GRX Programming Guide

GstarCAD 2027

Table of Contents

1. What is GRX?

GRX is the Runtime eXtension programming environment of CAD, which is the latest application development SDK. It provides Object-oriented development environment and APIs based on C++, which can be used to develop application programs, extension CAD classes and protocols, and create new commands.

GRX is compatible with ARX application programs of AutoCAD® on the source-code level.

GRX SDK allows direct access to database, graphics system and user-defined commands, etc. In addition, it can also be used with VLISP and other programming languages.

Developers only need few configurations to migrate ARX application for AutoCAD® to GRX application.

The following works can be accomplished with GRX:

- Access the CAD database
- Interact with the CAD editor
- Create user interfaces using the Microsoft® Foundation Classes (MFC) (Windows only)
- Support the multiple document interface (MDI)
- Create custom classes
- Build complex applications
- Interact with other programming environments

2. Programming Environment

- Microsoft® Visual Studio 2022 (version 17.8.0)
- Windows SDK 10.0 (latest installed version)
- CPU:

  Basic: 1.6 GHz CPU

  Recommended: 3.0 GHz CPU and above
- RAM:

  Basic: 2 GB

  Recommended: 8 GB and above
- Operation System (OS)

  Windows 11

  Windows 10 (version 1507 and above):

  Home, Professional, Education and Enterprise (not support LTSC and Windows 10 S)
- Monitor Resolution:

  1028x800 and above true color display, including 4K (3840x2160) display
- GRXSDK
- Microsoft .NET 8.0

3. Install SDK

Download SDK.

Unzip GRXSDK.ZIP file to the local disk (e.g. '*C:\grxsdk*') and there will be 5 directories generated (in '*C:\grxsdk*') which are: **arx**, **inc**, **inc-x64**, **lib-x64** and **utils**.

arx contains the header files, library files and sample programs used for porting ARX programs to GRX programs. It contains the following directories:

- **inc:** Header files used for porting from ARX to GRX
- **inc-x64**: Files used by COM and .NET (for 64-bit)
- **lib-x64**: GRX libraries (for 64-bit)
- **Samples**: Sample projects, including dotnet, fact_dg, HelloADS, HelloARX, SimplePalette.
  - **Dotnet**: .NET programming samples
    1) **Addline**: .NET programming sample of adding solid lines
    2) **Hello**: .NET programming sample of outputting prompt information
    3) **Vbhello**: Sample of .NET programming with VB .NET
  - **fact_dg**: Sample of LISP function definition
  - **HelloADS**: Sample of ADS programming
  - **HelloARX**: Sample of GRX programming
  - **SimplePalette**: Programming sample of how to create a set Palette windows
- **Utils:** Directory contains sub-directories of GRX extended applications, including APIs for extended function development, e.g. BREP for boundary representation.

inc: Header files used for programming the GRX

inc-x64: Files used by COM and .NET (for 64-bit)

lib-x64: GRX libraries (for 64-bit)

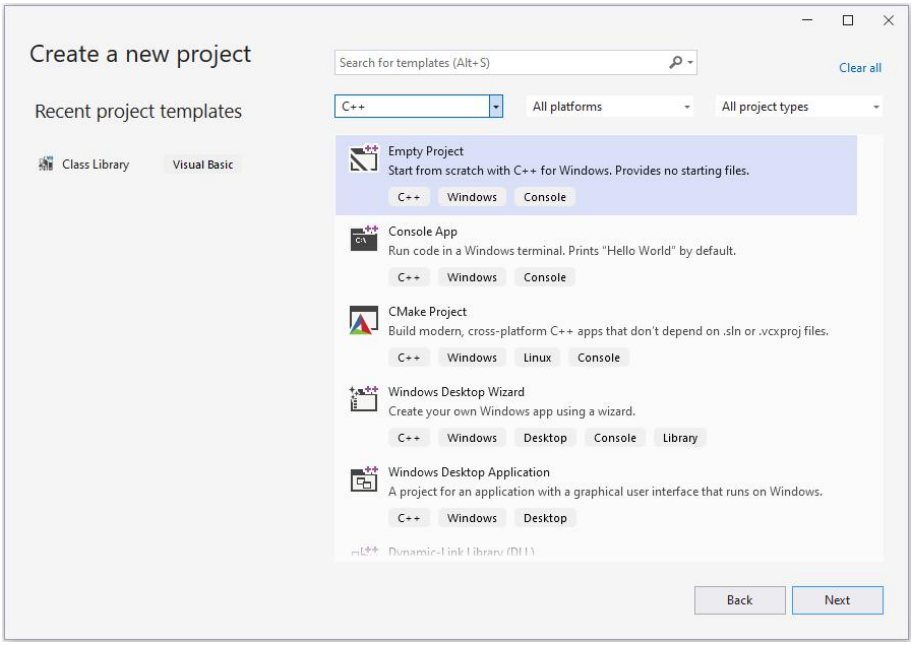
Utils: Directory contains subdirectories of GRX extended applications, including APIs for extended function development, e.g. BREP for boundary representation.

4. Programming Instruction

4.1. Create GRX Project under Win 64-Bit

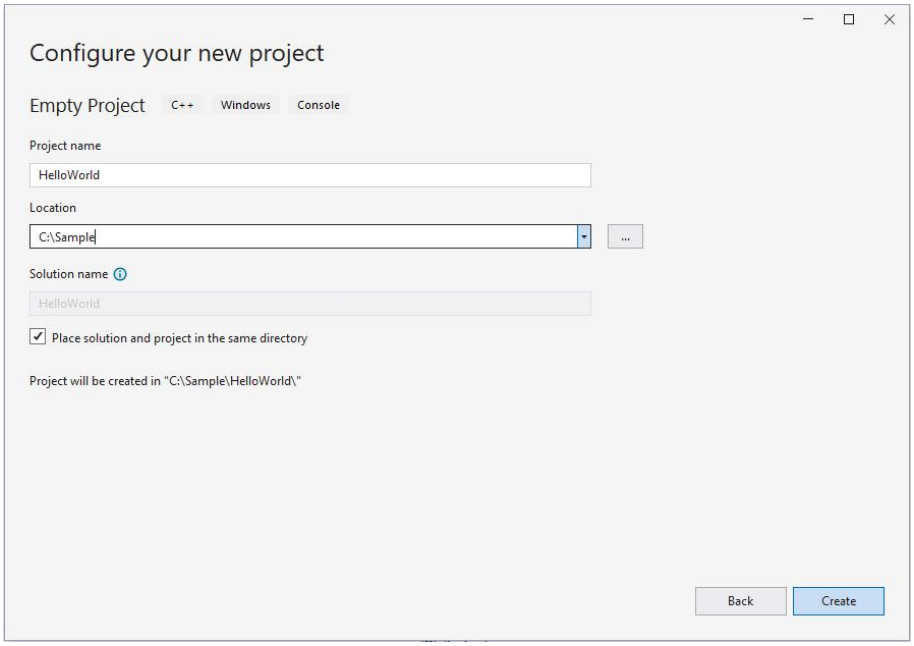
4.1.1. Run Microsoft® Visual Studio 2022

Please click **File→New→Project** to launch the **New Project** dialog window. Select **Visual C++** in the **Installed** on the left side and click **Empty Project** in the middle window.



4.1.2. Input Project Saving Path and Project Name

Input '*HelloWorld*' at the name field in the **New Project** dialog window.

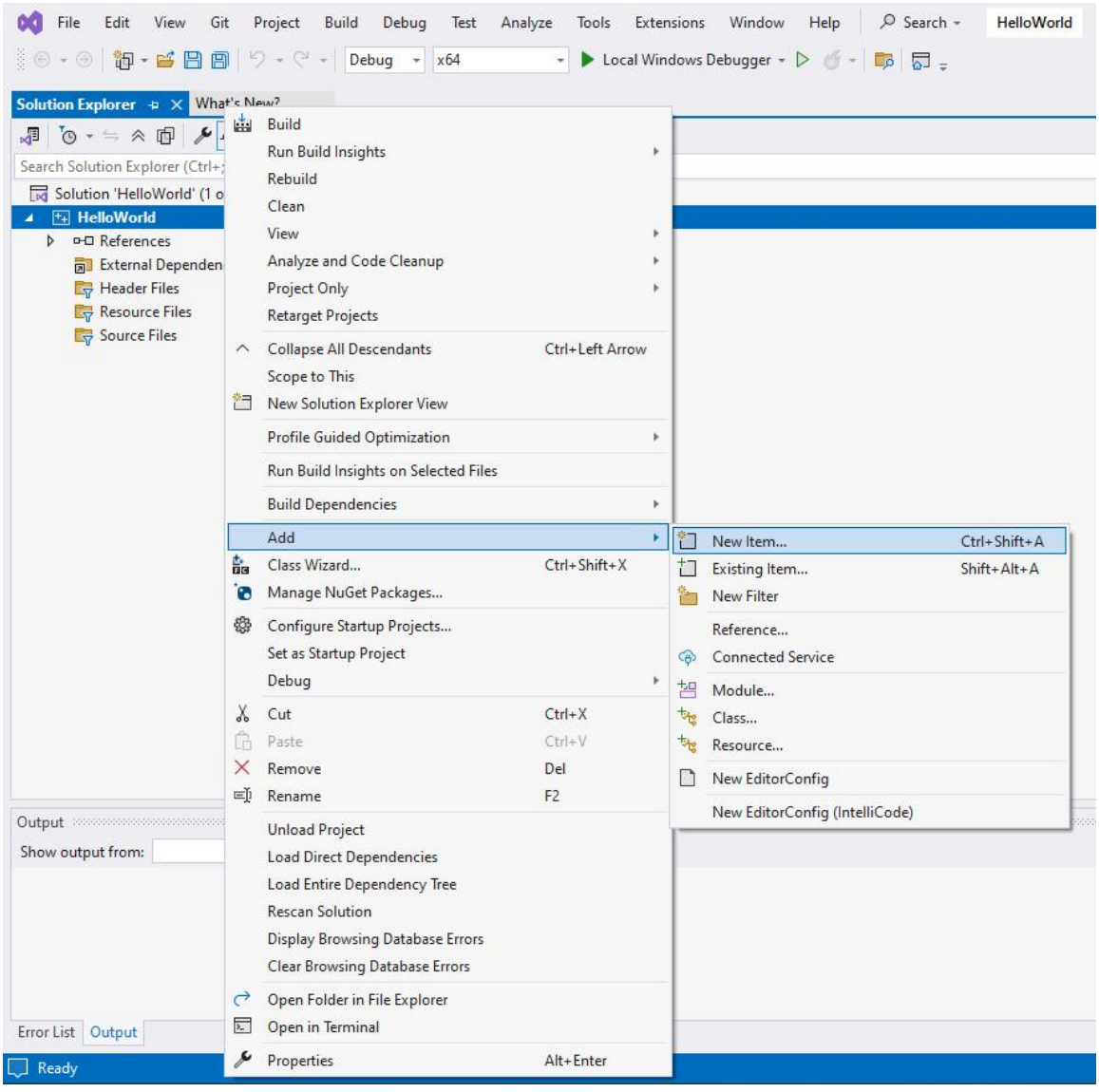


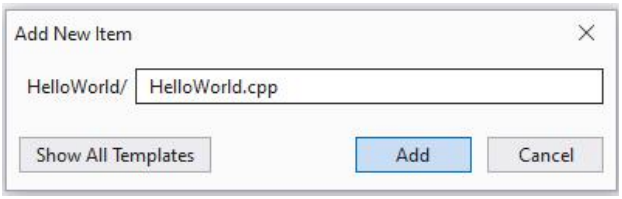
4.1.3. Finish Creating New Project

Click **Create** button to finish creating a new project.

4.1.4. Create New CPP File

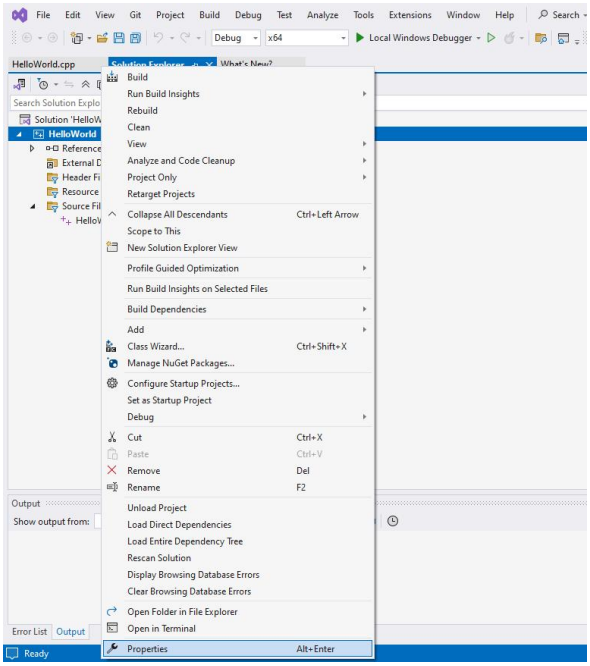
In Visual Studio 2022 dialog window, find the '*HelloWorld*' project you just created in the **Solution Explorer**. Select the '*HelloWorld*' project and right click on it, then select **Add→New Item**. After the **Add New Item - HelloWorld** dialog window pops out, name the .cpp file as HelloWord.cpp.





4.1.5. Set Compile and Link Configuration

Select the project '*HelloWorld*' in the **Solution Explorer** and select **Property** at the right click menu. After selecting the **Properties** option, the **HelloWorld Properties Pages** pops out.



1) Select **General** in **Configuration Properties** and set as below:

   Configuration Type: *Dynamic Library (.dll)*

   Windows SDK Version: *10.0(Latest installed version)*

   Platform Toolset: *Visual Studio 2022(v143)*

2) Select Advanced in **Configuration Properties** and set as below:

   Target File Extension: *.grx*

   Character Set: *Use Unicode Character Set*

3) Select **General** in **C/C++** and set as below:

   General/Additional Include Directories: *C:\grxsdk\arx\inc*

   NOTE: Under debug configuration, please delete **_DEBUG** macro definition from **C/C++** drop down list **Preprocessor -Preprocessor Definitions** and set **Code Generation/Runtime Library** as '*Multi-threaded DLL (/MD)*'.

   Language/Conformance mode: *No*

4) Select **Linker** and set as below:

   General/Additional Library Directories: *C:\grxsdk\arx\lib-x64*

   Input/Additional Dependencies:

   AecModeler.lib;gcad.lib;gcax.lib;gcbase.lib;gcbr.lib;gccore.lib;gcdb.lib;GcDbConstraints.lib;GcDbPointCloudObj.lib;gc

dyn.lib;GcGeolocationObj.lib;gcgs.lib;GcImaging.lib;GcModelDocObj.lib;gplot.lib;

Input/Module Definition File: *C:\grxsdk\arx\inc\AcRxDefault.def*

5) Click **Apply** and then click **OK** to finish the compiler and linker configuration.

6) Compile the project and make sure that the compilation is successful. Otherwise repeat the above steps to reconfigure the project settings.

4.1.6. Add Code

Add the following codes to '*HelloWorld.cpp*'.

```
#include "windows.h"
#include <tchar.h>
#include <arxHeaders.h>

void initApp();
void unloadApp();
void HelloWorld();

void initApp()
{
	acedRegCmds->addCommand(_T("HELLOWORLD_CMDS"), _T("Hello"), _T("Hello"),
ACRX_CMD_TRANSPARENT, HelloWorld);
}
void unloadApp()
{
	acedRegCmds->removeGroup(L"HELLOWORLD_CMDS");
}
void HelloWorld()
{
	acutPrintf(_T("\nHello World!"));
}

extern "C" AcRx::AppRetCode gcrxEntryPoint(AcRx::AppMsgCode msg, void *pkt)
{
	switch (msg)
```

```
        {
    case AcRx::kInitAppMsg:
            acrxDynamicLinker->unlockApplication(pkt);
            acrxDynamicLinker->registerAppMDIAware(pkt);
            initApp();
            break;
        case AcRx::kUnloadAppMsg:
            unloadApp();
            break;
        default:
            break;
        }
    return AcRx::kRetOK;
    }
```

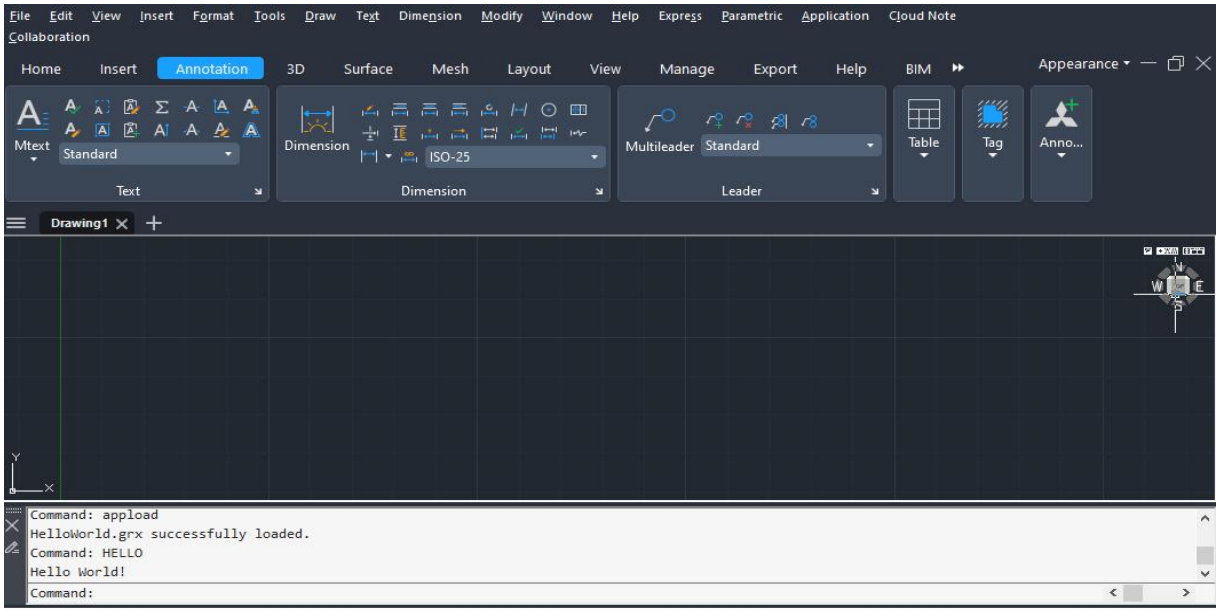
4.1.7. Compile Program

In Visual Studio 2022, click **Build→Build Solution** to create '*HelloWorld.grx*' file in '*C:\Sample\HelloWorld\x64\Debug*' directory.

4.1.8. Run Program

Run CAD and input '*appload*' at the command line, or select **Tools→Loading Applications** from menu to launch **Load/Unload Applications** dialog window. Select '*HelloWorld.grx*' and click **Load**.

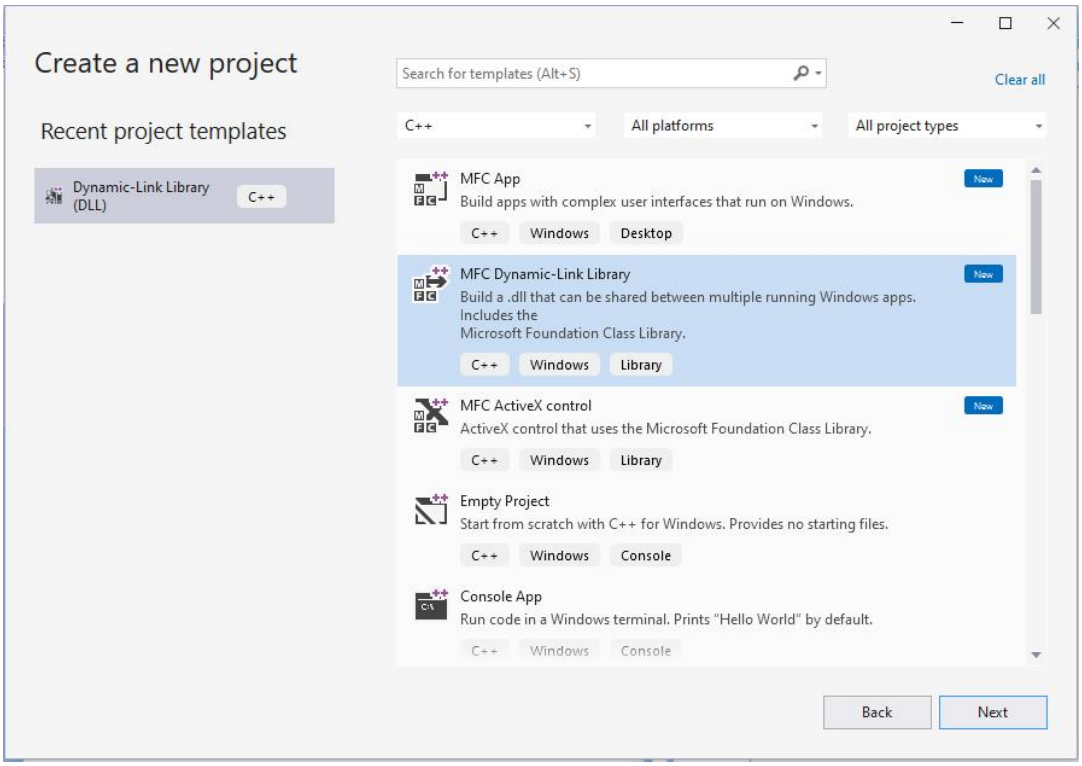
Close the **Load/Unload Applications** dialog window and input '*hello*' at command line, '*Hello World!*' will be displayed at the command line as shown below.



4.2. GRX Programming with MFC Library

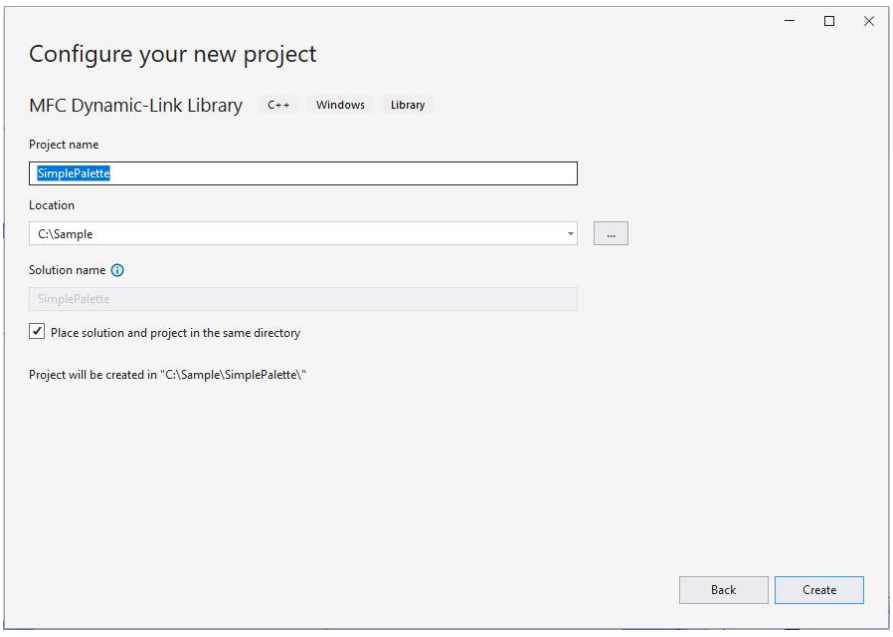
4.2.1. Run Microsoft® Visual Studio 2022

Please click **File→New→Project** to launch the **New Project** dialog window. Select **Visual C++** in the **Installed** and click **MFC Dynamic-Link Library**, then Click **Next** button.



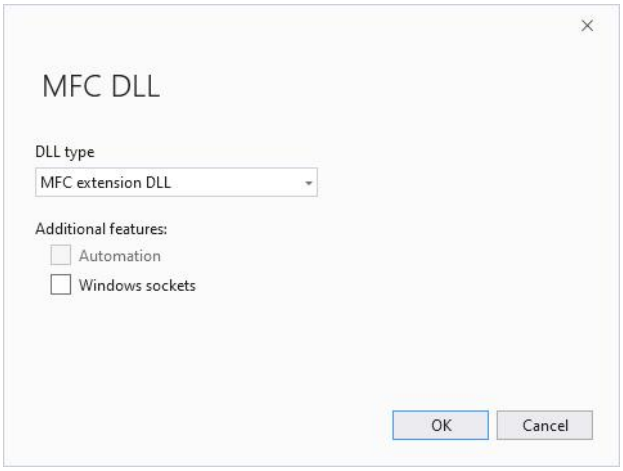
4.2.2. Input Project Saving Path and Project Name

The following sample shows how to create a '*SimplePalette*' project. Name the project as '*SimplePalette*' *and set the location* '*C:\Sample*'



4.2.3. Choose DLL Type

Click **OK**, the **MFC DLL** dialog window pops out. Select **MFC Extension DLL** and click **OK**.

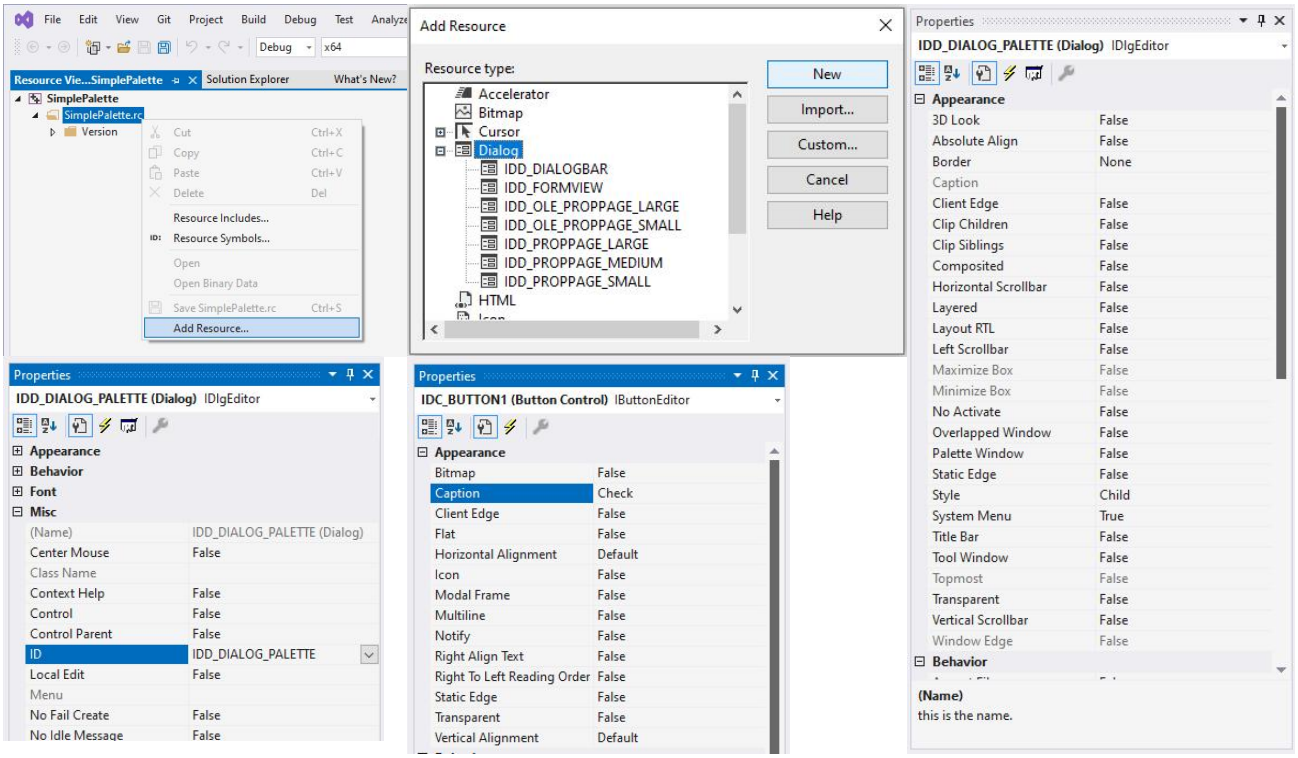


4.2.4. Finish Creating New Project

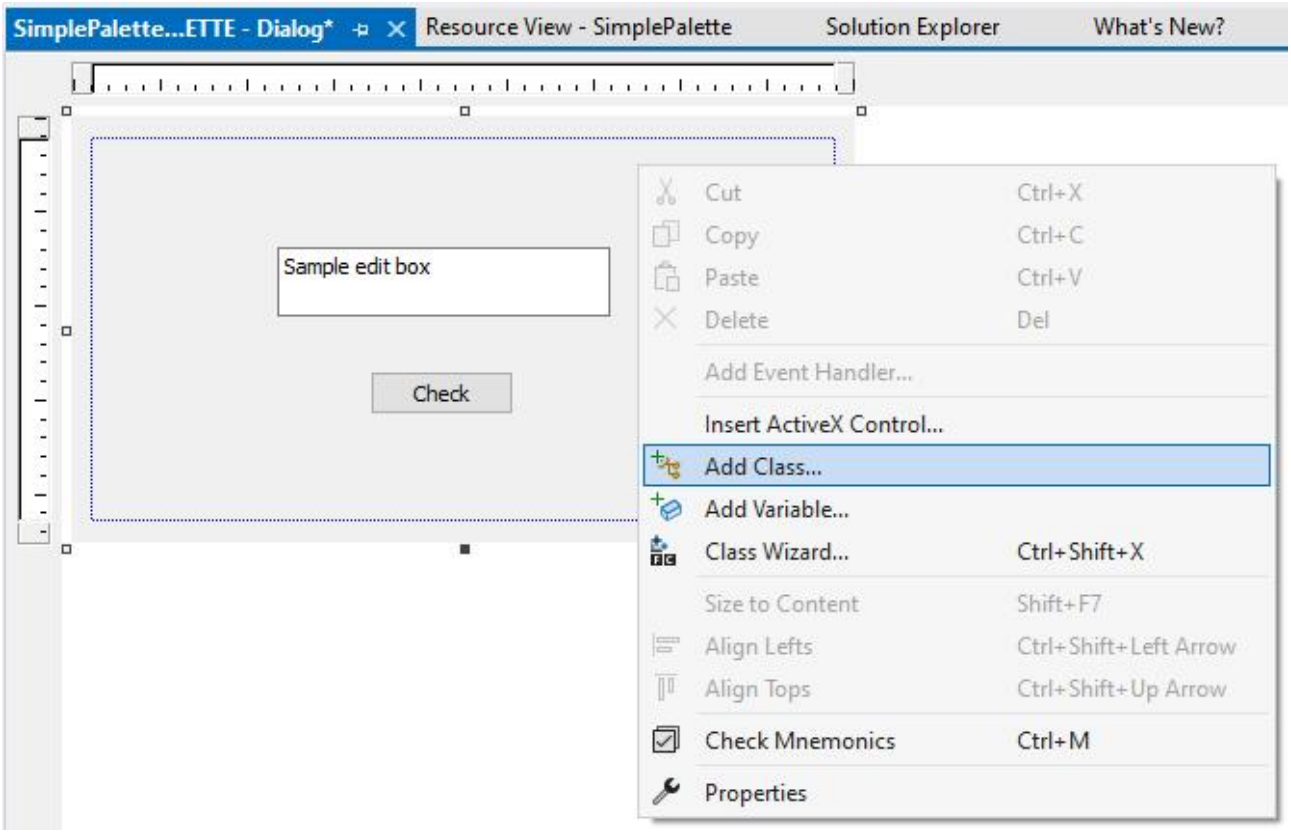
Click **OK** on **New Project** dialog window to finish creating '*SimplePalette*' project.

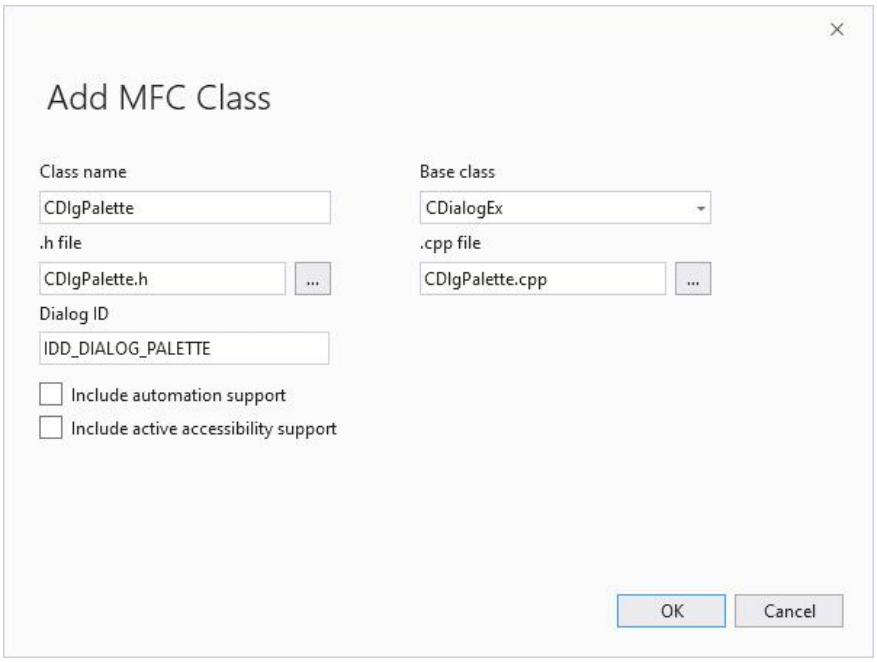
4.2.5. Create New Resource and Corresponding Class

Launch the **Resource View** of Visual Studio 2022 (Ctrl+Shift+E), right click on '*SimplePalette.rc*' and select **Add Resource...** in the drop-down menu to pop out **Add Resource** dialog window. Select **Dialog** in the **Resource Type** window and click **New** to add a new dialog. Change the **ID** of newly created '*IDD_DIALOG1*' dialog to '*IDD_DIALOG_PALETTE*'. Right click on '*IDD_DIALOG_PALETTE*' and select **Properties** to set **Border** as '*None*', **Style** as '*Child*'. Then add an **Edit Control** and a **Button** from **Toolbox** (Ctrl+Alt+X) and change the button **Caption** to '*Check*' and the **ID** to '*IDC_BUTTON_CHECK*', as shown below.



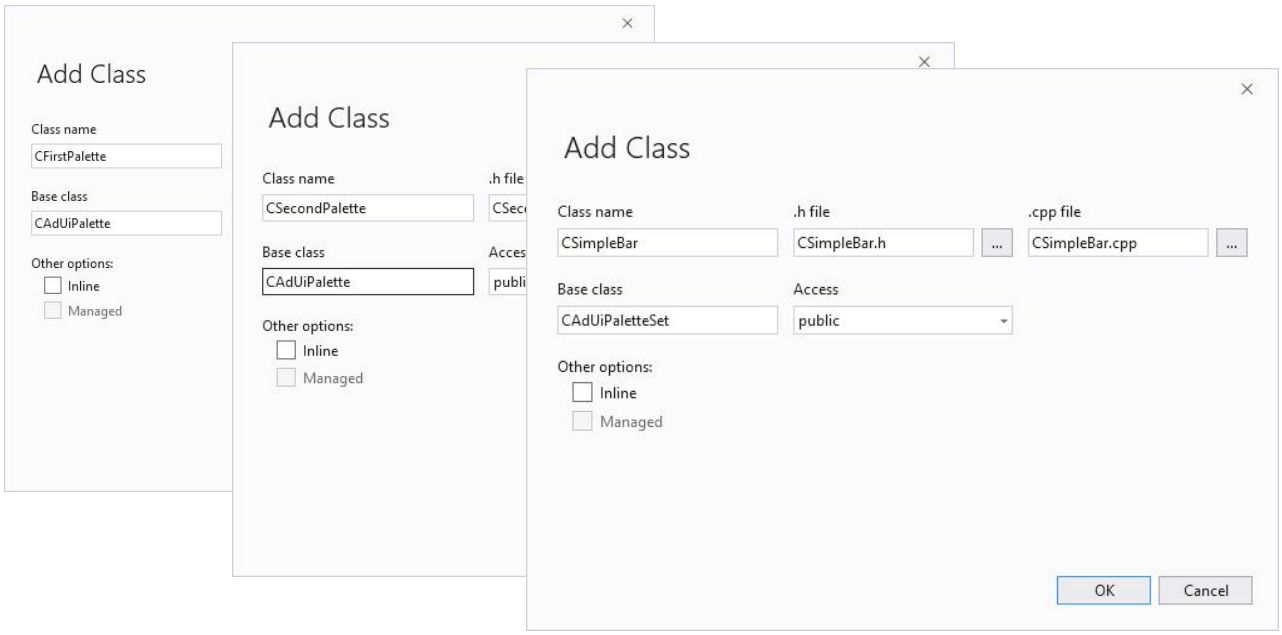
Right click on the newly created '*IDD_DIALOG_PALETTE*' dialog and select the **Add Class...** to pop out **Add MFC Class** dialog window. Set **Class Name** as '*CDlgPalette*' and click **OK**.





4.2.6. Create Three New C++ Class File

In the **Solution Explorer** of Visual Studio 2022, right click on the project '*SimplePalette*' and select **Add→Class...** to pop out **Add Class** dialog window, as shown below. Add 3 new classes named as '*CFirstPalette*', '*CSecondPalette*' and '*CSimpleBar*' and set the **Base class** as '*CAdUiPalette*', '*CAdUiPalette*' and '*CAdUiPaletteSet*' respectively.



4.2.7. Add Code

Add the following codes to the end of '*pch.h*' file:

```
#include <arxHeaders.h>
```

Add the following codes to the beginning of '*CDlgPalette.h*' Head File:

```
#include "resource.h"
```

(For the class codes of '*CDlgPalette*', '*CFirstPalette*', '*CSecondPalette*' and '*CSimpleBar*', please refer to samples (.cpp and .h files) of '*CDlgPalette*', '*CFirstPalette*', '*CSecondPalette*' and '*CSimpleBar*' in '*C:\grxsdk\arx\samples\SimplePalette*')

Add the following codes to '*SimplePalette.cpp*':

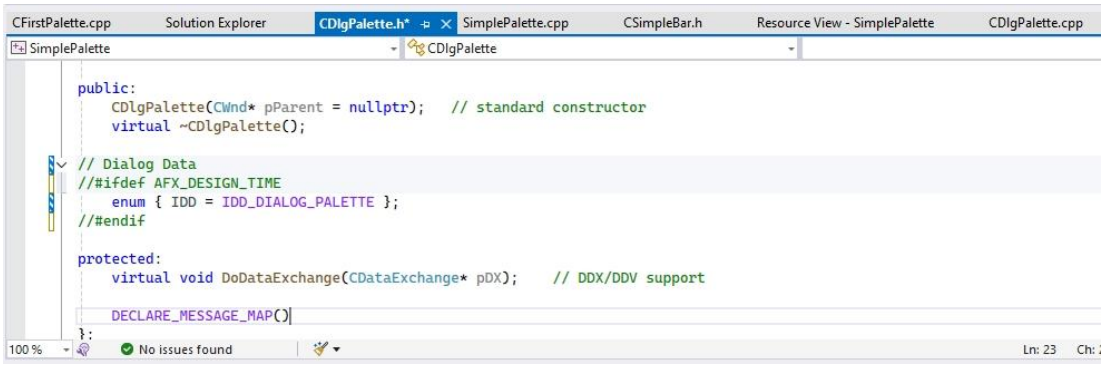
```
// SimplePalette.cpp: Defines the initialization routines for the DLL.
//

#include "pch.h"
#include "framework.h"
#include "CSimpleBar.h"


CSimpleBar* gpBar = NULL;

static void doIt()
{
	if (gpBar)
	{
		acedGetAcadFrame()->ShowControlBar(gpBar, TRUE, FALSE);
	}
	else
	{
		acedInitGet(0, L"Float Dock");
		ACHAR result[132] = { 0 };
		int rc = acedGetKword(L"\nFloat or Docking?", result);
		bool bFloat = true;
		if (rc == RTNONE)
		{
			bFloat = true;
		}
		else if (rc != RTNORM)
		{
			return;
		}
		else
		{
			bFloat = _wcsicmp(result, L"Float") == 0;
		}

		CSize sizeDefault(400, 450);
		CRect rcDefault(CPoint(150, 200), sizeDefault);

		gpBar = new CSimpleBar;

		DWORD dwStyle = WS_CHILD | WS_VISIBLE | CBRS_SIZE_DYNAMIC | CBRS_GRIPPER;
		if (bFloat)
```

```
            {
                dwStyle |= CBRS_FLOATING;
            }
            else
            {
                dwStyle |= CBRS_LEFT;
            }

            gpBar->Create(L"Simple", dwStyle, rcDefault, acedGetAcadFrame());
            gpBar->SetAutoRollup(FALSE);
            gpBar->EnableDocking(CBRS_ALIGN_LEFT | CBRS_ALIGN_RIGHT |
CBRS_ALIGN_BOTTOM);
            if (bFloat)
            {
                acedGetAcadFrame()->FloatControlBar(gpBar, CPoint(100, 100), CBRS_ALIGN_TOP);
            }
            else
            {
                acedGetAcadFrame()->DockControlBar(gpBar, AFX_IDW_DOCKBAR_RIGHT);
            }
        }
}

void initApp()
{
        acedRegCmds->addCommand(_T("ASDK_SAMPLES_SIMPLEPALETTE"),
            _T("ASDK_SIMPLEPALETTE"), _T("SIMPLEPALETTE"), ACRX_CMD_MODAL,
            dolt);
}


void unloadApp()
{
        acedRegCmds->removeGroup(_T("ASDK_SAMPLES_SIMPLEPALETTE"));

        if (gpBar)
        {
            gpBar->DestroyWindow();
            delete gpBar;
            gpBar = NULL;
        }
}
```

```
extern "C" AcRx::AppRetCode
acrxEntryPoint(AcRx::AppMsgCode msg, void* appId)
{
	switch (msg) {
	case AcRx::kInitAppMsg:
		acrxDynamicLinker->unlockApplication(appId);
		acrxDynamicLinker->registerAppMDIAware(appId);
		initApp();
		break;
	case AcRx::kUnloadAppMsg:
		unloadApp();
	}
	return AcRx::kRetOK;
}
```

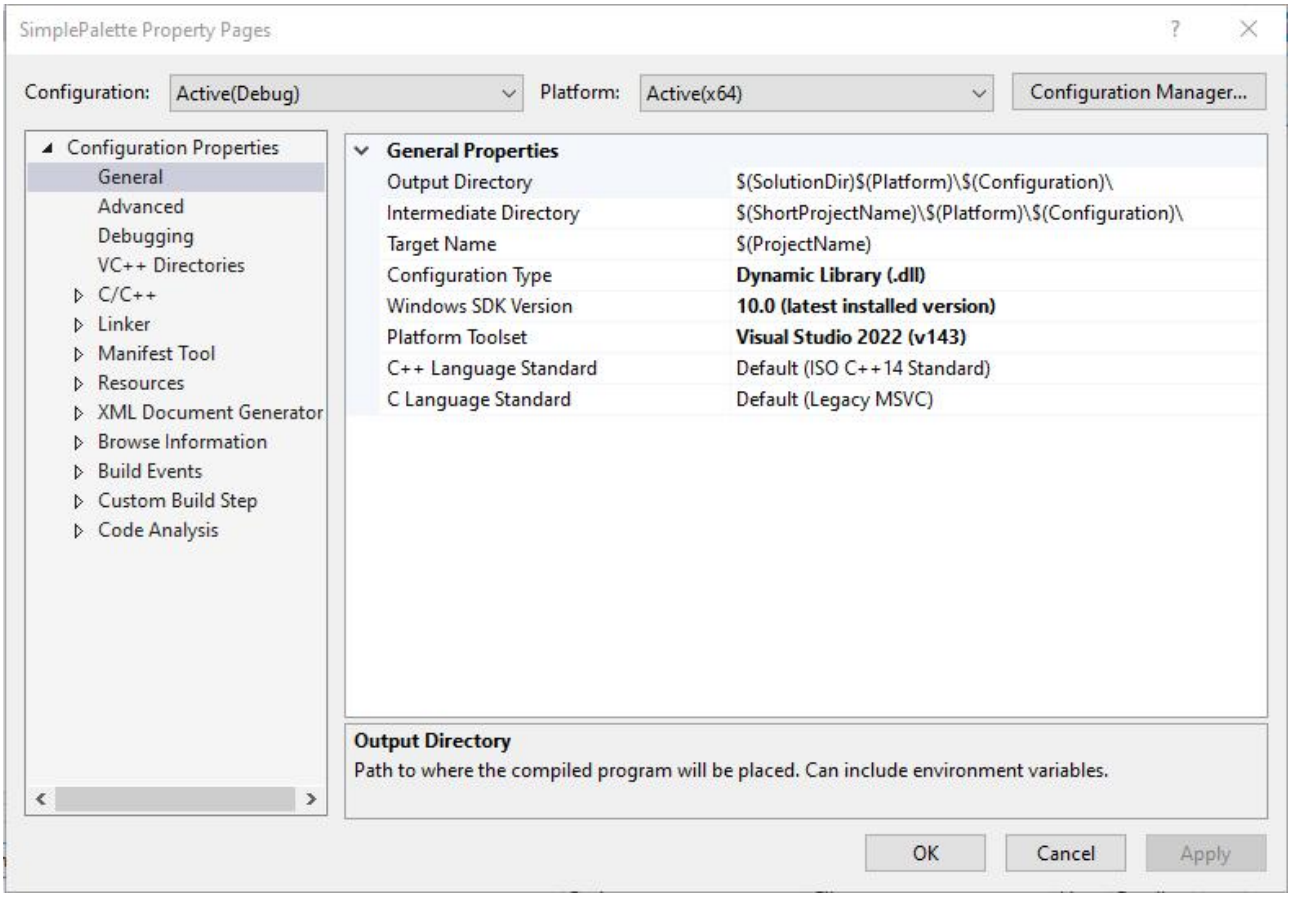
Note : Please delete the following code in CDlgPalette.h

```
//#ifdef AFX_DESIGN_TIME

	//#endif
```



4.2.8. Set Compile and Link Configuration

1) In the **Solution Explorer**, right click the project '*SimplePalette*' and select **Properties** to pop out the **SimplePalette Properties Pages** dialog window. The following settings in this section are completed in this dialog window.



2) Select **General** in **Configuration Properties** and set as below:

   Configuration Type: *Dynamic Library (.dll)*

   Windows SDK Version: *10.0(Latest installed version)*

   Platform Toolset: *Visual Studio 2022(v143)*

3) Select Advanced in **Configuration Properties** and set as below:

   Target File Extension: *.grx*

   Character Set: *Use Unicode Character Set*

4) Select **General** in **C/C++** and set as below:

   General-Additional Include Directories: *C:\grxsdk\arx\inc*

   NOTE: Under debug configuration, please delete **_DEBUG** macro definition from **C/C++** drop down list **Preprocessor -Preprocessor Definitions** and set **Code Generation/Runtime Library** as '*Multi-threaded DLL (/MD)*'.

   Language-Conformance mode: *No*

5) Select **Linker** and set as below:

General/Additional Library Directories: *C:\grxsdk\arx\lib-x64*

Input/Additional Dependencies:

AecModeler.lib;gcad.lib;gcax.lib;gcbase.lib;gcbr.lib;gccore.lib;gcdb.lib;GcDbConstraints.lib;GcDbPointCloudObj.lib;gcdyn.lib;GcGeolocationObj.lib;gcgs.lib;GcImaging.lib;GcModelDocObj.lib;gplot.lib;

Input/Module Definition File: *C:\grxsdk\arx\inc\AcRxDefault.def*

Advanced/Target Machine: *Machine X64*

6) Click **Apply** and then click **OK** to finish the compiler and linker configuration.

7) Compile the project and make sure that the compilation is successful. Otherwise repeat the above steps to reconfigure the project settings.

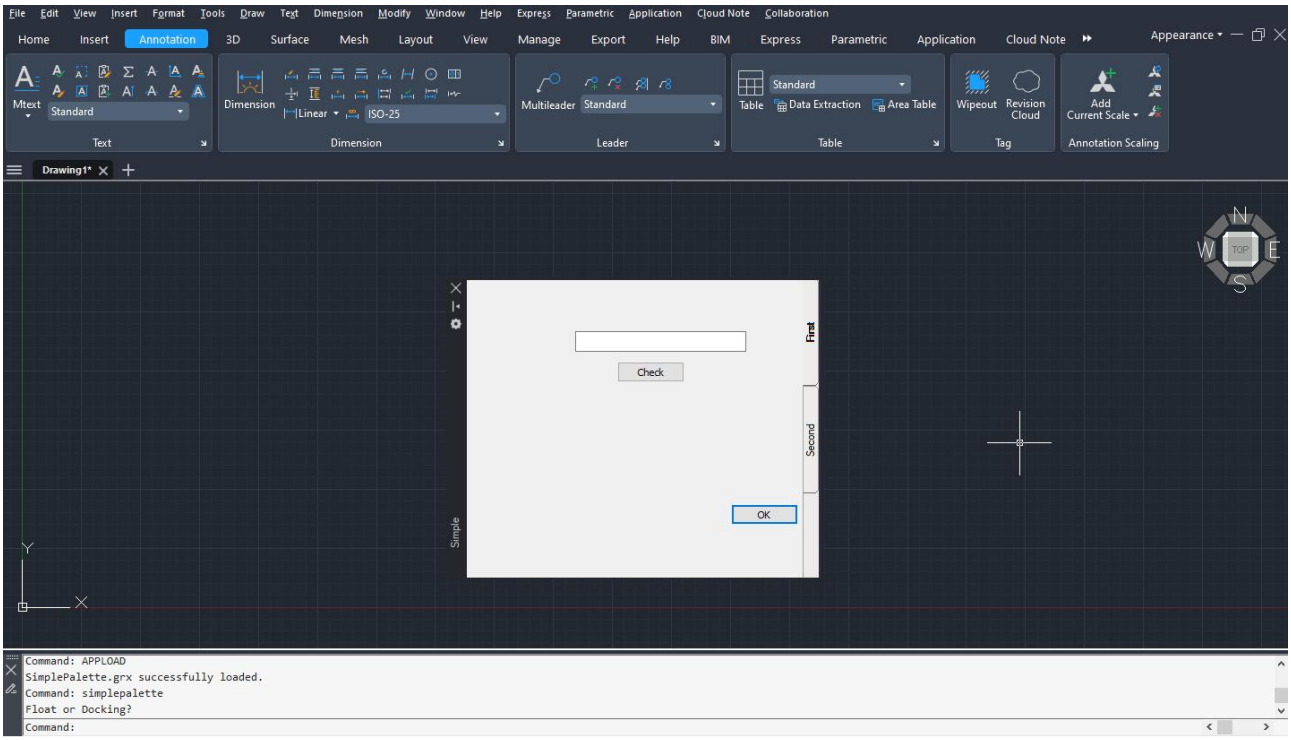
4.2.9. Compile Program

In Visual Studio 2022, click **Build→Build Solution** to create '*SimplePalette.grx*' file in the '*C:\Sample\SimplePalette\x64\Debug*' directory.

4.2.10. Run Program

Run CAD and input '*appload*' at the command line, or select **Tools→Loading Applications** from menu to launch **Load/Unload Applications** dialog window. Select '*SimplePalette.grx*' and click **Load**.

Close the **Load/Unload Applications** dialog window and input '*simplepalette*' at command line, '*Float or Docking?*' will be displayed at the command line. Press **Enter** key and a dialog window will pop out as shown below.



5. Description of GRX Class Library

The following libraries are frequently used in programming with GRX. These libraries have same functions with the corresponding ARX libraries.

- **GcRx**: same as **AcRX** of **ARX**, classes for binding an application and for runtime class registration and identification
- **GcEd**: same as **AcEd** of **ARX**, classes for registering commands and for event notification
- **GcDb**: same as **AcDb** of **ARX**, CAD database class
- **GcGi**: same as **AcGi** of **ARX**, graphics classes for rendering entities
- **GcGe**: same as **AcGe** of **ARX**, utility classes for common linear algebra and geometric objects

5.1. GcRx

The **GcRx** class library provides system-level functionality such as DLL initialization and linking, and runtime class registration and identification as below:

- Object runtime class identification and inheritance analysis
- Runtime addition of new protocol to an existing class
- Object equality and comparison testing
- Object copy

5.2. GcEd

The **GcEd** class library is used to define and register new commands which operate in the same manner as the original ones.

5.3. GcDb

The **GcDb** classes are components of the database. This database stores all the information for the graphical objects that compose a drawing, and the non-graphical objects (such as layers, linetypes, and text styles) that are also part of a drawing.

5.4. GcGi

The **GcGi** class library provides the graphic interface used for drawing CAD entities.

5.5. GcGe

The **GcGe** class library provides classes used for performing common 2D and 3D geometric operations, including utility classes such as vectors and matrices and basic geometric objects such as points, curves, and surfaces.

6. About the use of macro OBJECTPARAM_ALLOWED

In order to be compatible with ObjectARX2025, in GRX, if GcString is passed into a variable parameter function such as gcutPrintf , it is necessary to call GcString :: constPtr (); as follows:

Previous version:

GcString sz ;

gcutPrintf (_T(" \ n%s "), sz);

Current Edition

GcString sz ;

gcutPrintf (_T(" \ n%s "), sz.constPtr ());

If this modification is not made (i.e. constPtr () is not called), a compilation error will be reported when GRX is compiled:

error C4840: non-portable use of class " GcString " as argument to variadic function

Developers can use this compilation error to modify the code where GcString is called in previous versions.

This compatibility change will result in a compilation error when CStirng is passed into a variable parameter function such as gcutPrintf :

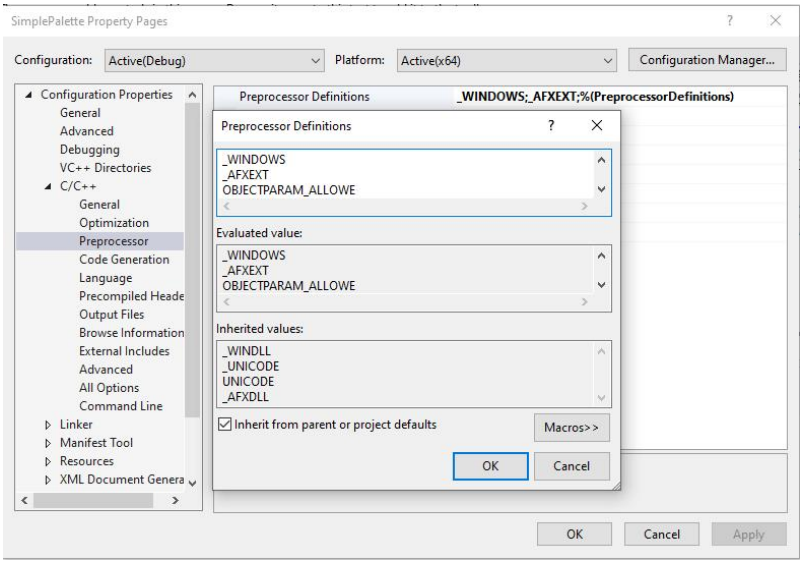
error C4840: non-portable use of class "ATL::CStringT<wchar_t,StrTraitMFC_DLL<wchar_t,ATL::ChTraitsCRT<wchar_t>>>" as argument to a variadic function

At this point, you can choose one of the following two methods to modify it:

Method 1: Change to

gcutPrintf (_T("%s"), (LPCTSTR (sz)));

Method 2: After modifying all GcString codes , add **OBJECTPARAM_ALLOWED definition** in the project configuration . This macro definition can ignore all such compilation errors. As shown below.



7. Copyright