Tech Note #012

Using Universal SDK Libraries from C++

Rev. A

Revised 1/16/2017

International Technologies & Systems Corporation
10721 Walker Street, Cypress, CA 90630-4720; Tel: (714) 761-6368; Fax (714) 761-8880
www.idtechproducts.com

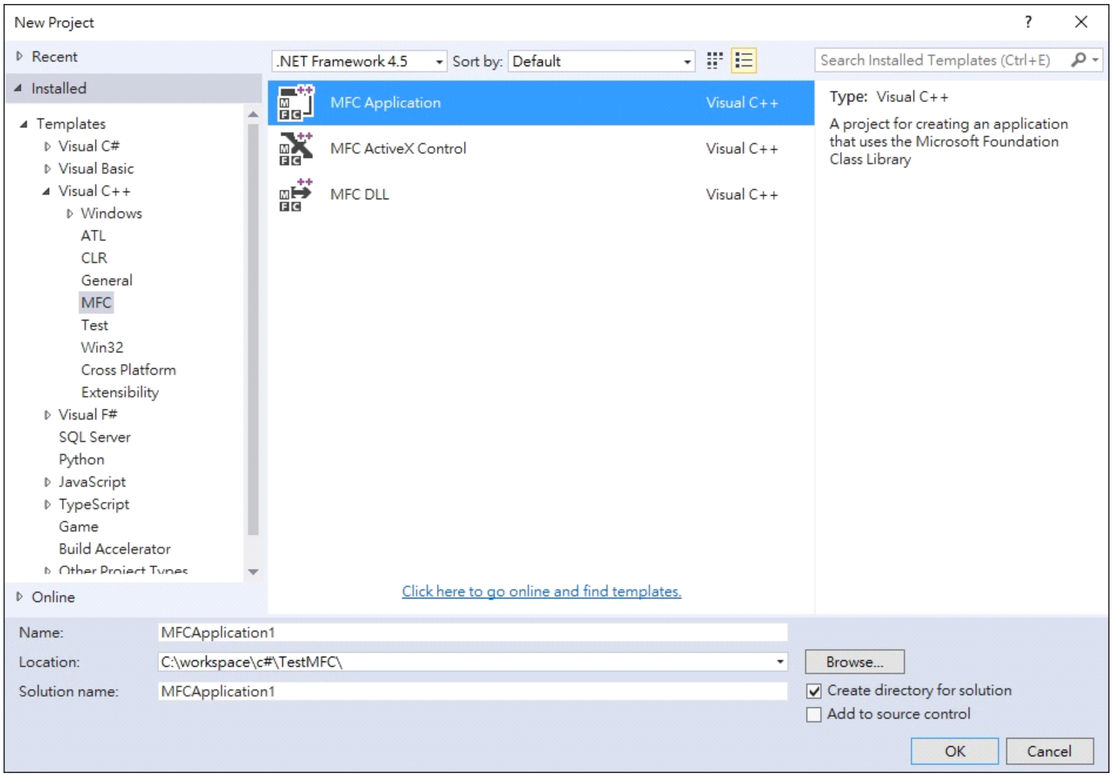
Introduction

From time to time, customers who wish to use ID TECH's Universal SDK for Windows (which is .NET-based and comes with C# code examples) ask if it is possible to do development against the SDK solely in C++ (on Windows). The answer is yes. Universal SDK library files (DLLs) are COM-visible and ready to be accessed from C++ code. (SDK runtimes require the .NET Common Language Runtime, but your C++ binaries can still use the SDK.)

Note that while the example shown in this document involves Microsoft's Visual Studio, it is also possible to use SDK libraries in C++ projects created in Eclipse or other IDEs.

How to Use the IDTechSDK.dll File in a C++ Project:

1. Create a Visual C++ project in Visual Studio 2015 (shown below, an MFC Application as an example).



.NET Framework 4.5

Sort by: Default

MFC Application Visual C++

Type: Visual C++

A project for creating an application that uses the Microsoft Foundation Class Library

Name: MFCApplication1

Location: C:\workspace\c#\TestMFC\

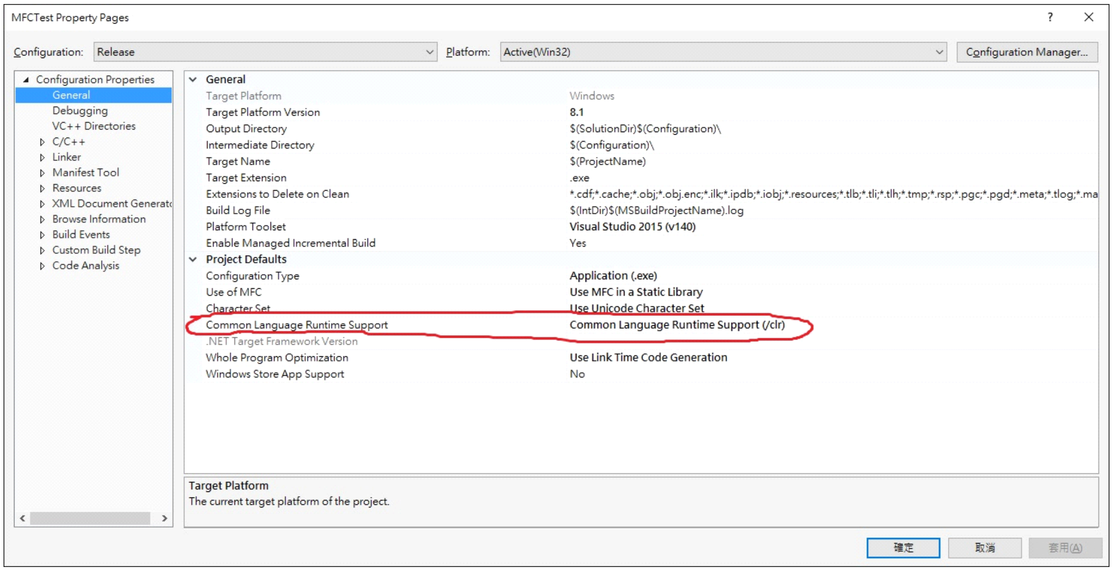
Solution name: MFCApplication1

☑ Create directory for solution

□ Add to source control

2. Change the properties of the Visual C++ project.

Under the **General** tag, set Commom Language Runtime Support under Target Platform to "Common Language Runtime Support (/clr)" under Windows.

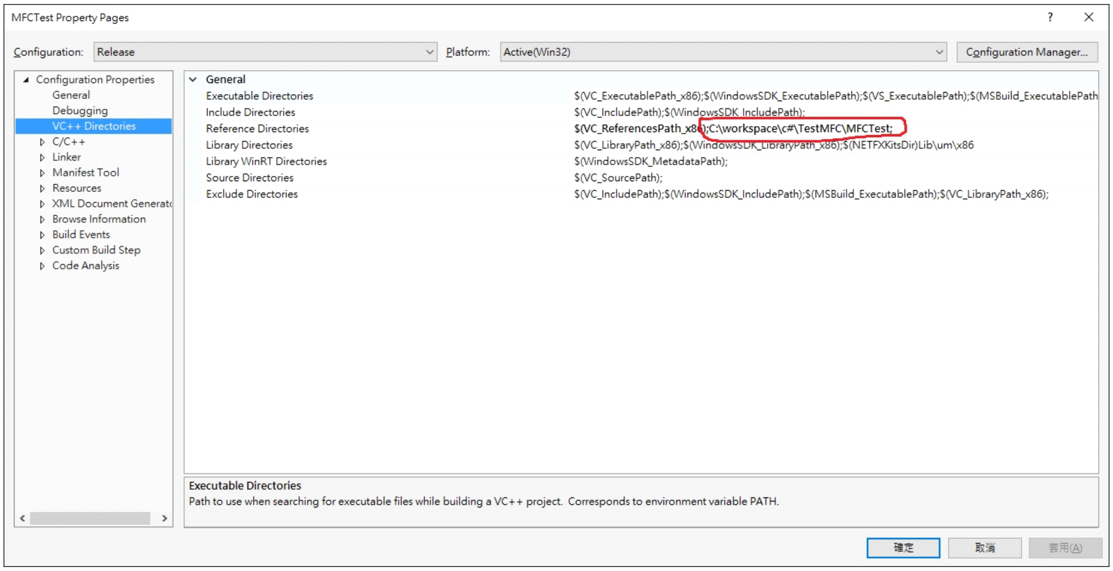


Configuration: Release
Platform: Active(Win32)
Configuration Manager...

General	
Target Platform	Windows
Target Platform Version	8.1
Output Directory	$(SolutionDir)$(Configuration)\
Intermediate Directory	$(Configuration)\
Target Name	$(ProjectName)
Target Extension	.exe
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.obj.enc;*.ilk;*.ipdb;*.iobj;*.resources;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;*.meta;*.tlog;*.ma
Build Log File	$(IntDir)$(MSBuildProjectName).log
Platform Toolset	Visual Studio 2015 (v140)
Enable Managed Incremental Build	Yes
Project Defaults	
Configuration Type	Application (.exe)
Use of MFC	Use MFC in a Static Library
Character Set	Use Unicode Character Set
Common Language Runtime Support	Common Language Runtime Support (/clr)
.NET Target Framework Version	
Whole Program Optimization	Use Link Time Code Generation
Windows Store App Support	No

Target Platform
The current target platform of the project.

3. Under **VC++ Directories**, add the path to the C# .dll file(s) to Reference Directories.

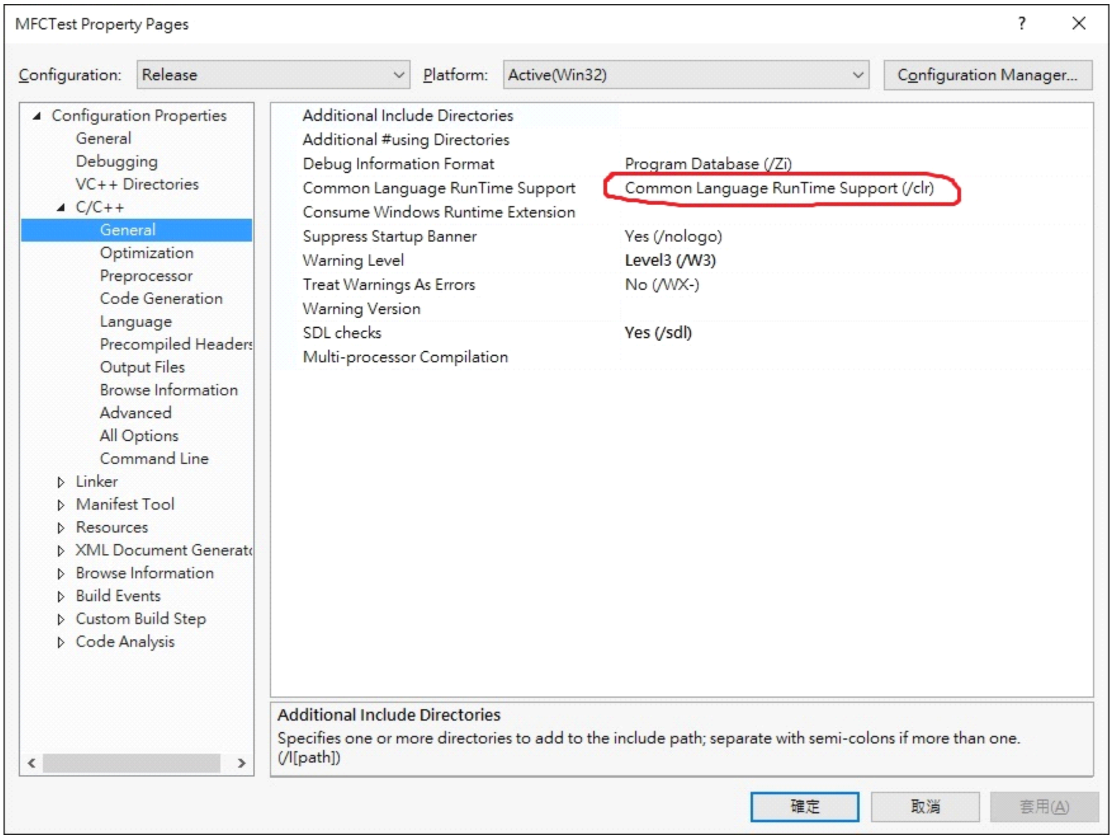


Configuration: Release
Platform: Active(Win32)
Configuration Manager...

General	
Executable Directories	$(VC_ExecutablePath_x86);$(WindowsSDK_ExecutablePath);$(VS_ExecutablePath);$(MSBuild_ExecutablePath
Include Directories	$(VC_IncludePath);$(WindowsSDK_IncludePath);
Reference Directories	$(VC_ReferencesPath_x86);C:\workspace\c#\TestMFC\MFCTest;
Library Directories	$(VC_LibraryPath_x86);$(WindowsSDK_LibraryPath_x86);$(NETFXKitsDir)Lib\um\x86
Library WinRT Directories	$(WindowsSDK_MetadataPath);
Source Directories	$(VC_SourcePath);
Exclude Directories	$(VC_IncludePath);$(WindowsSDK_IncludePath);$(MSBuild_ExecutablePath);$(VC_LibraryPath_x86);

Executable Directories
Path to use when searching for executable files while building a VC++ project. Corresponds to environment variable PATH.

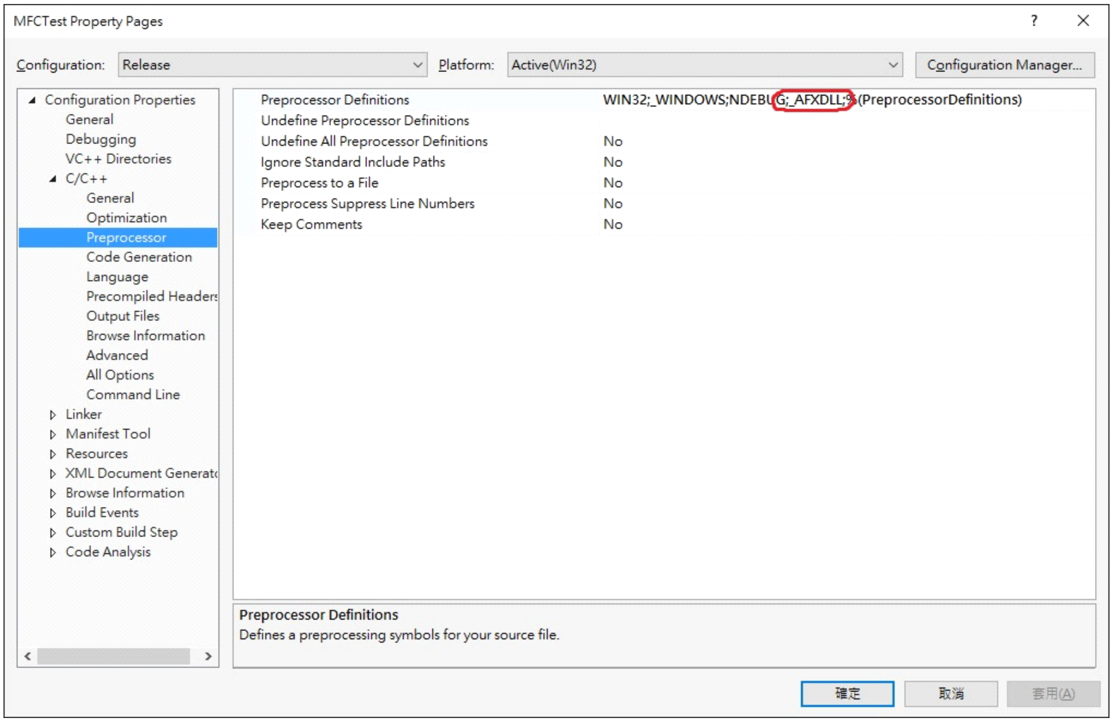
4. Under **C/C++ General**, set Commom Language Runtime Support to "Common Language Runtime Support (/clr)."



Additional Include Directories
Additional #using Directories
Debug Information Format — Program Database (/Zi)
Common Language RunTime Support — Common Language RunTime Support (/clr)
Consume Windows Runtime Extension
Suppress Startup Banner — Yes (/nologo)
Warning Level — Level3 (/W3)
Treat Warnings As Errors — No (/WX-)
Warning Version
SDL checks — Yes (/sdl)
Multi-processor Compilation

Additional Include Directories
Specifies one or more directories to add to the include path; separate with semi-colons if more than one. (/I[path])

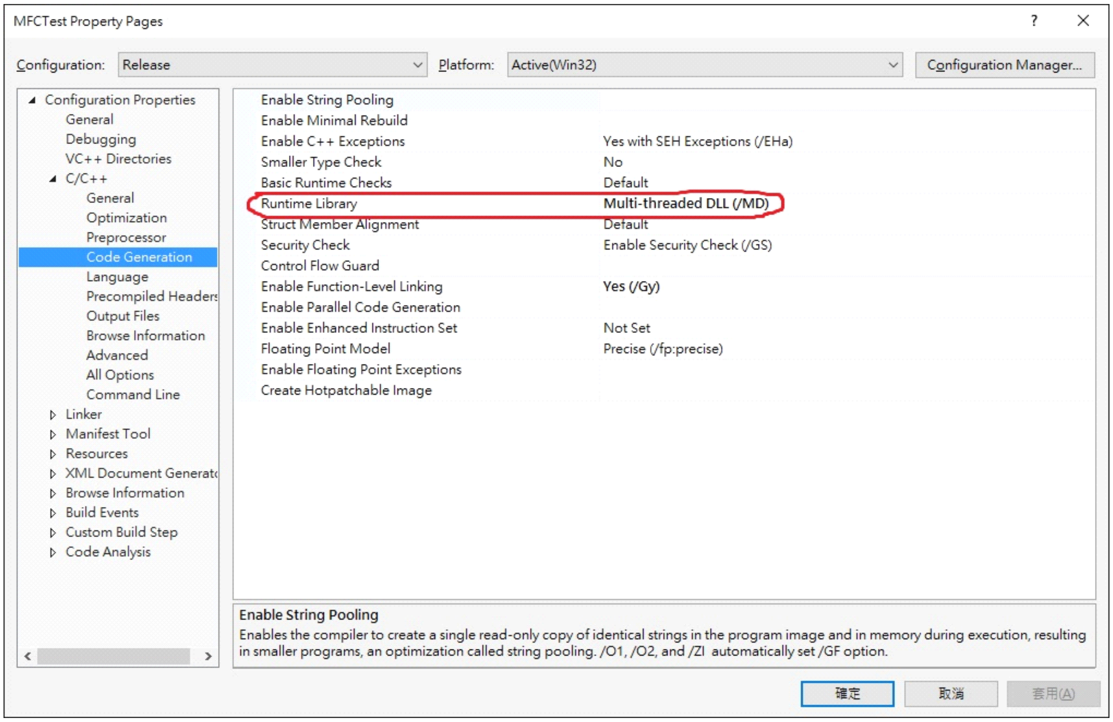
5. Under **C/C++ Preprocessor**, add _AFXDLL to Preprocessor Definitions.



Preprocessor Definitions — WIN32;_WINDOWS;NDEBUG;_AFXDLL;%(PreprocessorDefinitions)
Undefine Preprocessor Definitions
Undefine All Preprocessor Definitions — No
Ignore Standard Include Paths — No
Preprocess to a File — No
Preprocess Suppress Line Numbers — No
Keep Comments — No

Preprocessor Definitions
Defines a preprocessing symbols for your source file.

6. Under **C/C++ Code Generation**, change Runtime Library to "Multi-threaded DLL (/MD)."



Configuration: Release

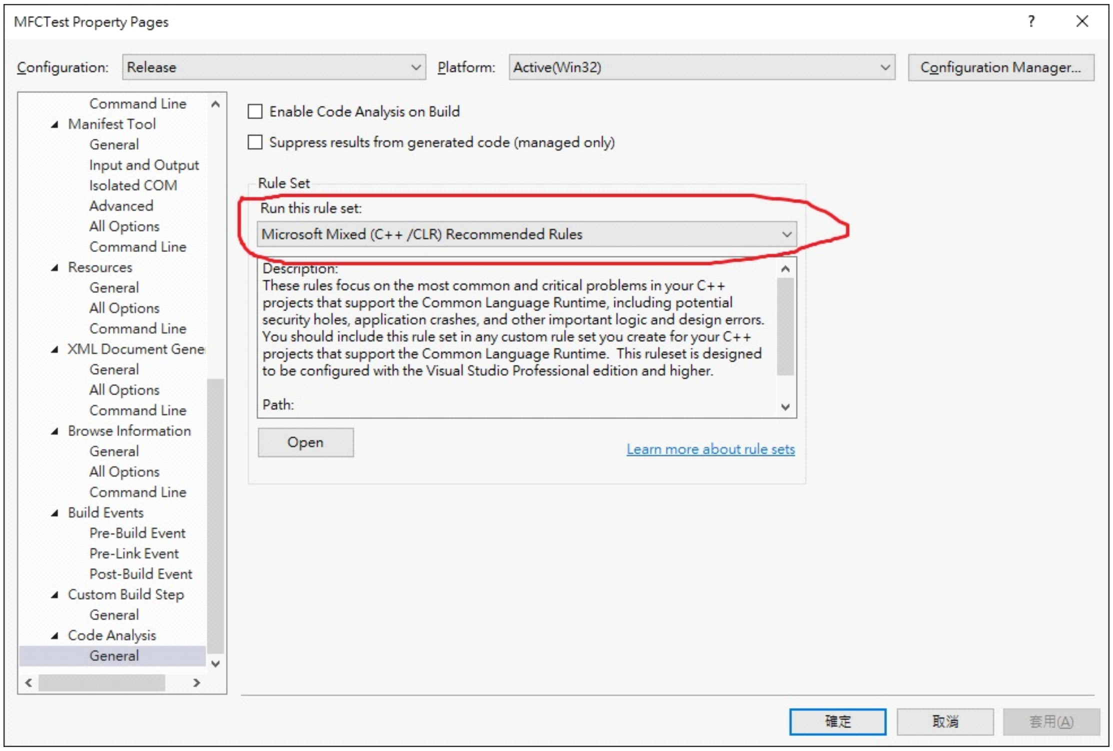
Platform: Active(Win32)

Enable String Pooling	
Enable Minimal Rebuild	
Enable C++ Exceptions	Yes with SEH Exceptions (/EHa)
Smaller Type Check	No
Basic Runtime Checks	Default
Runtime Library	Multi-threaded DLL (/MD)
Struct Member Alignment	Default
Security Check	Enable Security Check (/GS)
Control Flow Guard	
Enable Function-Level Linking	Yes (/Gy)
Enable Parallel Code Generation	
Enable Enhanced Instruction Set	Not Set
Floating Point Model	Precise (/fp:precise)
Enable Floating Point Exceptions	
Create Hotpatchable Image	

Enable String Pooling

Enables the compiler to create a single read-only copy of identical strings in the program image and in memory during execution, resulting in smaller programs, an optimization called string pooling. /O1, /O2, and /ZI automatically set /GF option.

7. Under **Code Analysis General**, change Rule Set to "Microsoft Mixed (C++ /CLR) Recommended Rules."



Configuration: Release

Platform: Active(Win32)

☐ Enable Code Analysis on Build

☐ Suppress results from generated code (managed only)

Rule Set

Run this rule set:

Microsoft Mixed (C++ /CLR) Recommended Rules

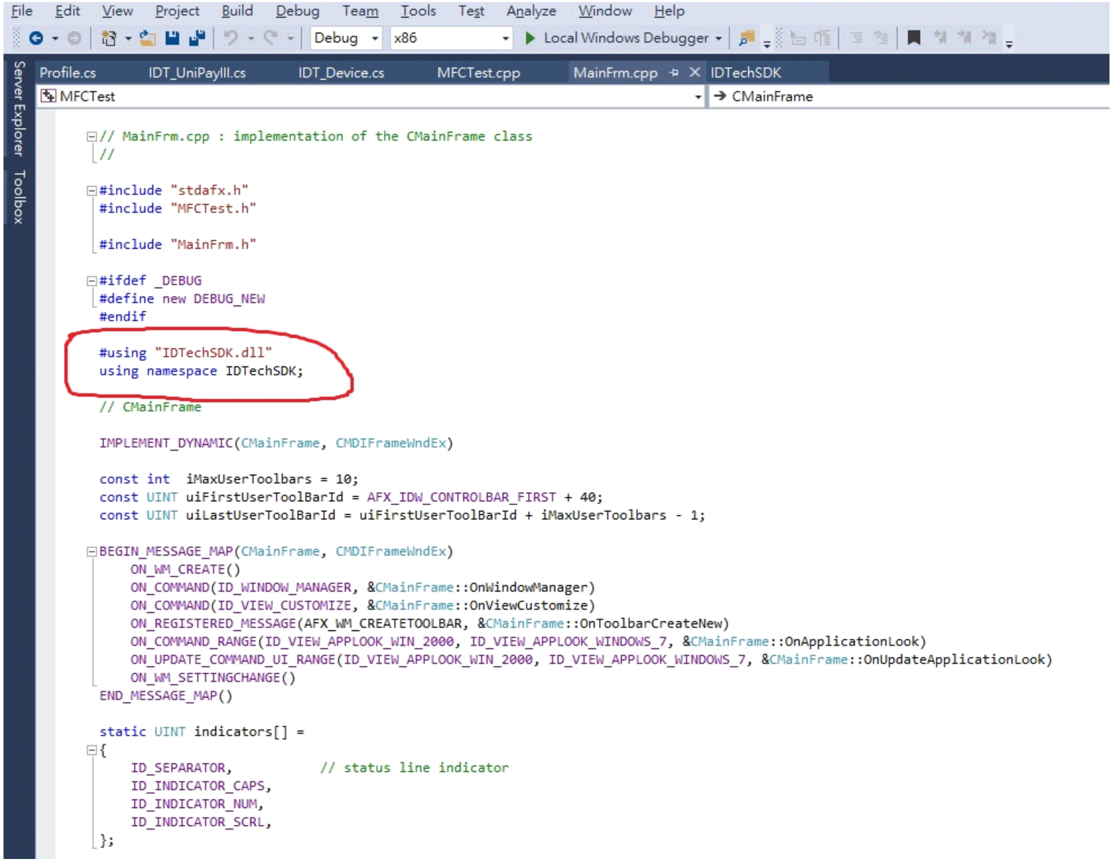
Description:
These rules focus on the most common and critical problems in your C++ projects that support the Common Language Runtime, including potential security holes, application crashes, and other important logic and design errors. You should include this rule set in any custom rule set you create for your C++ projects that support the Common Language Runtime. This ruleset is designed to be configured with the Visual Studio Professional edition and higher.

Path:

8. Use **IDTechSDK.dll** in your .cpp file.

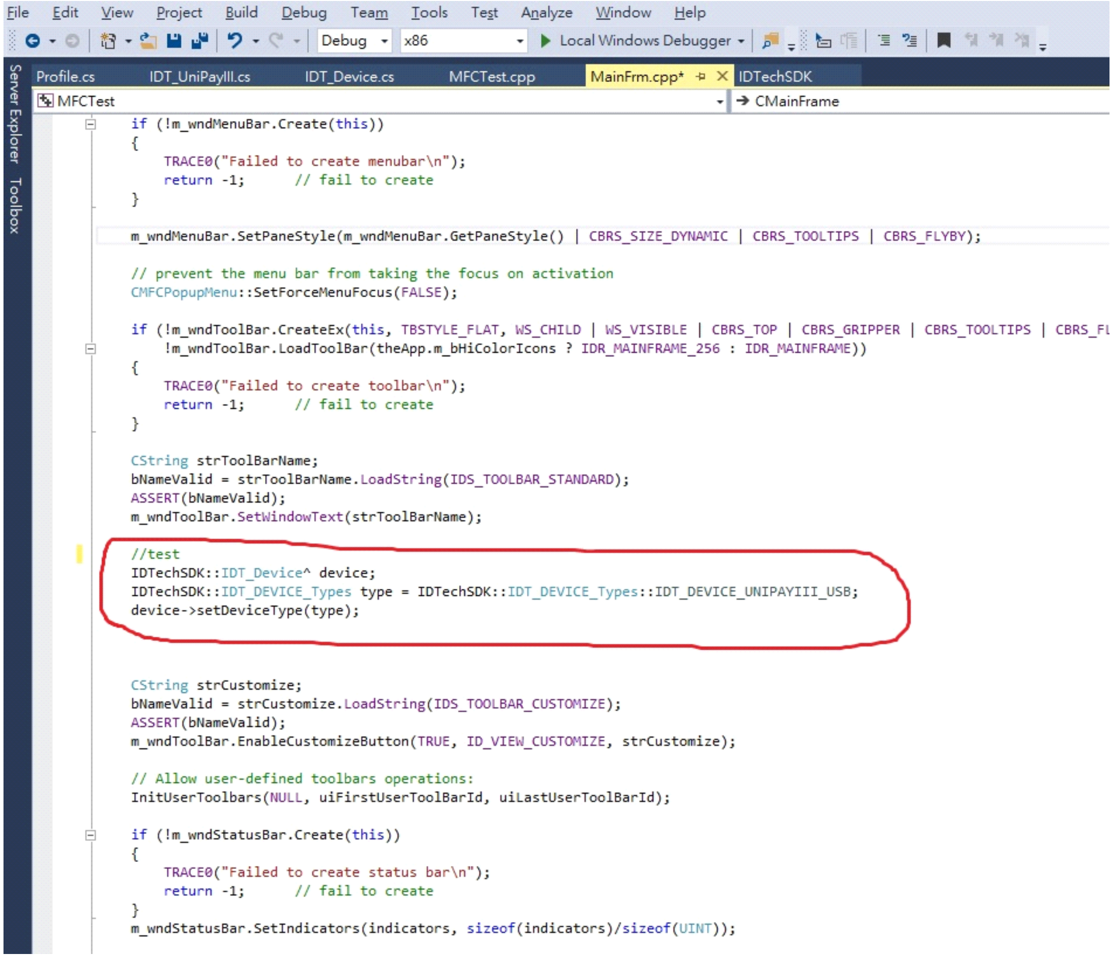
a. Open a .cpp file in the Visual C++ project (MainFrm.cpp, for example).

b. Add `#using "IDTechSDK.dll"` and `using namespace IDTechSDK` below any #include and #define statements.



```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "MFCTest.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

#using "IDTechSDK.dll"
using namespace IDTechSDK;

// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWndEx)

const int  iMaxUserToolbars = 10;
const UINT uiFirstUserToolBarId = AFX_IDW_CONTROLBAR_FIRST + 40;
const UINT uiLastUserToolBarId = uiFirstUserToolBarId + iMaxUserToolbars - 1;

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWndEx)
    ON_WM_CREATE()
    ON_COMMAND(ID_WINDOW_MANAGER, &CMainFrame::OnWindowManager)
    ON_COMMAND(ID_VIEW_CUSTOMIZE, &CMainFrame::OnViewCustomize)
    ON_REGISTERED_MESSAGE(AFX_WM_CREATETOOLBAR, &CMainFrame::OnToolbarCreateNew)
    ON_COMMAND_RANGE(ID_VIEW_APPLOOK_WIN_2000, ID_VIEW_APPLOOK_WINDOWS_7, &CMainFrame::OnApplicationLook)
    ON_UPDATE_COMMAND_UI_RANGE(ID_VIEW_APPLOOK_WIN_2000, ID_VIEW_APPLOOK_WINDOWS_7, &CMainFrame::OnUpdateApplicationLook)
    ON_WM_SETTINGCHANGE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};
```

c. Declare an object and call the functions in IDTechSDK.dll.



```
if (!m_wndMenuBar.Create(this))
{
    TRACE0("Failed to create menubar\n");
    return -1;      // fail to create
}

m_wndMenuBar.SetPaneStyle(m_wndMenuBar.GetPaneStyle() | CBRS_SIZE_DYNAMIC | CBRS_TOOLTIPS | CBRS_FLYBY);

// prevent the menu bar from taking the focus on activation
CMFCPopupMenu::SetForceMenuFocus(FALSE);

if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FL
    !m_wndToolBar.LoadToolBar(theApp.m_bHiColorIcons ? IDR_MAINFRAME_256 : IDR_MAINFRAME))
{
    TRACE0("Failed to create toolbar\n");
    return -1;      // fail to create
}

CString strToolBarName;
bNameValid = strToolBarName.LoadString(IDS_TOOLBAR_STANDARD);
ASSERT(bNameValid);
m_wndToolBar.SetWindowText(strToolBarName);

//test
IDTechSDK::IDT_Device^ device;
IDTechSDK::IDT_DEVICE_Types type = IDTechSDK::IDT_DEVICE_Types::IDT_DEVICE_UNIPAYIII_USB;
device->setDeviceType(type);


CString strCustomize;
bNameValid = strCustomize.LoadString(IDS_TOOLBAR_CUSTOMIZE);
ASSERT(bNameValid);
m_wndToolBar.EnableCustomizeButton(TRUE, ID_VIEW_CUSTOMIZE, strCustomize);

// Allow user-defined toolbars operations:
InitUserToolbars(NULL, uiFirstUserToolBarId, uiLastUserToolBarId);

if (!m_wndStatusBar.Create(this))
{
    TRACE0("Failed to create status bar\n");
    return -1;      // fail to create
}
m_wndStatusBar.SetIndicators(indicators, sizeof(indicators)/sizeof(UINT));
```

9. Finally, clean and build the Visual C++ project. Copy IDTechSDK.dll and all the other provided SDK .dll files to Debug and Release folders in the Visual C++ project.