract(a, b);

}

Step-06: A WrapperDLL.dll will come to D:\Reflectometer\Document\WrapperDLL\Debug

We will use this dll to our MFC App. This works as a bridge between C# and C++.

**Phase-03: Integrate the DLL to MFC x86 and use its APIs**

Step-01:Create an **MFC Dialog-Based Application** (MFCApp)

Step-02: Select x86 Debug

Step-03: Enable CLR Support.

Properties->Configuration Properties-> Advanced -> C++/CLI -> Common Language Runtime Support -> .NET Framework Runtime Support (/clr)

Enable Managed Incremental Build -> Yes

Apply

Step-04: Place **MathOperationsLibrary.dll** and **WrapperDLL.dll** in the same directory as the MFC application's executable.

Step-05: Under References->Add References -> Select 02 DLLs

Step-06: Add 02 buttons: Add and Sub

Step-07: Add this code in MFCAppDlg.h

#using <WrapperDLL.dll>

using namespace System;

Step-08: Add this in MFCAppDlg.cpp

#using <WrapperDLL.dll>

void CMFCAppDlg::OnBnClickedAdd()

{

// TODO: Add your control notification handler code here

WrapperDLL::WrapperClass^ wrapper = gcnew WrapperDLL::WrapperClass();

int addResult = wrapper->Add(10, 20);

AfxMessageBox(L"Add Complete");

}

void CMFCAppDlg::OnBnClickedSub()

{

// TODO: Add your control notification handler code here

WrapperDLL::WrapperClass^ wrapper1 = gcnew WrapperDLL::WrapperClass();

int subResult = wrapper1->Sub(30, 10);

AfxMessageBox(L"Sub Complete");

}

Step-07: Build the solution.

Step-08: Now click the Add and Sub button to see the result. Yess!!! All Done.

Now, you can use any C# DLL or API with your MFC App.

[C#: Hardware Integration and System-Level Coding-Free C# Hardware Coding (yeschat.ai)](https://www.yeschat.ai/gpts-9t557DXZ1Aj-C-Hardware-Integration-and-System-Level-Coding)

[C# Wrapper | Basler (baslerweb.com)](https://docs.baslerweb.com/frame-grabbers/c-wrapper)

[Programmer's Guide | Basler Product Documentation (baslerweb.com)](https://docs.baslerweb.com/pylonapi/net/Guide)

Books code: [PacktPublishing/Systems-Programming-with-C-Sharp-and-.NET: , published by Packt (github.com)](https://github.com/PacktPublishing/Systems-Programming-with-C-sharp-and-.NET)