



Solid Edge ST7 AddIn in C++



Jason Newell

23 Nov 2014 [CPOL](#)

Solid Edge ST7 AddIn in C++

This article was marked as deleted at 10 Mar 2016.

This article appears in the Third Party Products and Tools section. Articles in this section are for the members only and must not be used to promote or advertise products in any way, shape or form. Please report any spam or advertising.

[Download Solid_Edge_ST7_AddIn_CPP.zip - 27.5 KB](#)

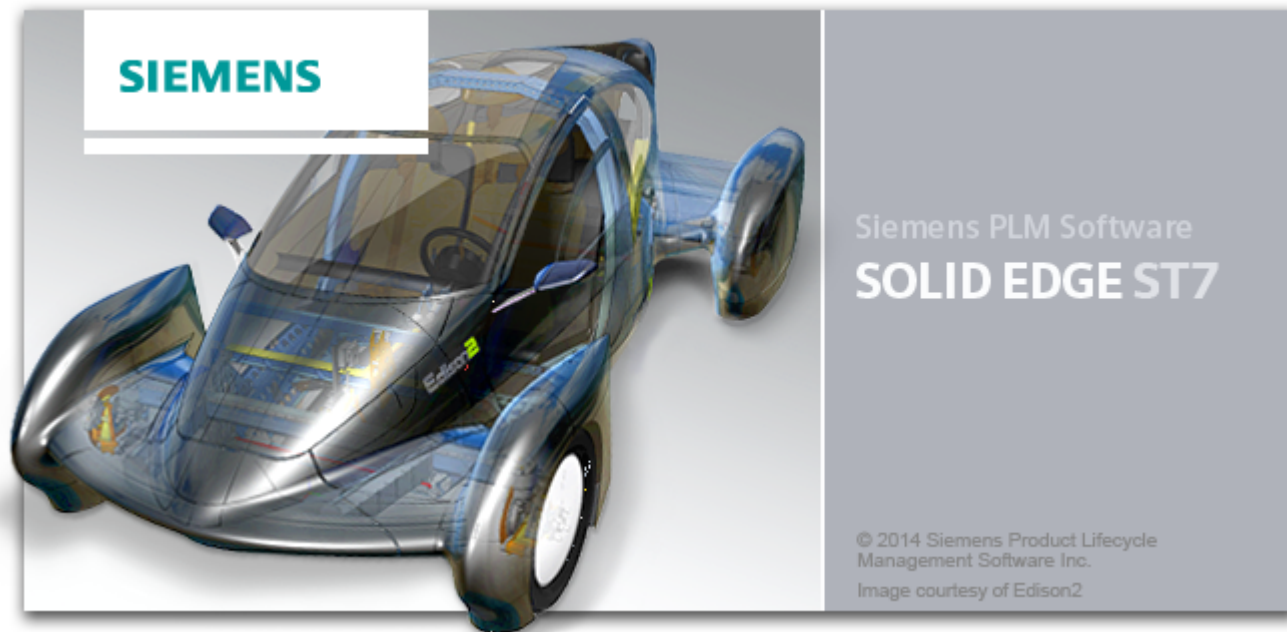


Table of content

- Introduction
- Background
- Creating the project
- Configuring the project
 - Add ATL Support To MFC
 - Configuration Manager
 - Project Properties
- Modifying stdafx.h
- Creating the addin class
- Configuring the addin class
 - Create a new GUID
 - Modify MyAddIn.h
 - Modify MyAddIn.cpp
 - Modify MyAddIn.rgs
 - Modify AddInDemo.idl
- Registration code
 - Create AddInRegistrar.h
 - Modify AddInDemo.cpp
- Build, Register and Debug
- Conclusion
- Resources

Introduction

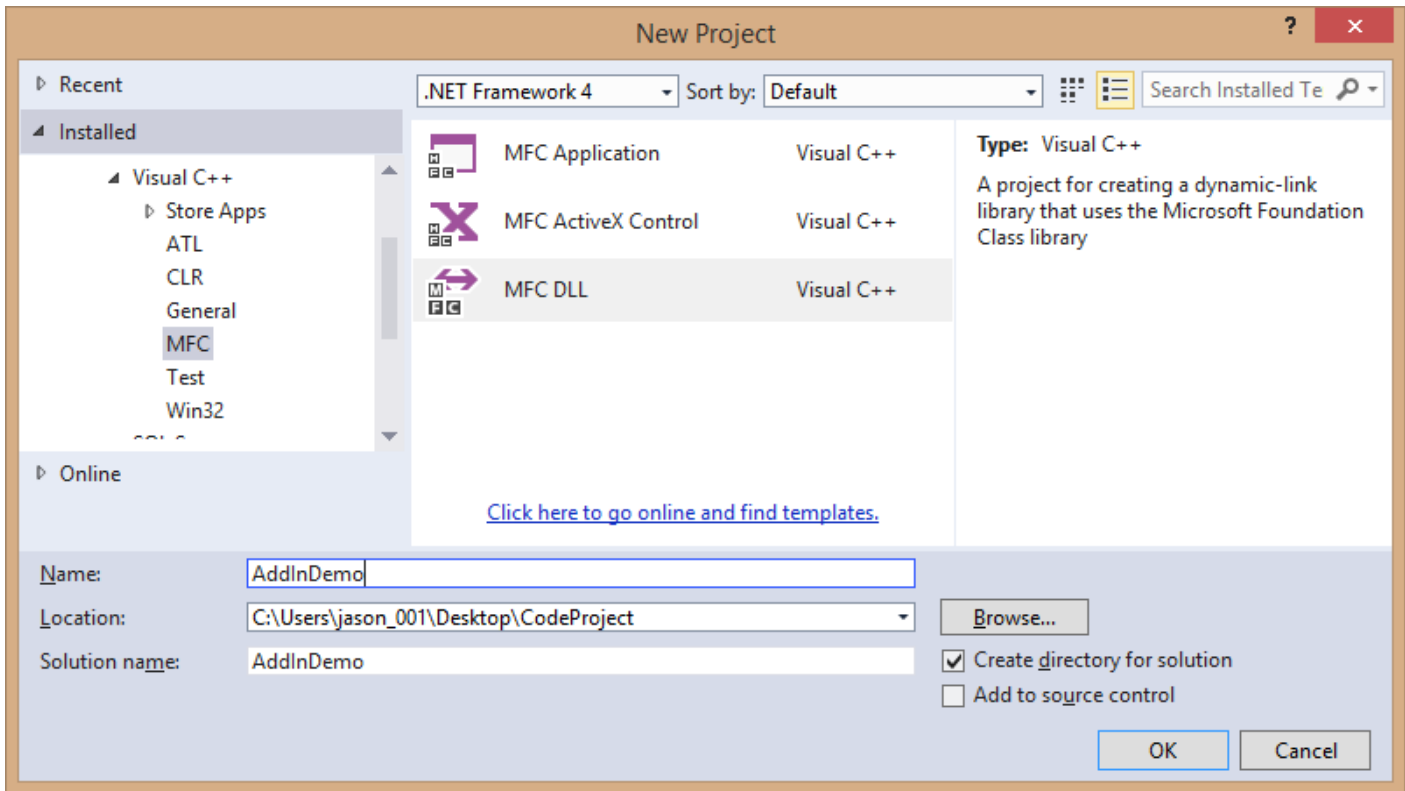
Solid Edge is a 3D CAD application. It is not uncommon for customers or partners to have a need to develop custom code that executes directly inside the Solid Edge process. This article is a continuation of the [Solid Edge ST7 AddIn Architecture Overview](#) article and focuses on the bare minimum requirements for implementing a Solid Edge AddIn using C++.

Background

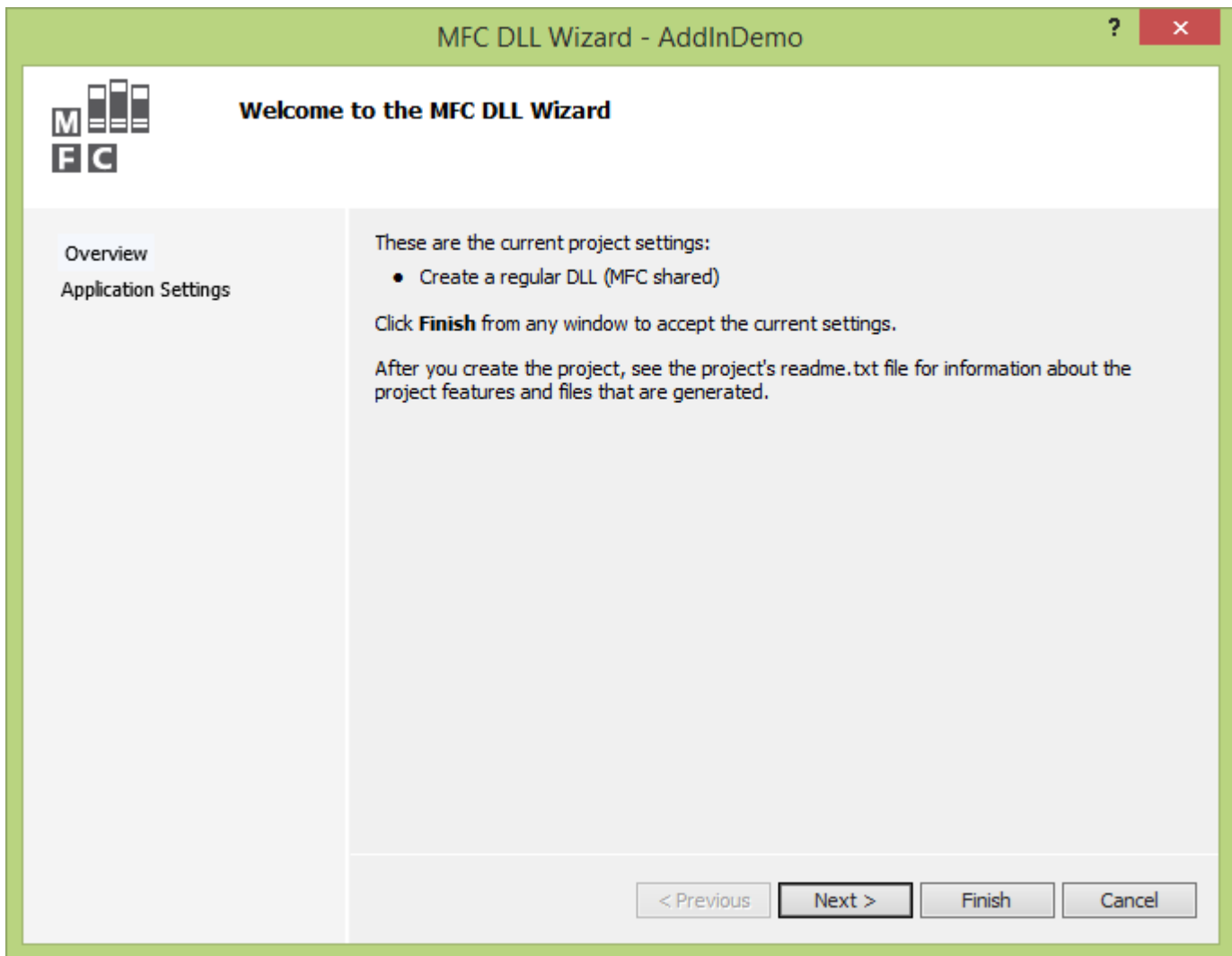
Windows 8.1 x64, Solid Edge ST7 x64 and Visual Studio 2013 Professional was used in creating this article but the steps are mostly the same if you're using different versions. I would also like to point out that there is more than one way to write a Solid Edge AddIn. In this article, I am simply showing one approach.

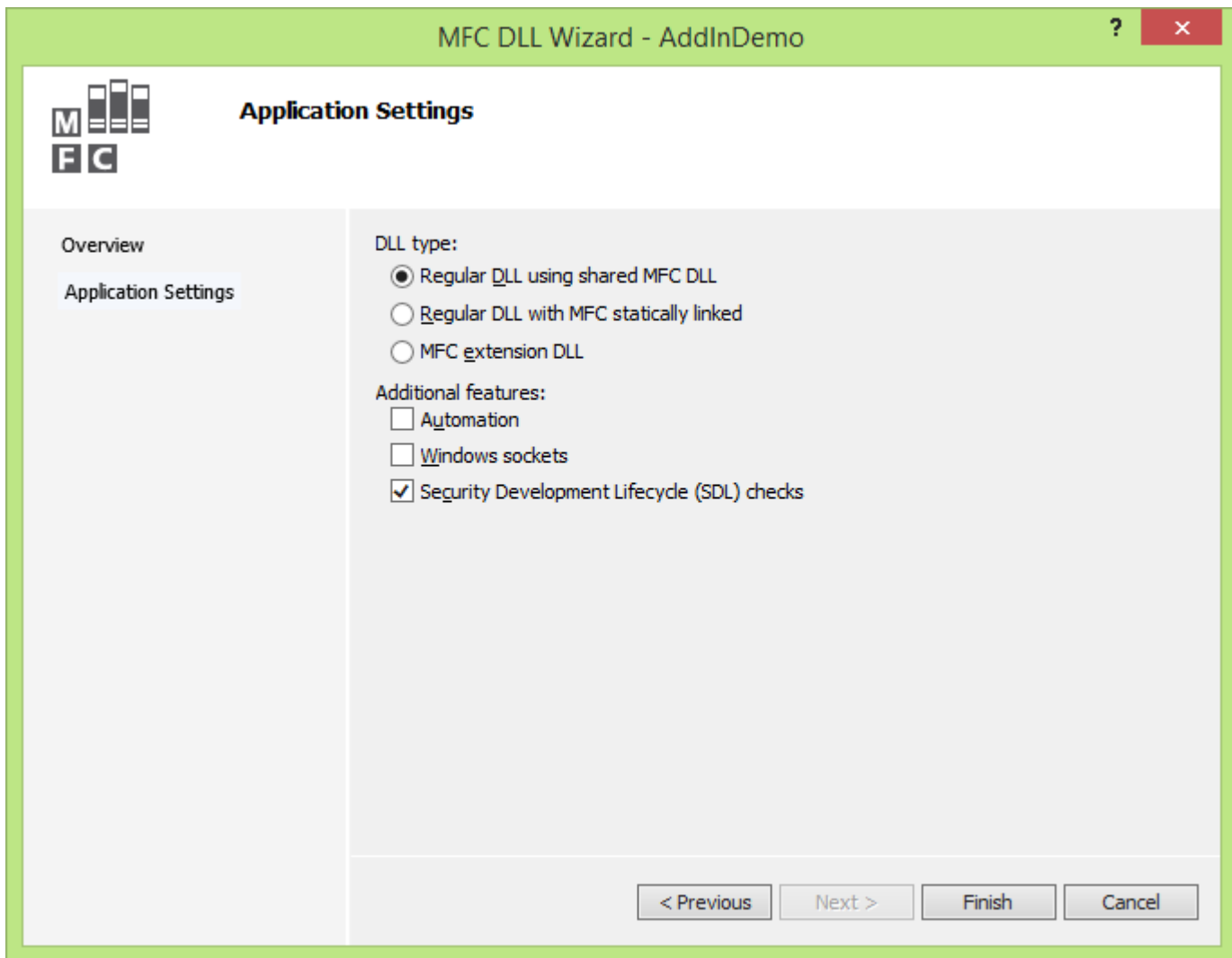
Creating the project

To get started, create a new **MFC DLL** project. You can certainly choose to create an **ATL Project** to implement your addin but generally speaking, you'll want to use MFC as it makes working with windows much easier.



You can accept all of the defaults in the MFC DLL Wizard.



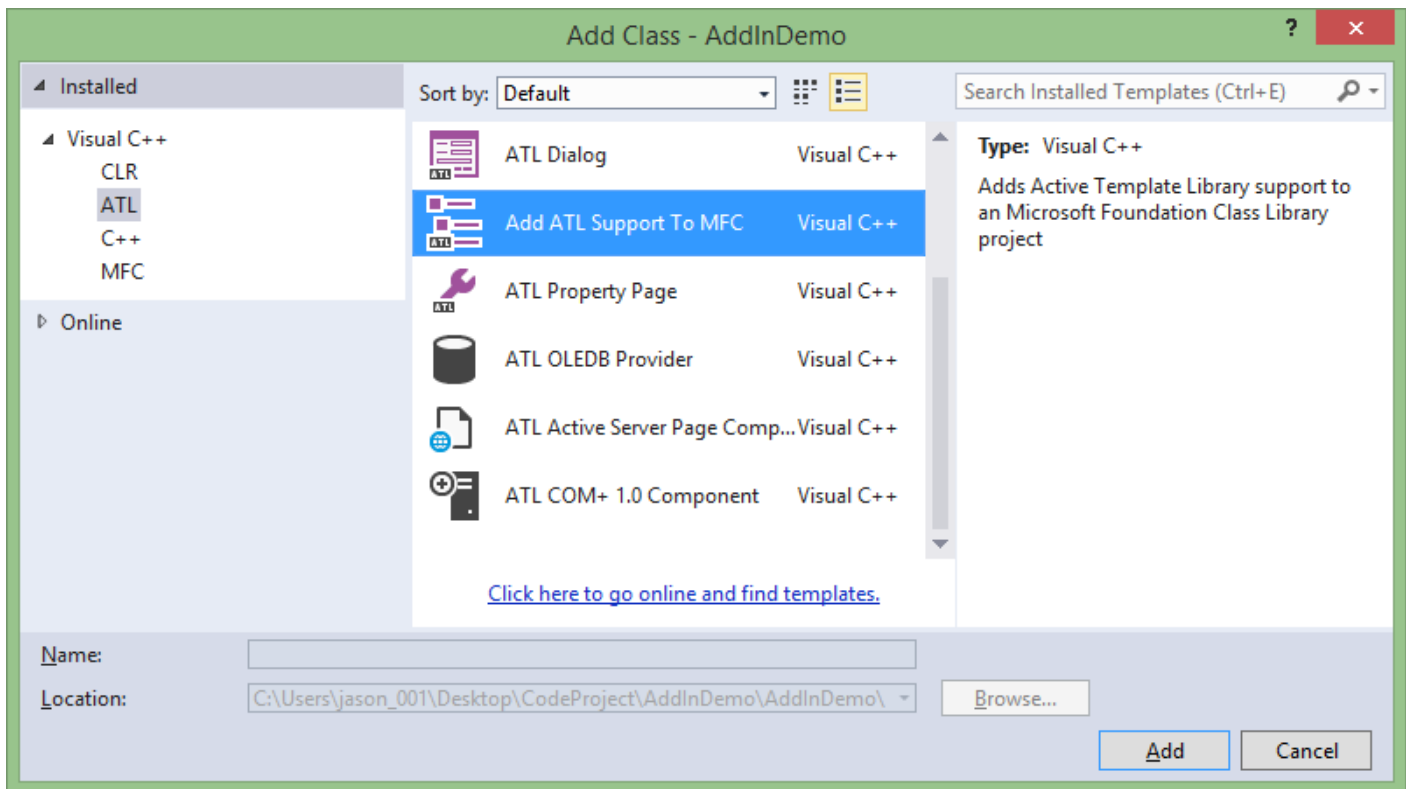
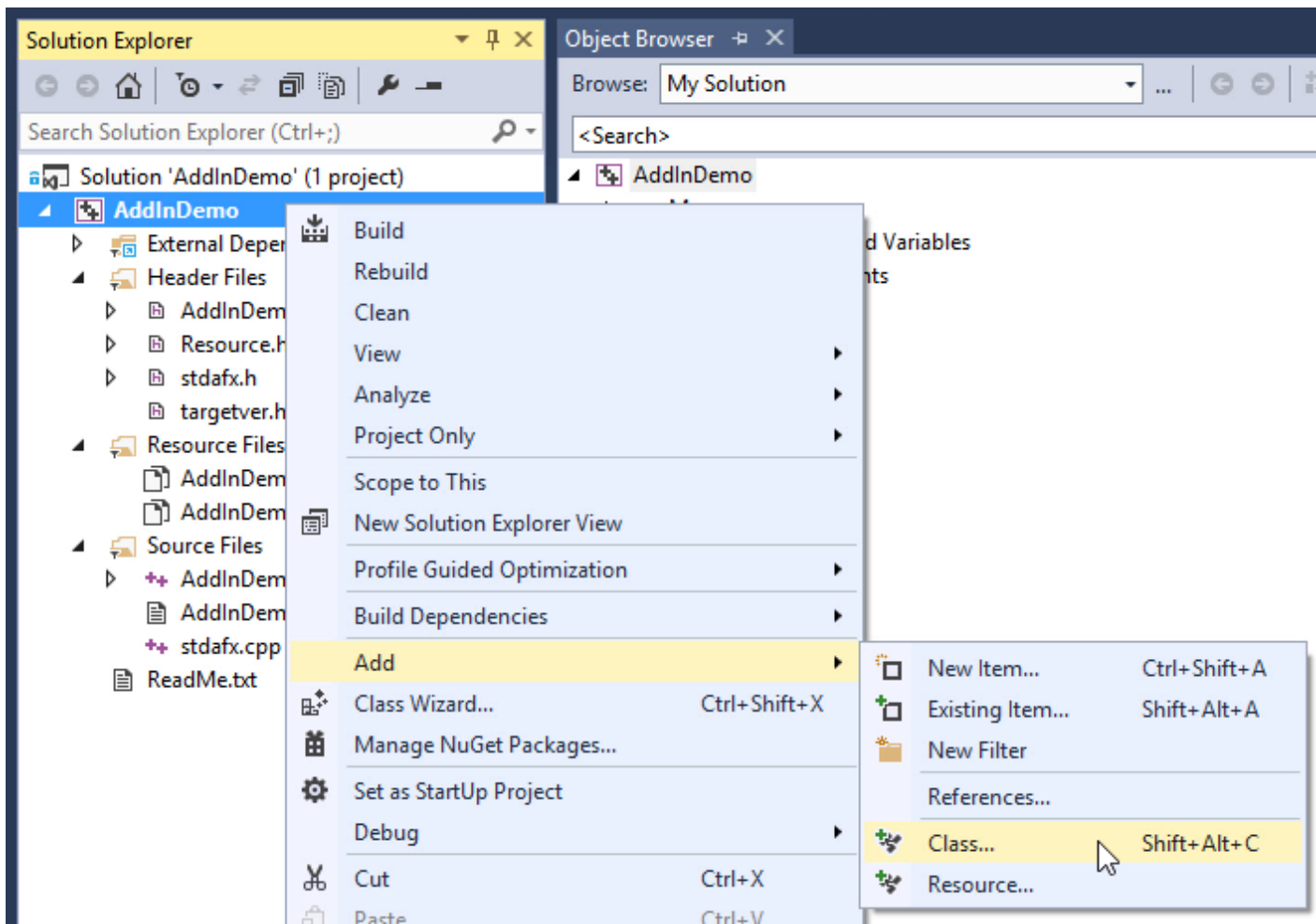


Configuring the project

Before starting to write code, there are a couple of things you'll want to change in your project.

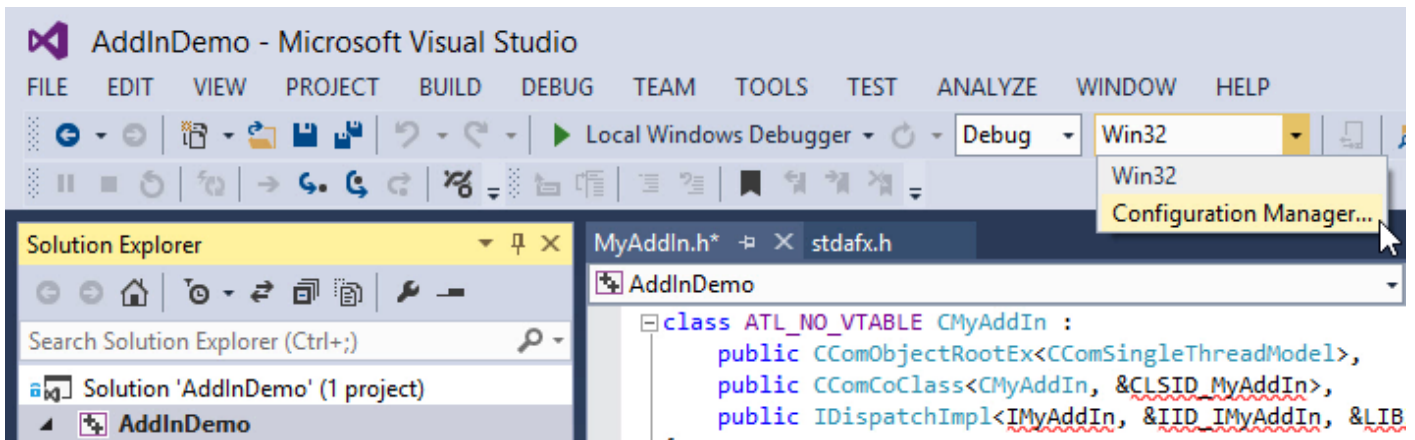
Add ATL Support To MFC

The [Active Template Library \(ATL\)](#) makes working with COM a lot easier so you'll want to add support for it to your project.

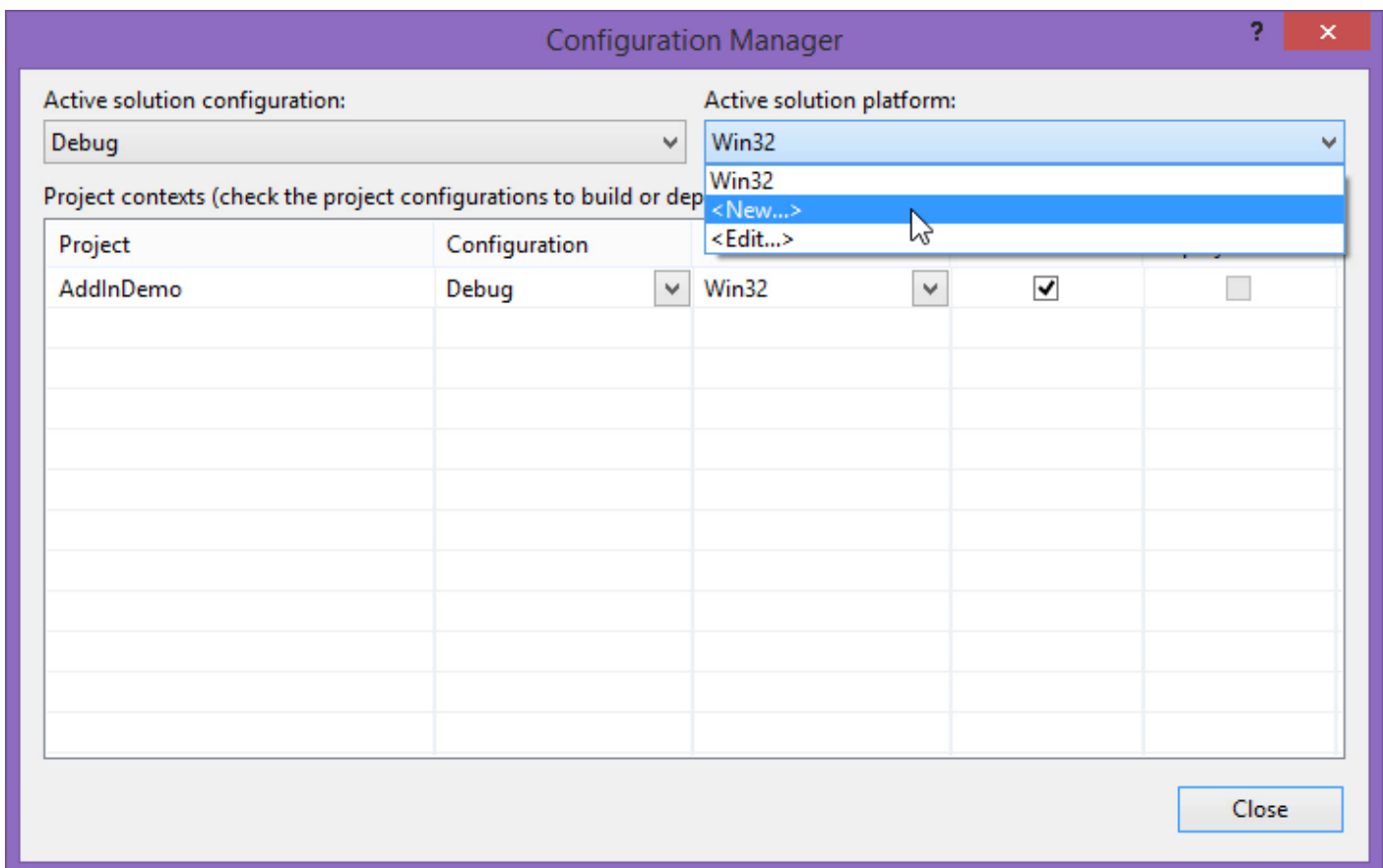


Configuration Manager

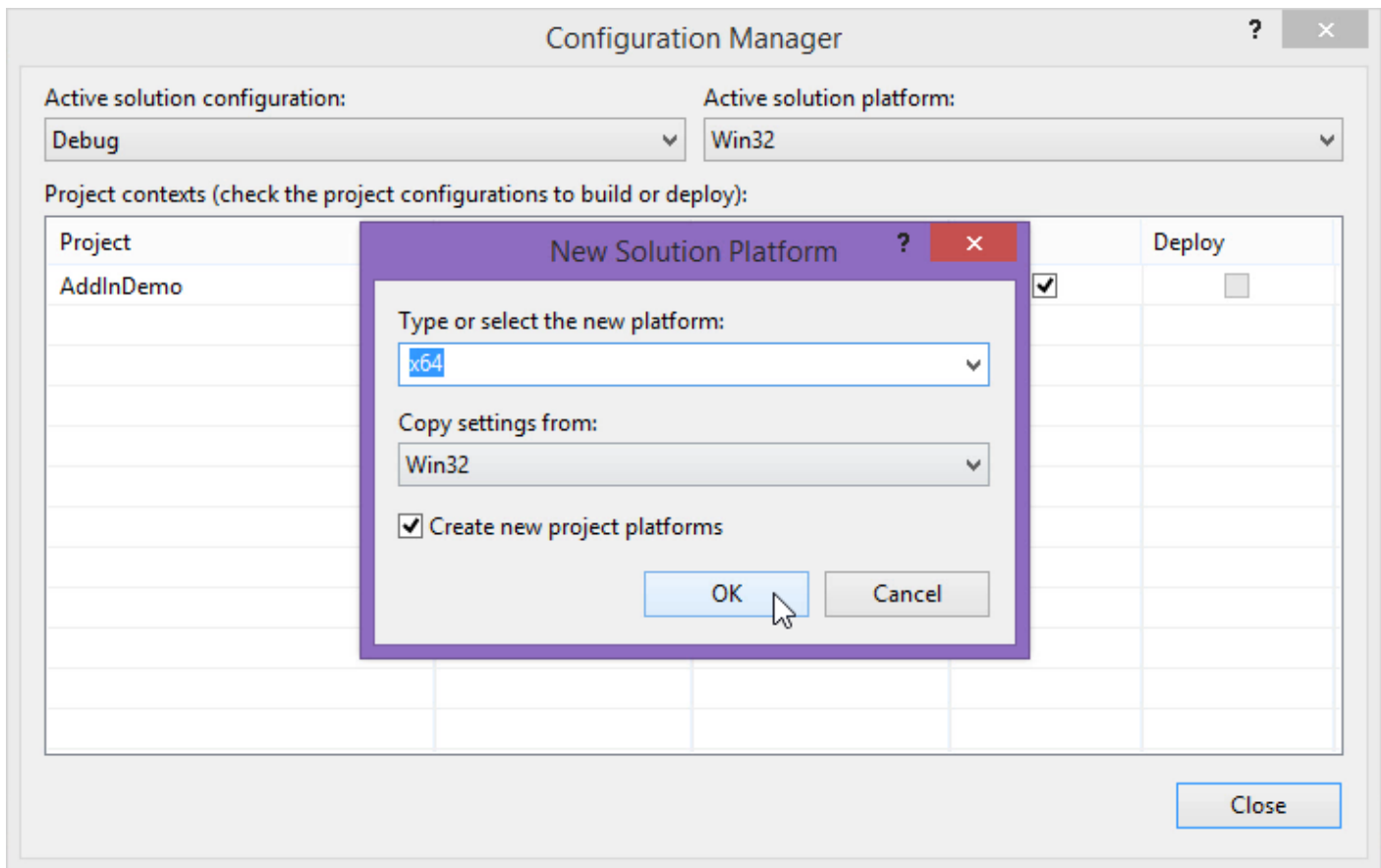
After the project is initially created, it will default to the **Win32** platform. Since we are targeting Solid Edge ST7 x64, we will need to create a new **x64** configuration. This can be accomplished using the Configuration Manager as shown below.



Once the Configuration Manager dialog is open, click on the **Active solution platforms** dropdown and select **<New...>**.

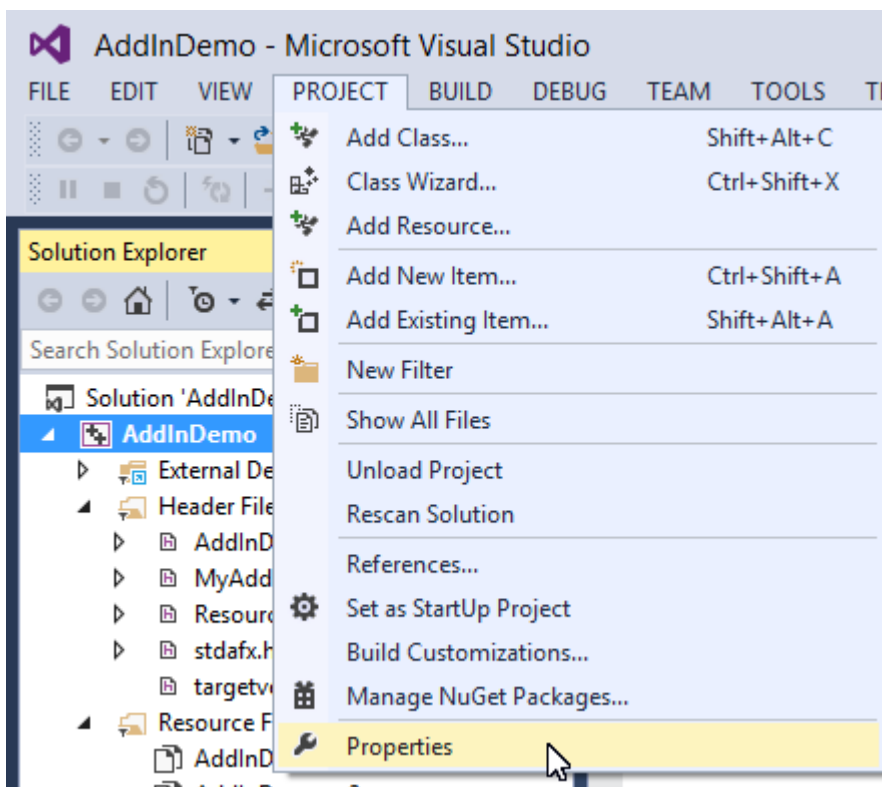


From the **Type or select the new platform** dropdown, select **x64** and click OK. This will copy the **Win32** platform settings to the **x64** settings and set **x64** as our target platform.



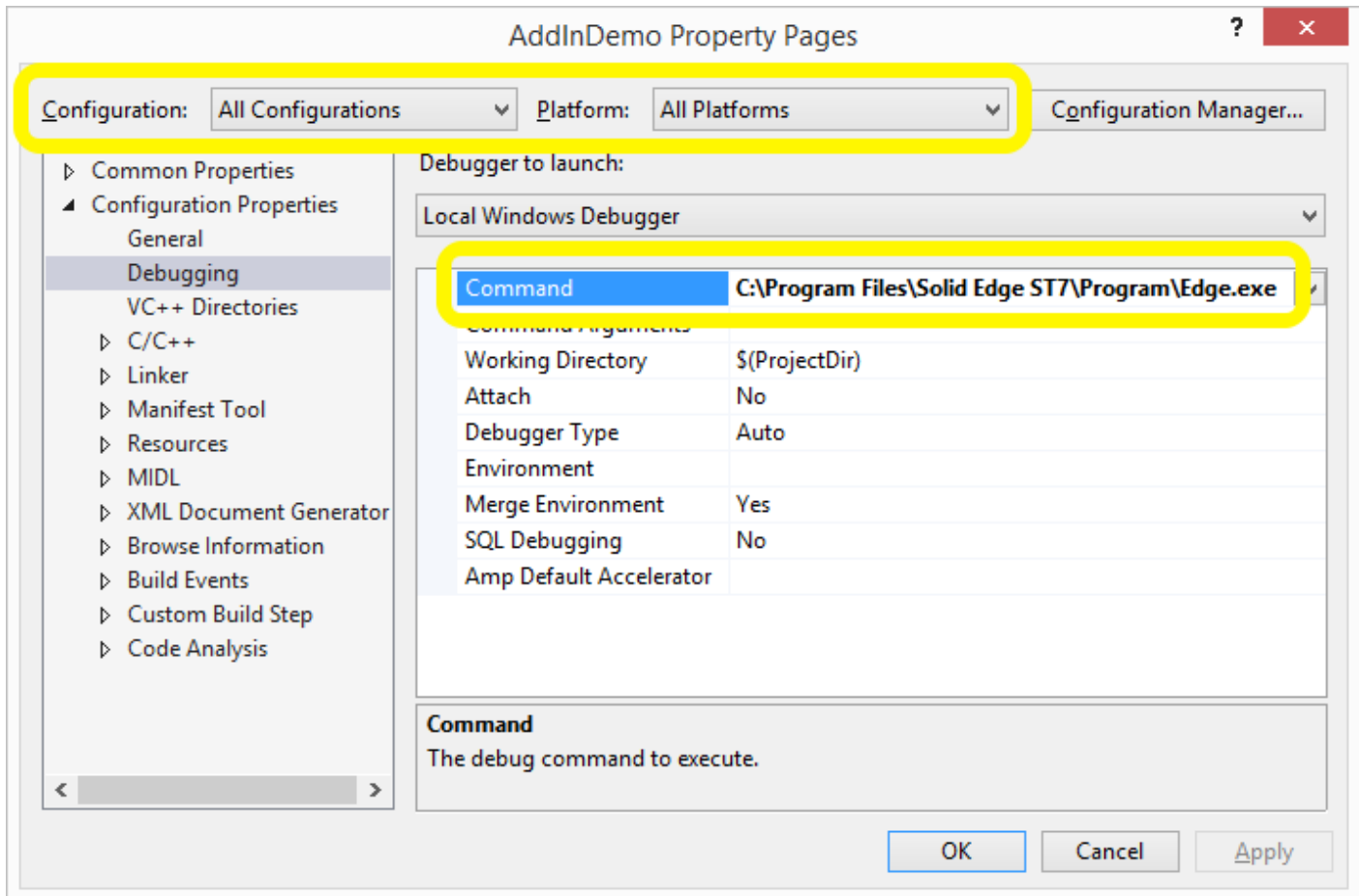
Project Properties

The next step is to configure the project properties. Select the project in Solution Explorer and open the properties dialog.

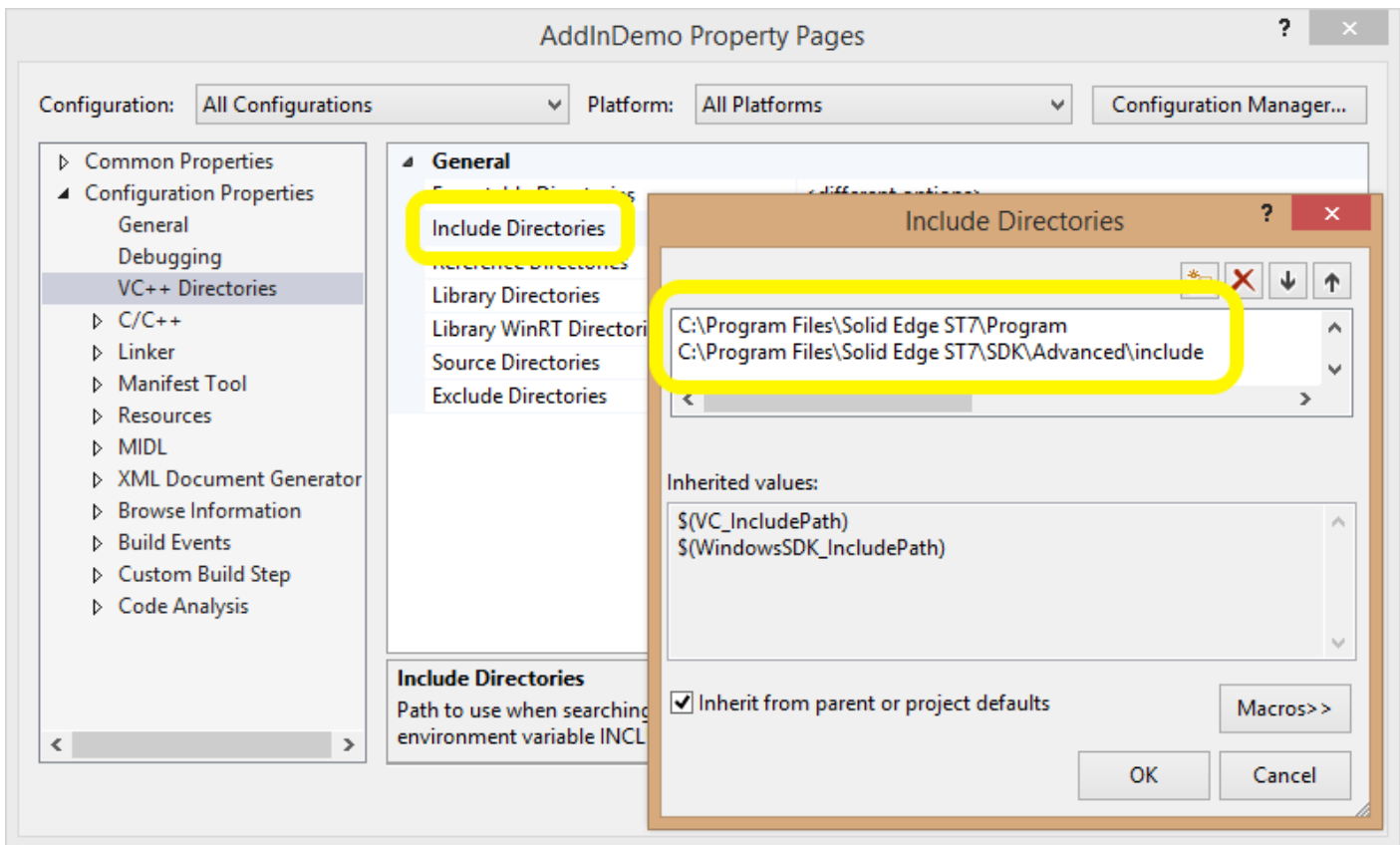


The first thing you want to do is set the Configuration to **All Configurations** and the Platform to **All Platforms**. This will save you from having to repeat these steps later.

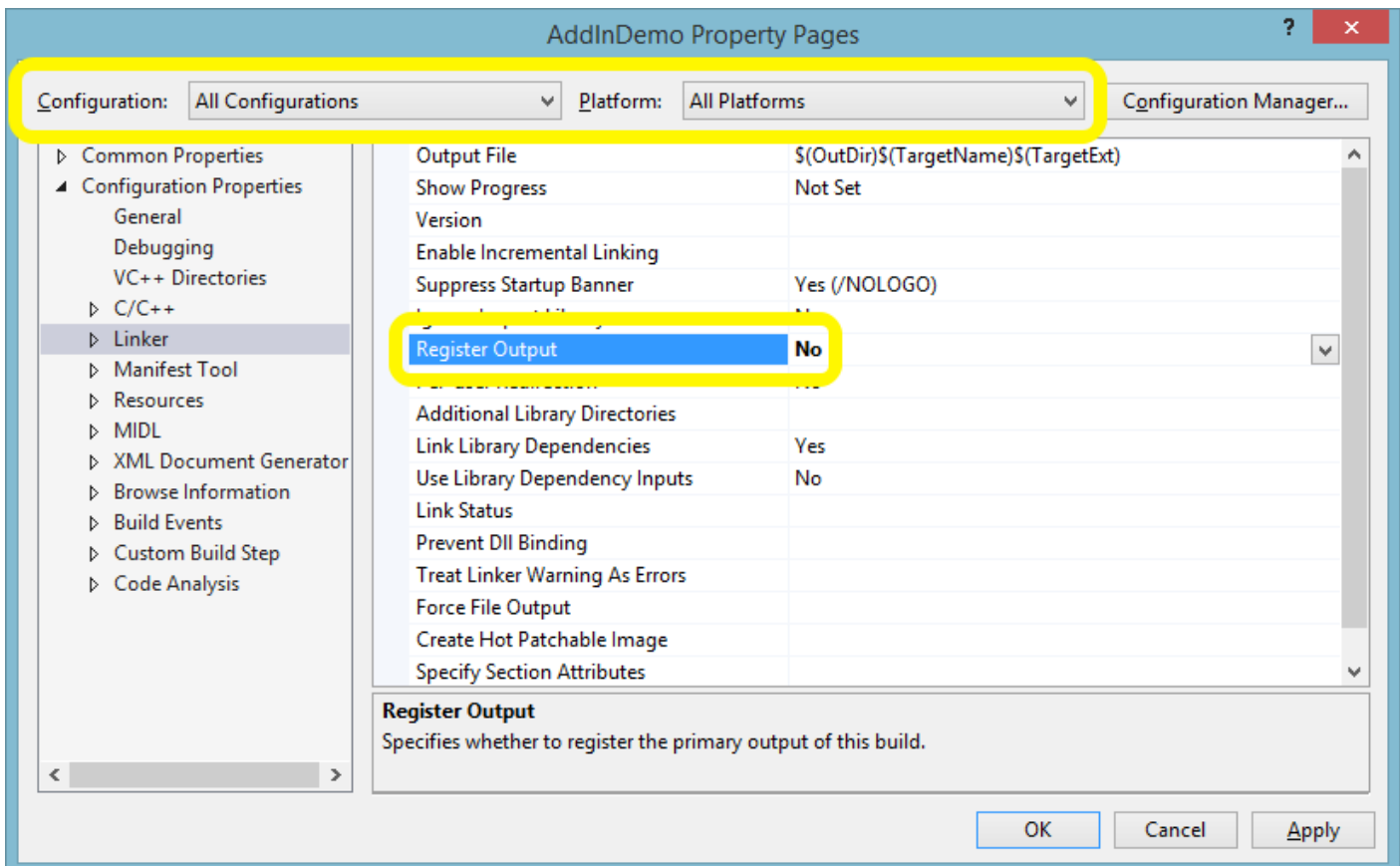
Under the **Debugging** section, modify the **Command** and set it to the path of Edge.exe.



Under the **VC++ Directories** section, modify the **Include Directories** as shown below. This is how the compiler will find the .h and .tlb files that we'll reference later. If you don't have the **\SDK\Advanced** folder, go back and read the [Solid Edge ST7 AddIn Architecture Overview](#).



Under the **Linker** section, set **Register Output** to No. This is optional but I find it easier to manage when the addin gets registered rather than on build. For example, if you're not running Visual Studio 2013 in administrator mode, the register will fail.



Modifying stdafx.h

Add the following code to the bottom of stdafx.h. This will include necessary header files and import the Solid Edge type libraries into your project.

C++

```
#pragma region Solid Edge specific

#include <atlsafe.h> // CComSafeArray
#include <initguid.h> // Necessary for secatids.h.
#include "secatids.h" // C:\Program Files\Solid Edge ST7\SDK\Advanced\include (Extract from DVD)

// SolidEdgeConstants
#import "constant.tlb"

//SolidEdgeFramework
#import "framework.tlb" exclude ("UINT_PTR", "LONG_PTR") rename ("GetOpenFileName", "SEGetOpenFileName")

// SolidEdgeGeometry
#import "geometry.tlb"

// SolidEdgeFrameworkSupport
#import "fwksupp.tlb"

// SolidEdgePart
#import "part.tlb"

// SolidEdgeAssembly
#import "assembly.tlb"

// SolidEdgeDraft
#import "draft.tlb"

// SEInstallDataLib
#import "SEInstallData.dll"

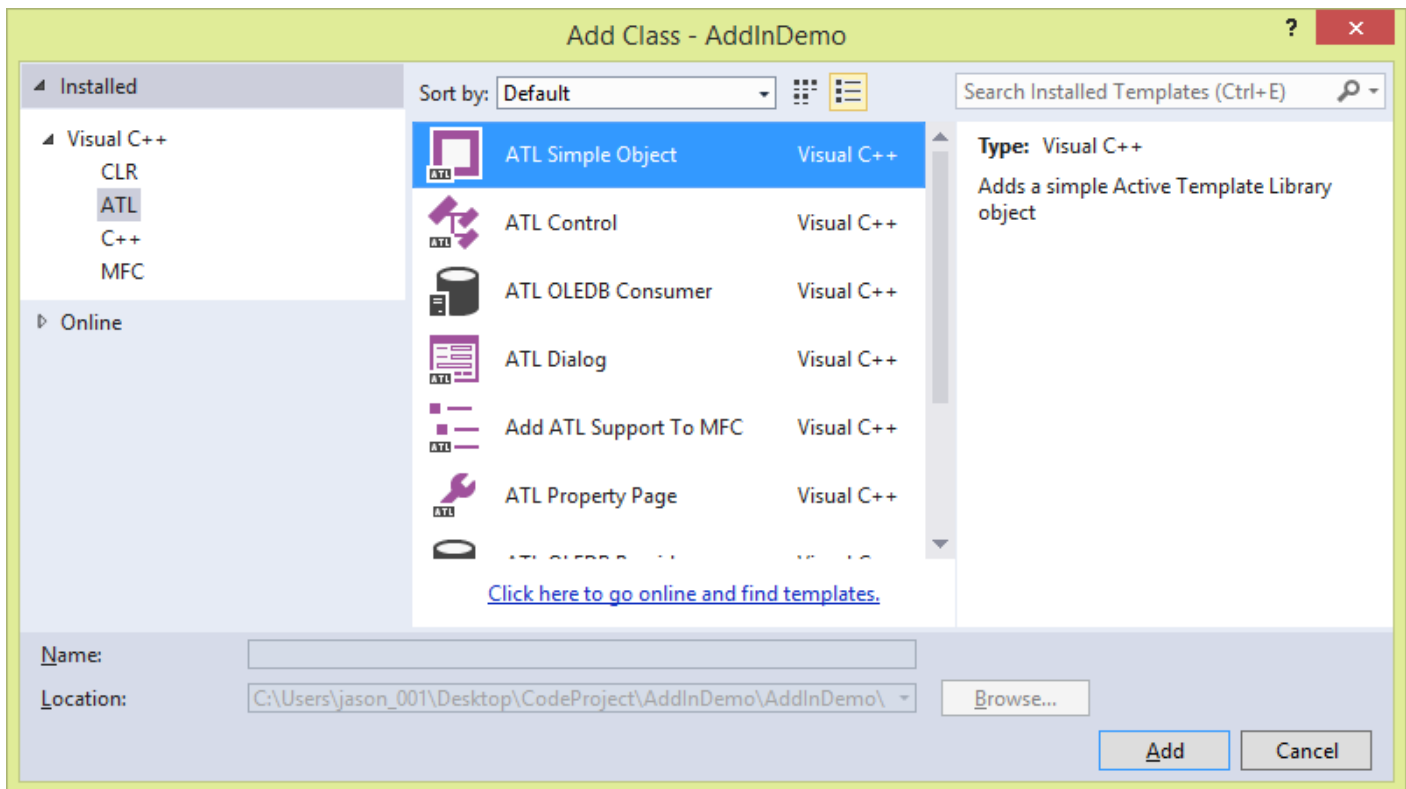
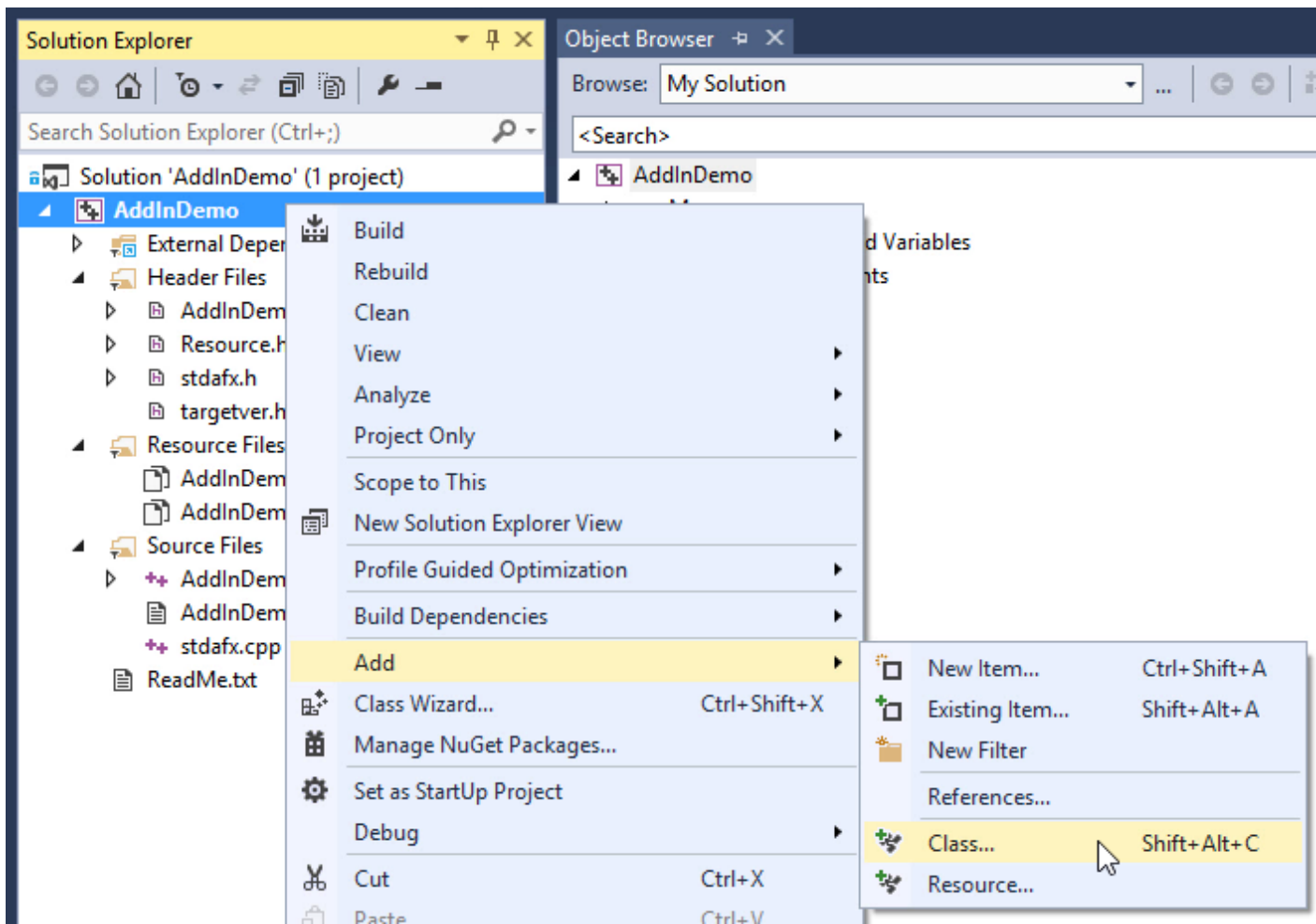
using namespace SolidEdgeFramework;

#define IfFailGo(x) { hr=(x); if (FAILED(hr)) goto Error; }
#define IfFailGoCheck(x, p) { hr=(x); if (FAILED(hr)) goto Error; if(!p) {hr = E_FAIL; goto Error; } }

#pragma endregion
```

Creating the addin class

Now that the project is configured, you're ready to create the addin class. We'll use the **ATL Simple Object** template as a starting point and modify it as we go.



ATL Simple Object Wizard - AddInDemo

Welcome to the ATL Simple Object Wizard

Names
File Type Options
Options

C++

Short name: MyAddIn .h file: MyAddIn.h ...

Class: CMyAddIn .cpp file: MyAddIn.cpp ...

COM

Cgclass: MyAddIn Type: MyAddIn Class X

Interface: IMyAddIn ProgID:

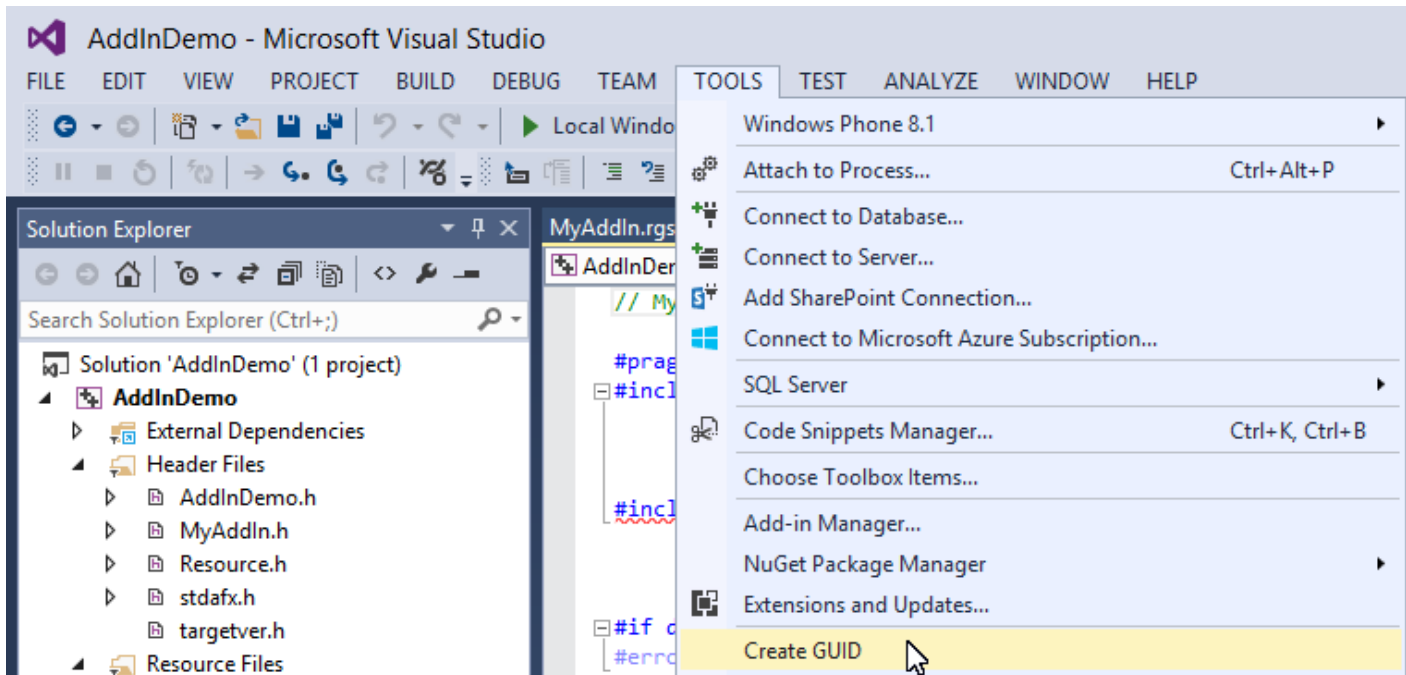
< Previous Next > Finish Cancel

Configuring the addin class

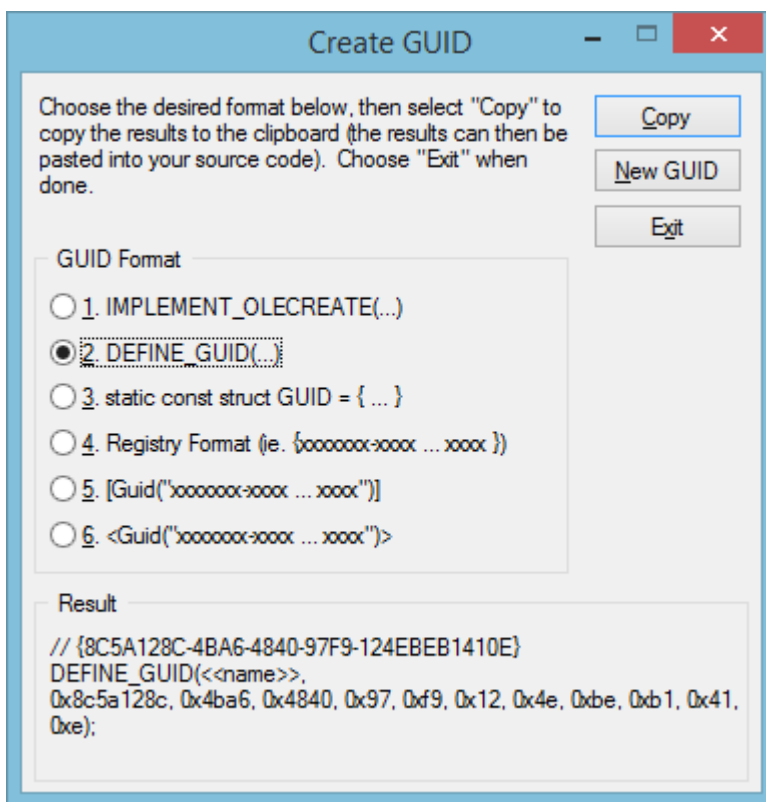
While the class wizard is useful, we do not need or want everything that it produces. Knowing what to modify correctly is the tricky part.

Create a new GUID

The first thing you want to do is to create a new GUID for your addin. The wizard created one for us but we'll create our own.



Click the **Copy** button then the **Exit** button.



This is what the text on the clipboard looks like. We'll add this code to AddInDemo.h in the next step.

C++

```
// {8C5A128C-4BA6-4840-97F9-124EBEB1410E}
DEFINE_GUID(<<name>>,
0x8c5a128c, 0x4ba6, 0x4840, 0x97, 0xf9, 0x12, 0x4e, 0xbe, 0xb1, 0x41, 0xe);
```

Modify MyAddIn.h

Here you can paste in the new GUID that you created in the previous step. Note that you'll have to assign a name to the GUID. In this example, I used **CLSID_MyAddIn**. I remarked out the wizard generated code that we don't need and bolded code that we need to add.

C++

```
// MyAddIn.h : Declaration of the CMyAddIn

#pragma once
#include "resource.h"           // main symbols
#include "AddInDemo_i.h"

using namespace ATL;

// {8C5A128C-4BA6-4840-97F9-124EBEB1410E}
DEFINE_GUID(CLSID_MyAddIn, 0x8c5a128c, 0x4ba6, 0x4840, 0x97, 0xf9, 0x12, 0x4e, 0xbe, 0xb1, 0x41, 0xe);

// CMyAddIn

class ATL_NO_VTABLE CMyAddIn :
public CComObjectRootEx<CComSingleThreadModel>,
public CComCoClass<CMyAddIn, &CLSID_MyAddIn>,
//public IDispatchImpl<IMyAddIn, &IID_IMyAddIn, &LIBID_AddInDemoLib, /*wMajor =*/ 1,
/*wMinor =*/ 0>
public ISolidEdgeAddIn
{
public:
    CMyAddIn()
    {
    }

    DECLARE_REGISTRY_RESOURCEID(IDR_MYADDIN)

    BEGIN_COM_MAP(CMyAddIn)
        COM_INTERFACE_ENTRY(ISolidEdgeAddIn)
        //COM_INTERFACE_ENTRY(IMyAddIn)
        //COM_INTERFACE_ENTRY(IDispatch)
    END_COM_MAP()

    DECLARE_PROTECT_FINAL_CONSTRUCT()

    HRESULT FinalConstruct()
    {
        return S_OK;
    }

    void FinalRelease()
    {
    }

public:
```

```
#pragma region ISolidEdgeAddIn

    STDMETHODCALLTYPE (LPDISPATCH Application,
SolidEdgeFramework::SeConnectMode ConnectMode, SolidEdgeFramework::AddIn * AddInInstance);
    STDMETHODCALLTYPE (BSTR EnvCatID, LPDISPATCH
pEnvironmentDispatch, VARIANT_BOOL bFirstTime);
    STDMETHODCALLTYPE (SolidEdgeFramework::SeDisconnectMode
DisconnectMode);

#pragma endregion
};

//OBJECT_ENTRY_AUTO(__uuidof(MyAddIn), CMyAddIn)
OBJECT_ENTRY_AUTO(CLSID_MyAddIn, CMyAddIn)
```

Modify MyAddIn.cpp

Since we implemented the **ISolidEdgeAddIn** interface in the previous step, we must define the methods as shown below.

C++

```
// MyAddIn.cpp : Implementation of CMyAddIn

#include "stdafx.h"
#include "MyAddIn.h"

// CMyAddIn
#pragma region ISolidEdgeAddIn

STDMETHODIMP CMyAddIn::raw_OnConnection(IDispatch *pAppDispatch,
SolidEdgeFramework::SeConnectMode ConnectMode, SolidEdgeFramework::AddIn* pAddIn)
{
    return S_OK;
}

STDMETHODIMP CMyAddIn::raw_OnConnectToEnvironment(BSTR EnvironmentCatid, LPDISPATCH
pEnvironment, VARIANT_BOOL bFirstTime)
{
    return S_OK;
}

STDMETHODIMP CMyAddIn::raw_OnDisconnection(SolidEdgeFramework::SeDisconnectMode
DisconnectMode)
{
    return S_OK;
}

#pragma endregion
```

Modify MyAddIn.rgs

The wizard created the registry script **MyAddIn.rgs** but we need to change the GUID to use our generated GUID. This script gets executed when the DLL is registered via regsvr32.exe. As discussed in the [Solid Edge ST7 AddIn Architecture Overview](#), you can certainly add the additional Solid Edge addin required registry entries to this file but we will be handling the additional registry entries in code in a later step.

```
HKCR
{
    NoRemove CLSID
    {
        ForceRemove {8C5A128C-4BA6-4840-97F9-124EBEB1410E} = s 'MyAddIn Class'
        {
            ForceRemove Programmable
            InprocServer32 = s '%MODULE%'
            {
                val ThreadingModel = s 'Apartment'
            }
            TypeLib = s '{A43285C8-B0B7-4DB1-BABE-E883E9968AA1}'
            Version = s '1.0'
        }
    }
}
```

Modify AddInDemo.idl

When creating a class via the class wizard, it will add lines to **AddInDemo.idl**. Under normal circumstances this is fine but in our case, we don't really need it as it just gets in the way. I've remarked out the lines that the wizard added as shown below.

MIDL

```
// AddInDemo.idl : IDL source for AddInDemo
//

// This file will be processed by the MIDL tool to
// produce the type library (AddInDemo.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";

//[
//    object,
//    uuid(FBC988E8-2024-4D60-838C-EDD98181A0BA),
//    dual,
//    nonextensible,
//    pointer_default(unique)
//]
//interface IMyAddIn : IDispatch{
//};
[
    uuid(A43285C8-B0B7-4DB1-BABE-E883E9968AA1),
```

```

    version(1.0)
}
library AddInDemoLib
{
    importlib("stdole2.tlb");
    //[
    //    uuid(27CD07FE-C481-4EC6-BA79-7032E1FC18BE)
    //]
    //coclass MyAddIn
    //{
    //    [default] interface IMyAddIn;
    //};
};

```

Registration code

Since [Solid Edge has special requirements for COM addins](#), we'll need to write some code that will create the required registry entries. Later we'll call this code during **DllRegisterServer** and **DllUnregisterServer**.

Create AddInRegistrar.h

Manually create **AddInRegistrar.h**. You can copy\paste the following code into that file.

C++

```

#pragma once
#include "stdafx.h"

class CAddInRegistrar
{
public:
    CAddInRegistrar()
    {
    }

    ~CAddInRegistrar()
    {
    }

    static HRESULT Register(CLSID clsid, LPCTSTR name, LPCTSTR summary, ULONG
cEnvironments, CATID environments[], VARIANT_BOOL bAutoConnect)
    {
        LCID lcid = GetSystemDefaultLCID();
        return CAddInRegistrar::Register(clsid, lcid, name, summary, cEnvironments,
environments, bAutoConnect);
    }

    static HRESULT Register(CLSID clsid, LCID lcid, LPCTSTR name, LPCTSTR summary, ULONG
cEnvironments, CATID environments[], VARIANT_BOOL bAutoConnect)
    {
        HRESULT hr = S_OK;

```

```
#pragma region "HKEY_CLASSES_ROOT\\CLSID\\{ADDIN_GUID}"

LPOLESTR lpClsidString;
StringFromCLSID(clsid, &lpClsidString);

CString strKeyCLSID;
strKeyCLSID.AppendFormat(L"CLSID\\%s", lpClsidString);

CoTaskMemFree(lpClsidString);

CString strValueName;
strValueName.AppendFormat(L"%x", lcid);

CRegKey key;

// Create\Open the key.
LONG result = key.Create(HKEY_CLASSES_ROOT, strKeyCLSID);

if (result == ERROR_SUCCESS)
{
    DWORD autoConnect = bAutoConnect == VARIANT_TRUE ? 1 : 0;

    // Set the AutoConnect flag. (Enabled or Disabled by default).
    key.SetDWORDValue(L"AutoConnect", autoConnect);

    // Set the localized addin name.
    key.SetStringValue(strValueName, name);
    key.Close();
}

#pragma endregion

#pragma region "HKEY_CLASSES_ROOT\\CLSID\\{ADDIN_GUID}\\Summary"

CString strKeySummary;
strKeySummary = strKeyCLSID;
strKeySummary.Append(L"\\Summary");

result = key.Create(HKEY_CLASSES_ROOT, strKeySummary);

if (result == ERROR_SUCCESS)
{
    key.SetStringValue(strValueName, summary);
    key.Close();
}

#pragma endregion

#pragma region "HKEY_CLASSES_ROOT\\CLSID\\{ADDIN_GUID}\\Environment Categories"

CString strKeyEnvironmentCategories;
strKeyEnvironmentCategories = strKeyCLSID;
strKeyEnvironmentCategories.Append(L"\\Environment Categories");

result = key.Create(HKEY_CLASSES_ROOT, strKeyEnvironmentCategories);
```

```

    if (result == ERROR_SUCCESS)
    {
        for (ULONG i = 0; i < cEnvironments; i++)
        {
            LPOLESTR lpCatidString = NULL;
            StringFromCLSID(environments[i], &lpCatidString);

            CString strEnvCATID(lpCatidString);
            CRegKey keyEnvironment;
            keyEnvironment.Create(key, strEnvCATID);

            CoTaskMemFree(lpCatidString);
        }

        key.Close();
    }

#pragma endregion

#pragma region "HKEY_CLASSES_ROOT\\CLSID\\{ADDIN_GUID}\\Implemented Categories"

    ICatRegisterPtr pCatRegister;
    IfFailGoCheck(pCatRegister.CreateInstance(CLSID_StdComponentCategoriesMgr),
    pCatRegister);

    CATID implementedCategories[] = { CATID_SolidEdgeAddIn };
    IfFailGo(pCatRegister->RegisterClassImplCategories(clsid,
    _countof(implementedCategories), implementedCategories));

#pragma endregion

    Error:
        pCatRegister = NULL;
        return hr;
}

static void Unregister(CLSID clsid)
{
    LPOLESTR lpClsidString;
    StringFromCLSID(clsid, &lpClsidString);

    CString strKeyCLSID = lpClsidString;

    CRegKey key;
    key.Open(HKEY_CLASSES_ROOT, L"CLSID");
    key.RecurseDeleteKey(strKeyCLSID);
    key.Close();

    CoTaskMemFree(lpClsidString);
}

};

```

Modify AddInDemo.cpp

This file is long so I'll only include the relevant parts. I added two **#include** statements and modified the **DLLRegisterServer** and **DllUnregisterServer** methods. I bolded the added lines.

C++

```
// AddInDemo.cpp : Defines the initialization routines for the DLL.
//

#include "stdafx.h"
#include "AddInDemo.h"
#include <initguid.h>
#include "AddInDemo_i.c"

#pragma region Solid Edge specific

#include "MyAddIn.h"
#include "AddInRegistrar.h"

#pragma endregion
```

... lines omitted ...

C++

```
// DLLRegisterServer - Adds entries to the system registry
STDAPI DllRegisterServer()
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    _AtlModule.UpdateRegistryAppId(TRUE);
    HRESULT hRes2 = _AtlModule.RegisterServer(TRUE);
    if (hRes2 != S_OK)
        return hRes2;
    if (!COleObjectFactory::UpdateRegistryAll(TRUE))
        return ResultFromScode(SELFREG_E_CLASS);

    #pragma region Solid Edge specific

    // Solid Edge environment(s) that your addin supports. Defined in secatids.h.
    CATID environments[] =
    {
        CATID_SEApplication,
        CATID_SEAllDocumentEnvrionments
    };

    VARIANT_BOOL bAutoConnect = VARIANT_TRUE;

    // Note that Register() is overloaded for additional options.
    CAddInRegistrar::Register(CLSID_MyAddIn, L"AddInDemo", L"My description",
        _countof(environments), environments, bAutoConnect);

    #pragma endregion

    return S_OK;
}
```

```
// DLLUnregisterServer - Removes entries from the system registry
STDAPI DllUnregisterServer()
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    _AtlModule.UpdateRegistryAppId(FALSE);
    HRESULT hRes2 = _AtlModule.UnregisterServer(TRUE);
    if (hRes2 != S_OK)
        return hRes2;
    if (!COleObjectFactory::UpdateRegistryAll(FALSE))
        return ResultFromScode(SELFREG_E_CLASS);

#pragma region Solid Edge specific

    CAddInRegistrar::Unregister(CLSID_MyAddIn);

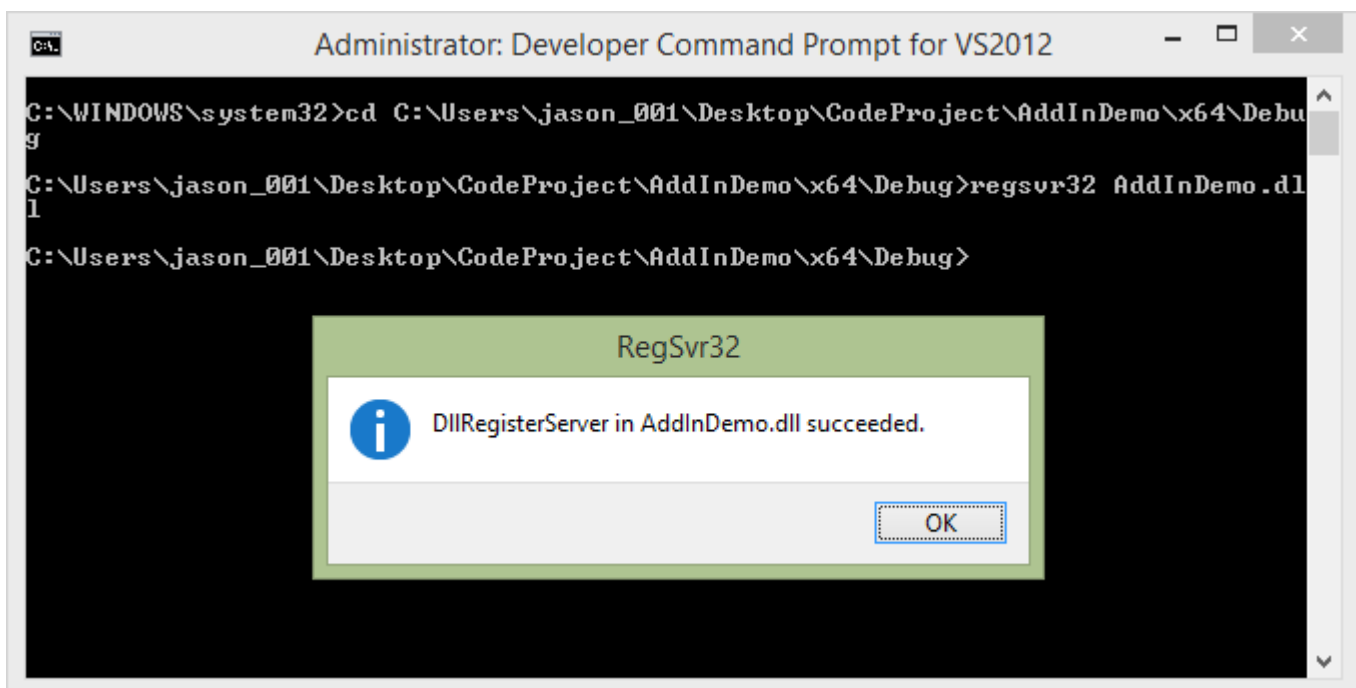
#pragma endregion

    return S_OK;
}
```

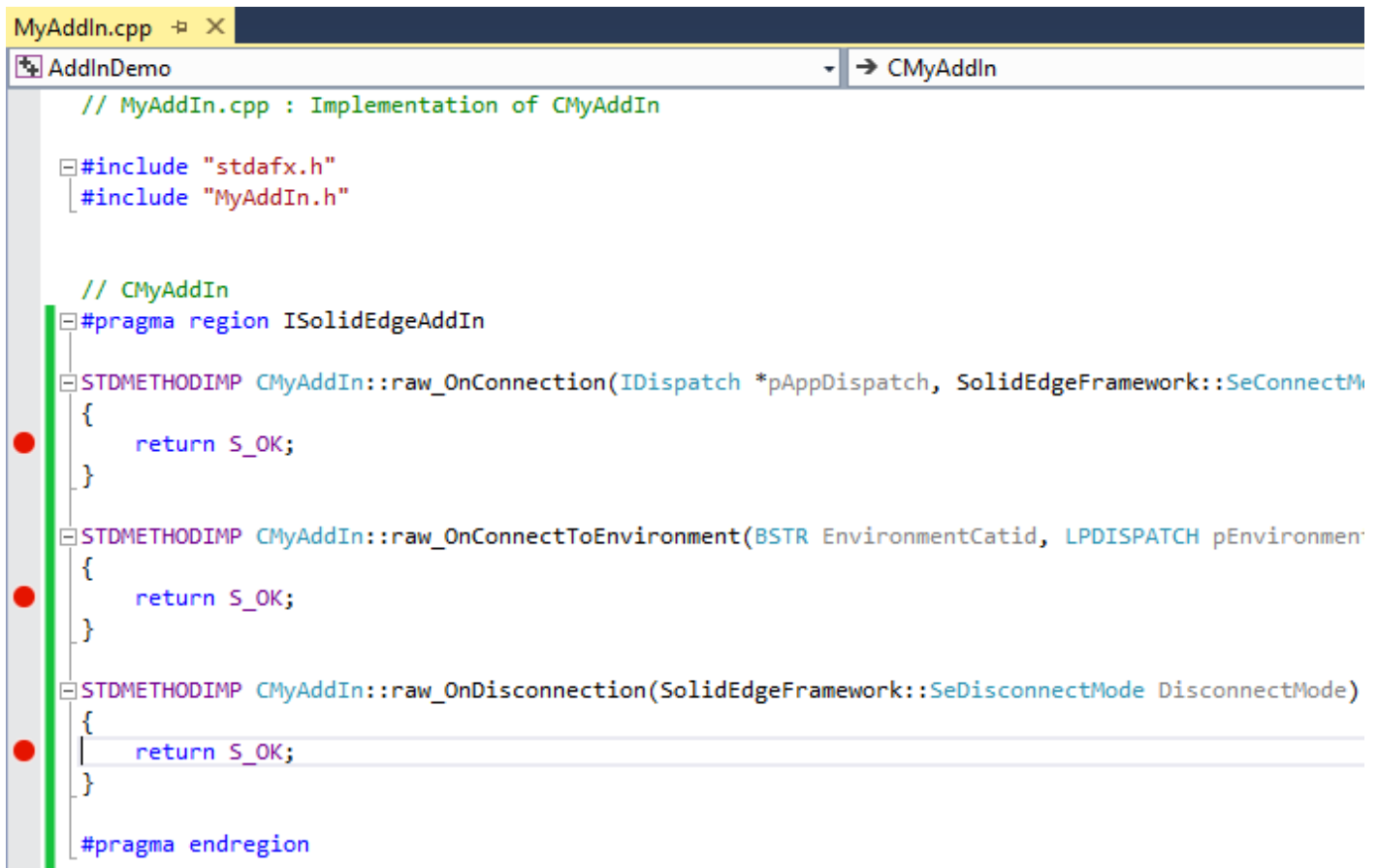
Build, Register and Debug

If you got all of the previous steps right, you should be able to compile the project. If not, go back and check to see what you missed. Once you're able to compile without errors, you're ready to register your addin. Open a command prompt with administrator privileges and set the current directory to the path to the DLL. You can then use **regsvr32.exe** to register the .dll.

```
CD %ADDIN_DLL_FOLDER%
REGSVR32 %ADDIN_DLL%
```



If you're able to successfully register your addin, you're ready to start debugging. The best way to check to see if Solid Edge loads your addin correctly is to set breakpoints in the `raw_OnConnection`, `raw_OnConnectToEnvironment` and `raw_OnDisconnection` method implementations.



```

MyAddIn.cpp
AddInDemo
// MyAddIn.cpp : Implementation of CMyAddIn

#include "stdafx.h"
#include "MyAddIn.h"

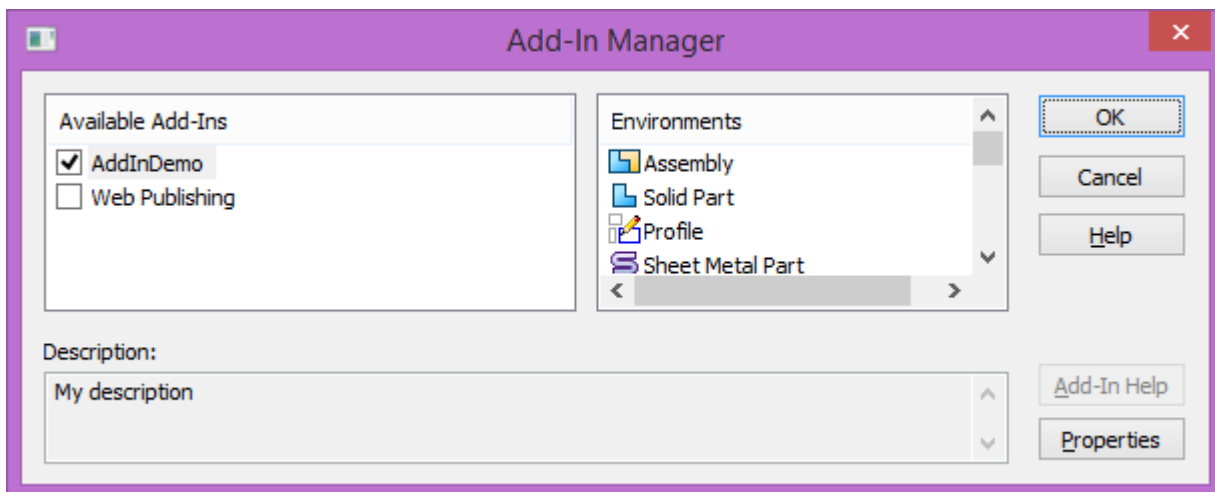
// CMyAddIn
#pragma region ISolidEdgeAddIn
STDMETHODIMP CMyAddIn::raw_OnConnection(IDispatch *pAppDispatch, SolidEdgeFramework::SeConnectM
{
    return S_OK;
}

STDMETHODIMP CMyAddIn::raw_OnConnectToEnvironment(BSTR EnvironmentCatid, LPDISPATCH pEnvironmen
{
    return S_OK;
}

STDMETHODIMP CMyAddIn::raw_OnDisconnection(SolidEdgeFramework::SeDisconnectMode DisconnectMode)
{
    return S_OK;
}

#pragma endregion
  
```

You can also open the Solid Edge Add-In Manager to verify that Solid Edge sees your addin.



To unregister your addin, open a command prompt with administrator privileges and set the current directory to the path to the DLL. You can then use **regsvr32.exe /u** to register the .dll.

```
CD %ADDIN_DLL_FOLDER%  
REGSVR32 /U %ADDIN_DLL%
```

Conclusion

This article only focused on the bare minimum requirements to writing a Solid Edge AddIn. In future articles, I will focus on other topics like sinking events and adding commands to the Ribbon.

Resources

- [Solid Edge ST7 SDK](#)
- [Solid Edge Developer Community](#)
- [Solid Edge Community on GitHub](#)

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)

Written By

Jason Newell


Software Developer (Senior) JasonNewell.NET

 United States

Jason Newell is an Applications Architect for [Ditch Witch](#) and a Siemens PLM Solutions Partner specializing in custom development for [Solid Edge](#). Jason also runs www.jasonnewell.net, a website dedicated to [Solid Edge](#) programming.



Comments and Discussions

 **3 messages** have been posted for this article Visit

<https://www.codeproject.com/Articles/840912/Solid-Edge-ST-AddIn-CPP> to post and view comments on this article, or click [here](#) to get a print view with messages.

[Permalink](#)
[Advertise](#)
[Privacy](#)
[Cookies](#)
[Terms of Use](#)

Article Copyright 2014 by Jason Newell
Everything else Copyright © [CodeProject](#),
1999-2023

Web01 2.8:2023-07-24:2