Creating AutoCAD Cross-Platform Plug-ins

Fernando Malard

Chief Technology Officer – ofcdesk, LLC.

@fpmalard

Class summary

This class will present strategies used to create cross-platform plug-ins, taking advantage of C++ language in order to create core source code that could be used by AutoCAD software for Windows and by AutoCAD for Mac software. We will then demonstrate how to consume this shared core code into each platform, taking advantage of each specific user interface feature (Microsoft MFC, Microsoft .NET inside Windows, and Cocoa inside Mac OSX).

Key learning objectives

At the end of this class, you will be able to:

- Learn how to create C++ cross-platform core
- Learn how to create code with ObjectARX technology for Windows
- Learn how to create code with ObjectARX technology for Mac
- Learn how to use Microsoft Visual Studio and Mac OSX Xcode

Introduction

Introduction

- AutoCAD support for Mac resumed in **2011**
- Totally redesigned UI following **Mac OSX concepts**
- Partial **Windows** and **MFC** libraries support
- **Xcode** is the **Visual Studio** equivalent IDE
- Shared **C++** code is possible
- Different UI approaches with **Cocoa**

Introduction
Requirements

- **AutoCAD** 2016
- **ObjectARX** 2016
- **Visual Studio** 2012
- **Xcode** 6.4 (7.1 can be used with prompt build and 6.4)
- **Apple Hardware** to run AutoCAD for Mac
- **Virtual Machine** (Parallels Desktop or VMWare Fusion)

Shared C++ Core

Shared C++ Core

- AutoCAD for **Windows is 32/64-bit**
- AutoCAD for **Mac is 64-bit** only
- **MVC** (Model View Controller) pattern recommended to separate **UI** (View) from the **data** (Model)
- Use of **.mm** files (can mix C++ with Objective-C code)
- **DBX** code can be easily recompiled across platforms
- **ARX** code will require more work

Shared C++ Core
Platform API Comparison

API / Library	Windows	Mac OSX
ObjectARX C++	Ok	Ok
Objective-C / Cocoa	Not supported	Ok
Win32	Ok	Partial
MFC	Ok	Limited support
ActiveX / COM	Ok	Not supported
.NET	Ok	Not supported
LISP	Ok	Ok
DCL	Ok	Not supported
Qt	Ok	Ok
Javascript	Ok	Not yet

Shared C++ Core
Polymorphic Types

- Can accommodate **32/64-bit** numbers and strings
- Will **simplify** the code
- Included into the ***adesk.h*** header file
- Don’t assume **sizeof(int)** is equal to **sizeof(long)**
- Use **Adesk::Int32** and **Adesk:UInt32**, etc.
- Use **ACHAR*** and **AcString** for strings
- You can also use ***_ADESK_MAC_*** and ***_ADESK_WINDOWS_*** definitions

Creating Windows Project

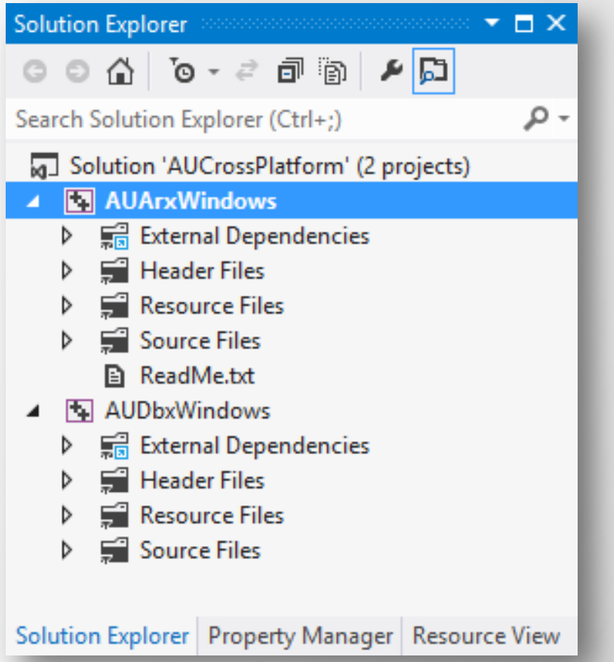
Create Windows Project

- **Visual Studio 2013/2015** can be used with 2012 also installed
- Use **Platform Toolset** feature in 2013/2015 so it will build the code using Visual Studio 2012 engine
- Use ObjectARX Wizard for both **DBX** and **ARX** modules:

  http://images.autodesk.com/adsk/files/ObjectARXWizards-2016.zip
- Create **Debug/Release** compilation modes
- Adjust project output folders if necessary

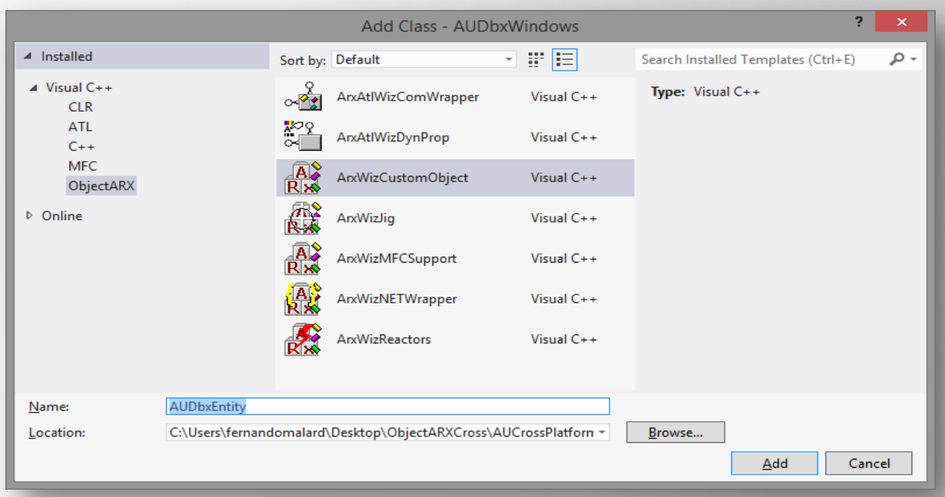
Create Windows Project
Creating the DBX Custom Entity

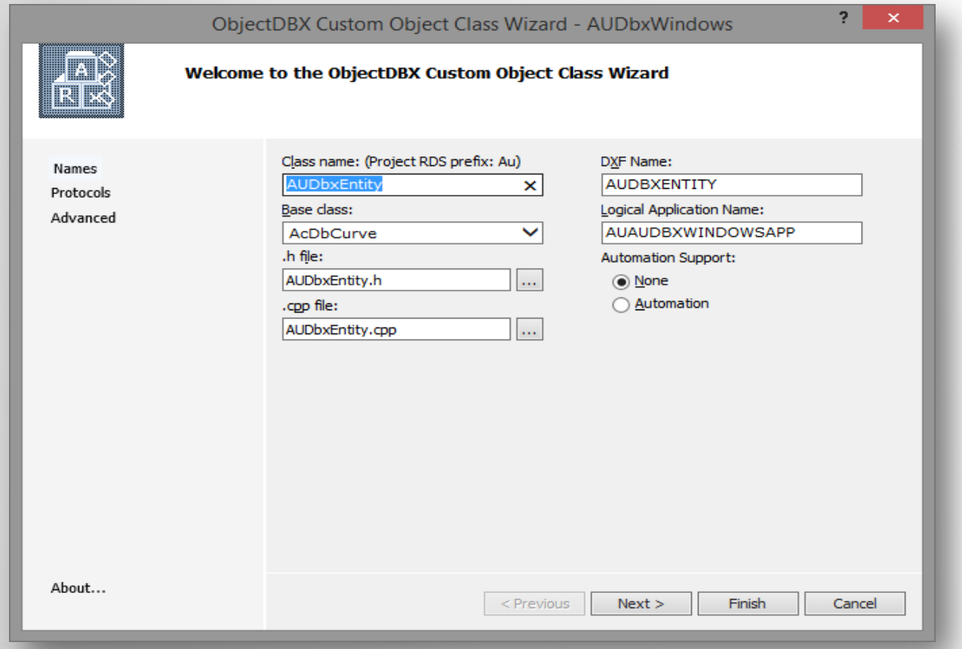
- Bind **ARX** project to DBX
- Add **DBX** project **library**:
  $(SolutionDir)\AUDbxWindows\$(PlatformTarget)\$(Configuration)*.lib
- Make sure you select all **Platforms** and all **Configurations**

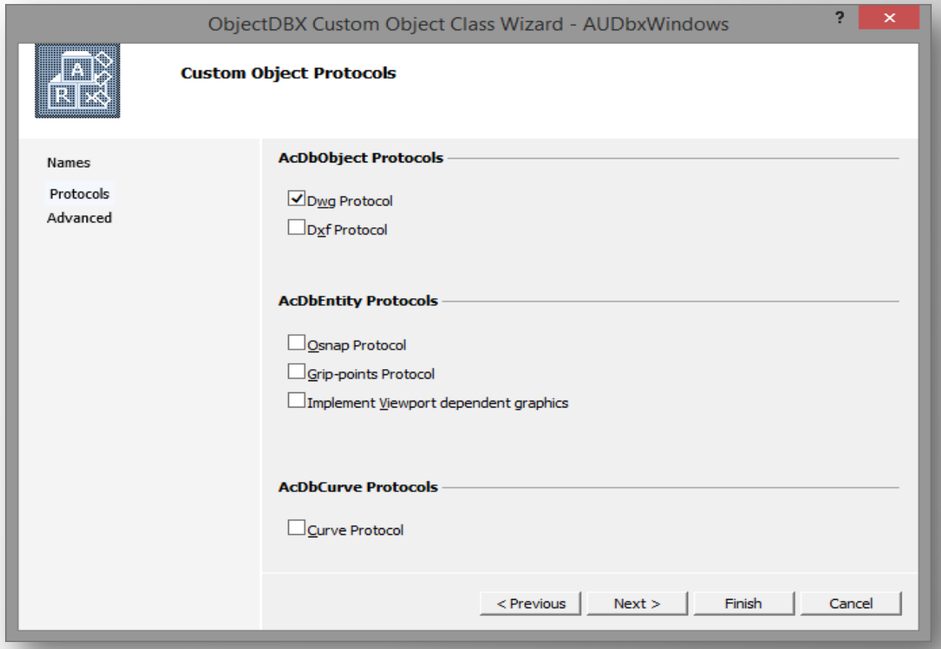


Create Windows Project
Creating the DBX Custom Entity

- Using **ClassWizard**, add a new **Custom Entity** class







Create Windows Project
Creating the DBX Custom Entity

- Add **variables** and access **methods**
- Use appropriate **assert**:
  assertReadEnabled() / assertWriteEnabled()

```
Adesk::Int16   _iTemperature;
AcString       _sText;
AcGePoint3d    _pts[2];
```

```
void SetTemperature(Adesk::Int16 iTemperature);
Adesk::Int16 GetTemperature() const;

void SetText(const ACHAR* sText);
const ACHAR* GetText() const;

void SetStartPoint(AcGePoint3d pt);
AcGePoint3d GetStartPoint() const;

void SetEndPoint(AcGePoint3d pt);
AcGePoint3d GetEndPoint() const;
```

Create Windows Project
Creating the DBX Custom Entity

- **Save** and **Read** variables through **DWG In/Out** methods
- Use the appropriate **filer** override method to read or write variables

```
// AUDbxEntity::dwgOutFields
pFiler->writeInt16(_iTemperature);
pFiler->writeString(_sText);
pFiler->writePoint3d(_pts[0]);
pFiler->writePoint3d(_pts[1]);

// AUDbxEntity::dwgInFields
pFiler->readInt16(&_iTemperature);
pFiler->readString(_sText);
pFiler->readPoint3d(&_pts[0]);
pFiler->readPoint3d(&_pts[1]);
```

Create Windows Project
Creating the DBX Custom Entity

- Make **subWorldDraw**() return **Adesk::kFalse** and implement **subViewportDraw**()
- Drawing will be a **line** in **RED** with a **text** aligned in **GREEN**
- The text value will be the **prefix** followed by the **temperature**

```
void AUDbxEntity::subViewportDraw (AcGiViewportDraw *mode) {
        assertReadEnabled () ;

        AcDbPolyline plBuff;
        plBuff.addVertexAt(0,_pts[0].convert2d(AcGePlane::kXYPlane));
        plBuff.addVertexAt(1,_pts[1].convert2d(AcGePlane::kXYPlane));
        mode->subEntityTraits().setColor(1);
        mode->geometry().pline(plBuff);

        AcString strMsg;
        strMsg.format(_T("%s > %d"),_sText.constPtr(),_iTemperature);
        AcGeVector3d vDir = _pts[1] - _pts[0];
        AcGeVector3d vecPerp = vDir.perpVector().normalize();
        double dLen = vDir.length();

        mode->subEntityTraits().setColor(3);
        mode->geometry().text(_pts[0] + vecPerp*(dLen / 20.0),
           AcGeVector3d::kZAxis,vDir,(dLen / 10.0),1.0,0,strMsg);
}
```

Create Windows Project
Adding the Custom Entity to Model Space

- Into ARX **entrypoint** file, Instantiate a new entity
- Call the variable **set methods**
- Open **BlockTable**
- Open **ModelSpace**
- **Append** the entity
- **Close** pointers

```
// Entity util
static void CreateCustomEntity(ads_point pti, ads_point ptj, ACHAR* pText, int iTemp)
{
	AUDbxEntity* pEntity = new AUDbxEntity();

	pEntity->SetStartPoint(asPnt3d(pti));
	pEntity->SetEndPoint(asPnt3d(ptj));
	pEntity->SetText(pText);
	pEntity->SetTemperature(iTemp);

	AcDbBlockTable *pBlockTable = NULL;
	acdbHostApplicationServices()->workingDatabase()
	   ->getSymbolTable(pBlockTable, AcDb::kForRead);

	AcDbBlockTableRecord *pBlockTableRecord = NULL;
	pBlockTable->getAt(ACDB_MODEL_SPACE, pBlockTableRecord, AcDb::kForWrite);
	pBlockTable->close();

	AcDbObjectId entId;
	pBlockTableRecord->appendAcDbEntity(entId, pEntity);
	pEntity->close();
	pBlockTableRecord->close();
}
```

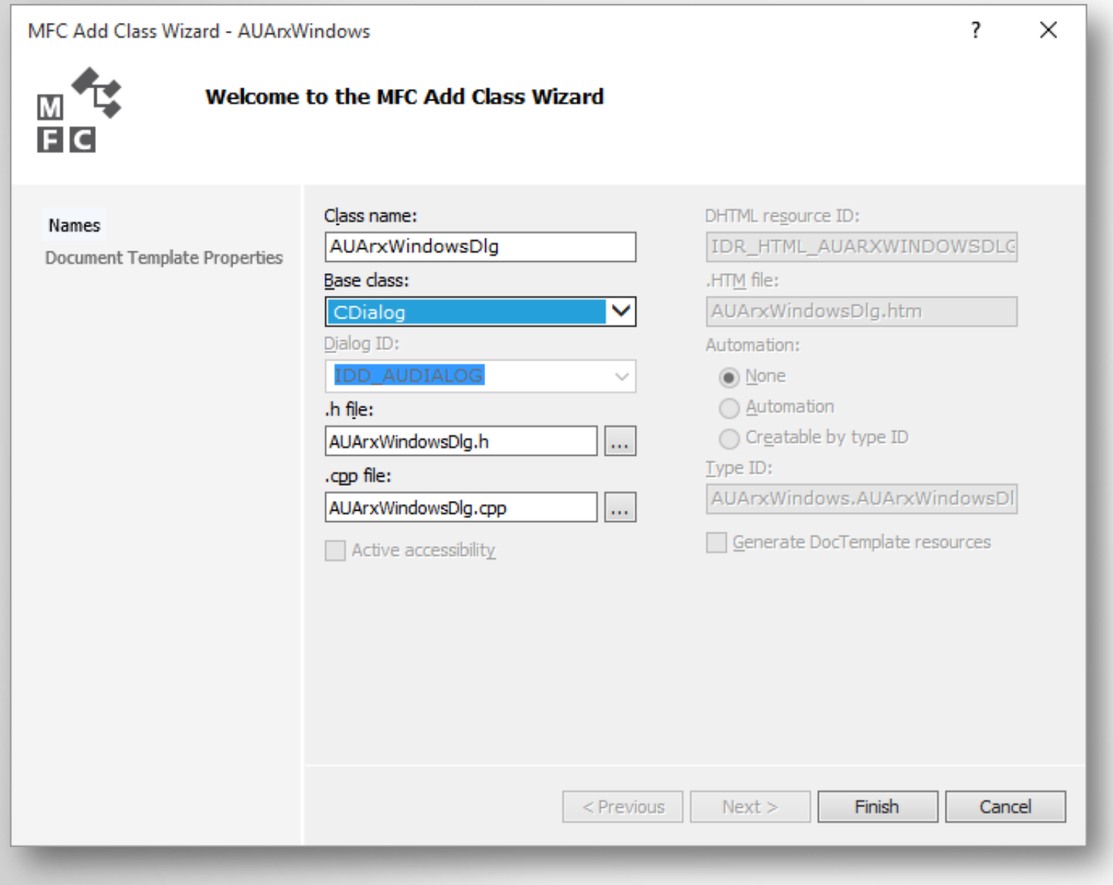
Create Windows Project
Creating basic command prompt interaction

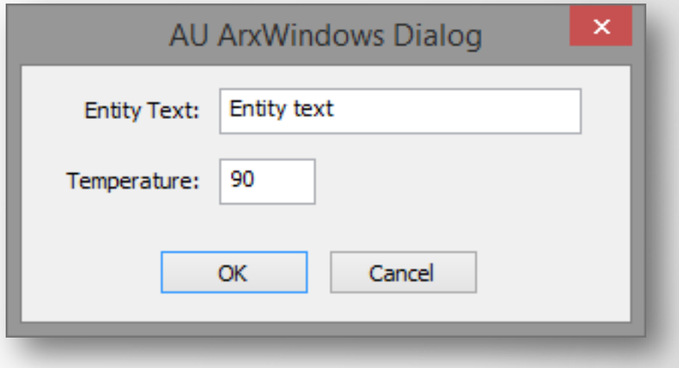
- ARX module will refer to DBX Custom Entity:
  #include "..\AUDbxWindows\AUDbxEntity.h"

```
// Prompt version
static void AuMyGroupAUCROSSPROMPT () {

	ads_point pti, ptj;
	if (acedGetPoint(NULL, _T("\nClick at the start point:"), pti) != RTNORM)
		return;

	if (acedGetPoint(pti, _T("\nClick at the end point:"), ptj) != RTNORM)
		return;

	ACHAR pText[255] = _T("");
	if (acedGetString(1,_T("\nEnter the Text Message:"),pText) != RTNORM)
		return;

	int iTemp = 0;
	if (acedGetInt(_T("\nEnter the Temperature:"),&iTemp) != RTNORM)
		return;

	CreateCustomEntity(pti, ptj, pText, iTemp);
}
```

Create Windows Project
Creating the Windows UI with MFC

- Add a new Resource then a MFC dialog class





Create Windows Project
Creating the Windows UI with MFC

- The MFC dialog will return the input data to create the Custom Entity

```
// Dialog version
static void AuMyGroupAUCROSSDLG() {
	AUArxWindowsDlg dlg(CWnd::FromHandle(adsw_acadMainWnd()));

	if (dlg.DoModal() == IDOK)
	{
	   ads_point pti, ptj;
	   if (acedGetPoint(NULL, _T("\nClick at the start point:"), pti) != RTNORM)
	       return;

	   if (acedGetPoint(pti, _T("\nClick at the end point:"), ptj) != RTNORM)
	       return;

	   CreateCustomEntity(pti, ptj, dlg._sText.GetBuffer(), dlg._iTemp);
	}
}
```

Creating Mac OSX Project

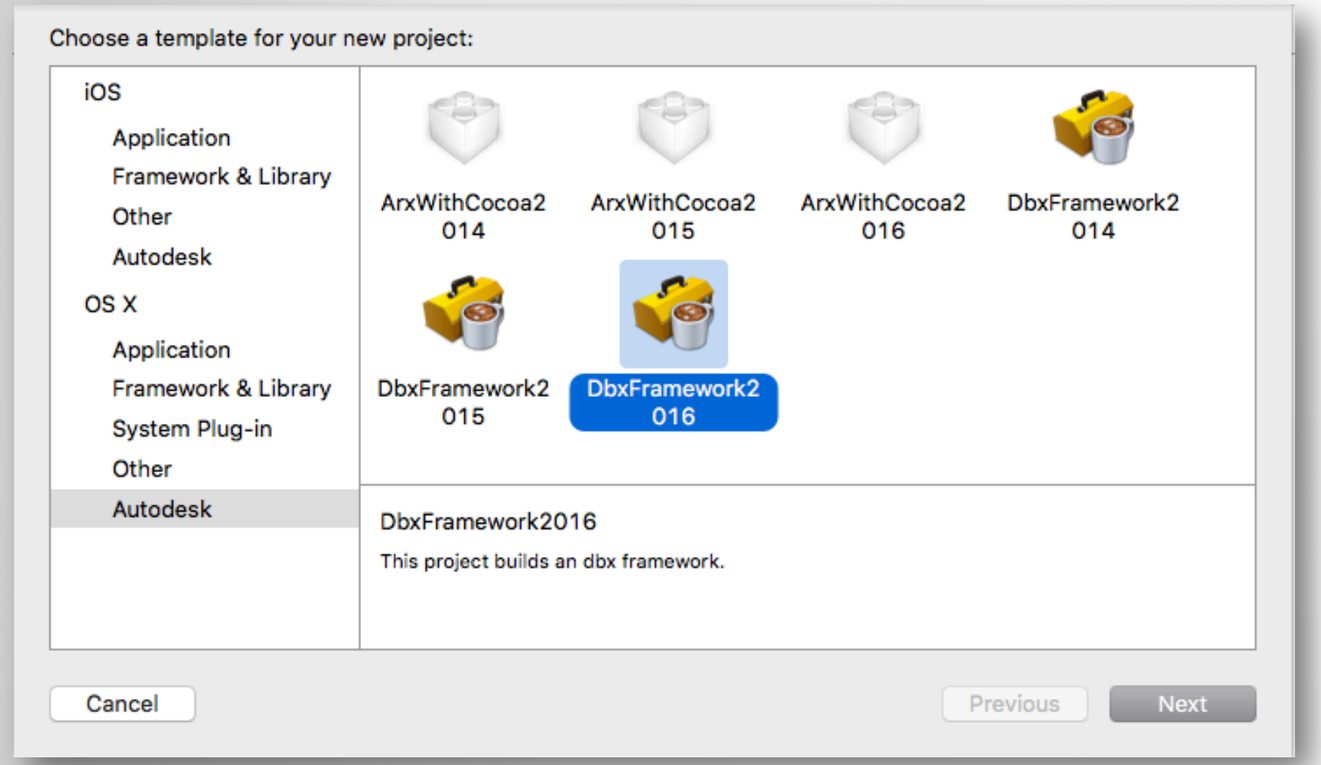
Creating MacOSX Project

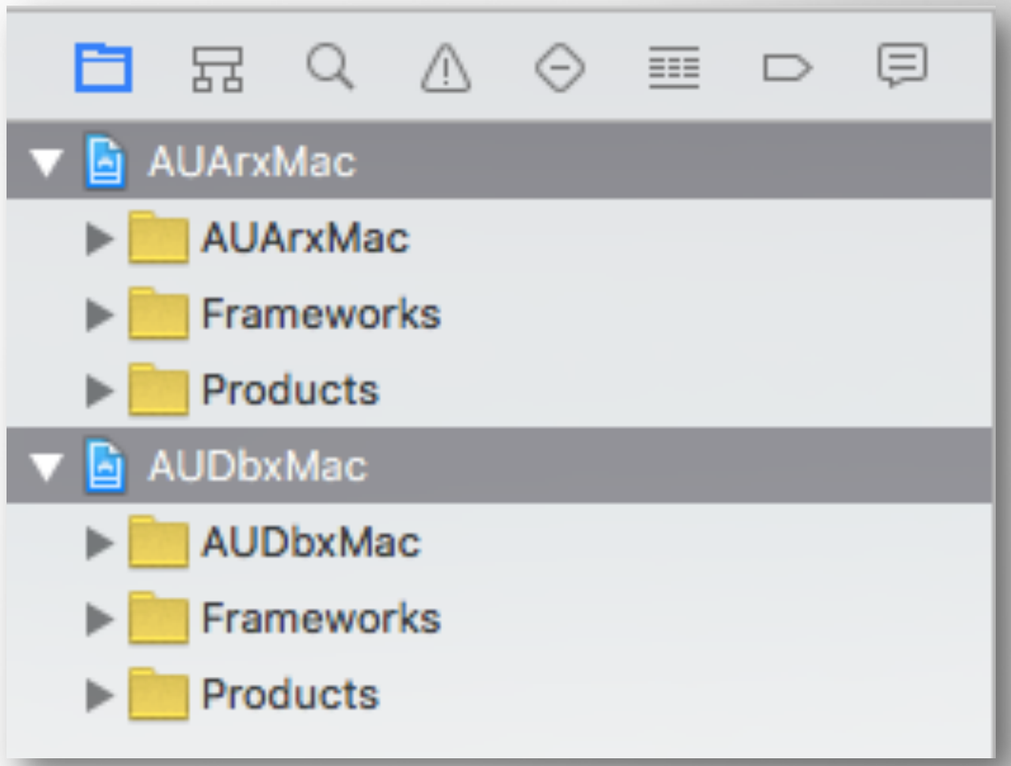
- Apple Xcode is the IDE for creating AutoCAD plugins inside MacOSX

Feature	Visual Studio	Xcode
Integrated Environment	Solution (.sln)	Workspace (.xcworkspace)
Project	VC++ Project (.vcxproj)	Xcode Project (.xcodeproj)
Configuration	Solution Configuration	Schemes
Output Files	.arx / .dbx / .crx	.bundle / .dbx
Project Configuration File	.props	.xcconfig
Platform support	Win32 / x64	x86_64
Resources / Dialogs	.rc / MFC	.xib / Cocoa
C++ source file	.CPP	.CPP / .MM
Resource editor	Resource View	Interface Builder
Application Design Pattern	any pattern	MVC by default

Creating MacOSX Project
Creating the Xcode projects

- Modules are dynamic libraries but ARX is a **Bundle**





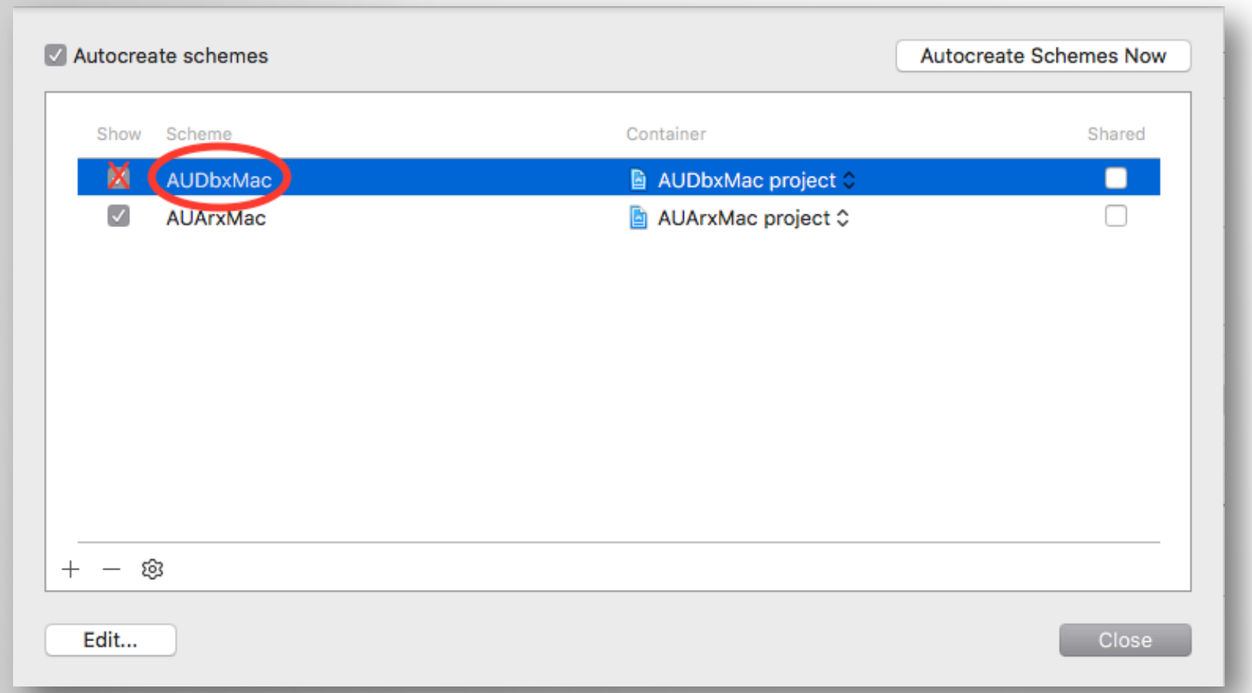
Creating MacOSX Project
Install Path

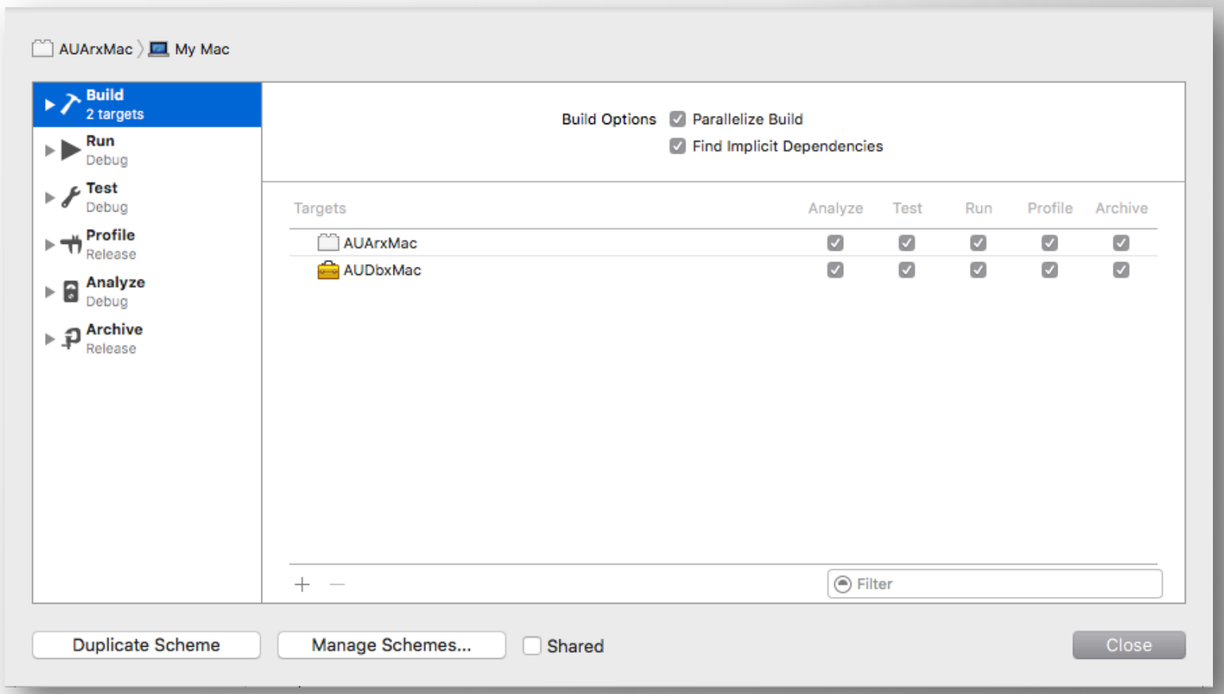
- OSX dynamic libraries default to global Library path: **$(LOCAL_LIBRARY_DIR)/Frameworks** which resolves to **/Library/Frameworks**
- To fix that, we will modify the **dbx_common.xcconfig** and **arx_common.xcconfig** files in DBX and ARX projects: **INSTALL_PATH = @rpath**
- By doing this we can freely load the modules from other folders inside Mac OSX

Creating MacOSX Project

Adjusting the Schemes and Debugging the Project

- Keep only the **AUArxMac** Scheme
- The AUArxMac **Scheme** will build both **bundle** and **DBX** targets

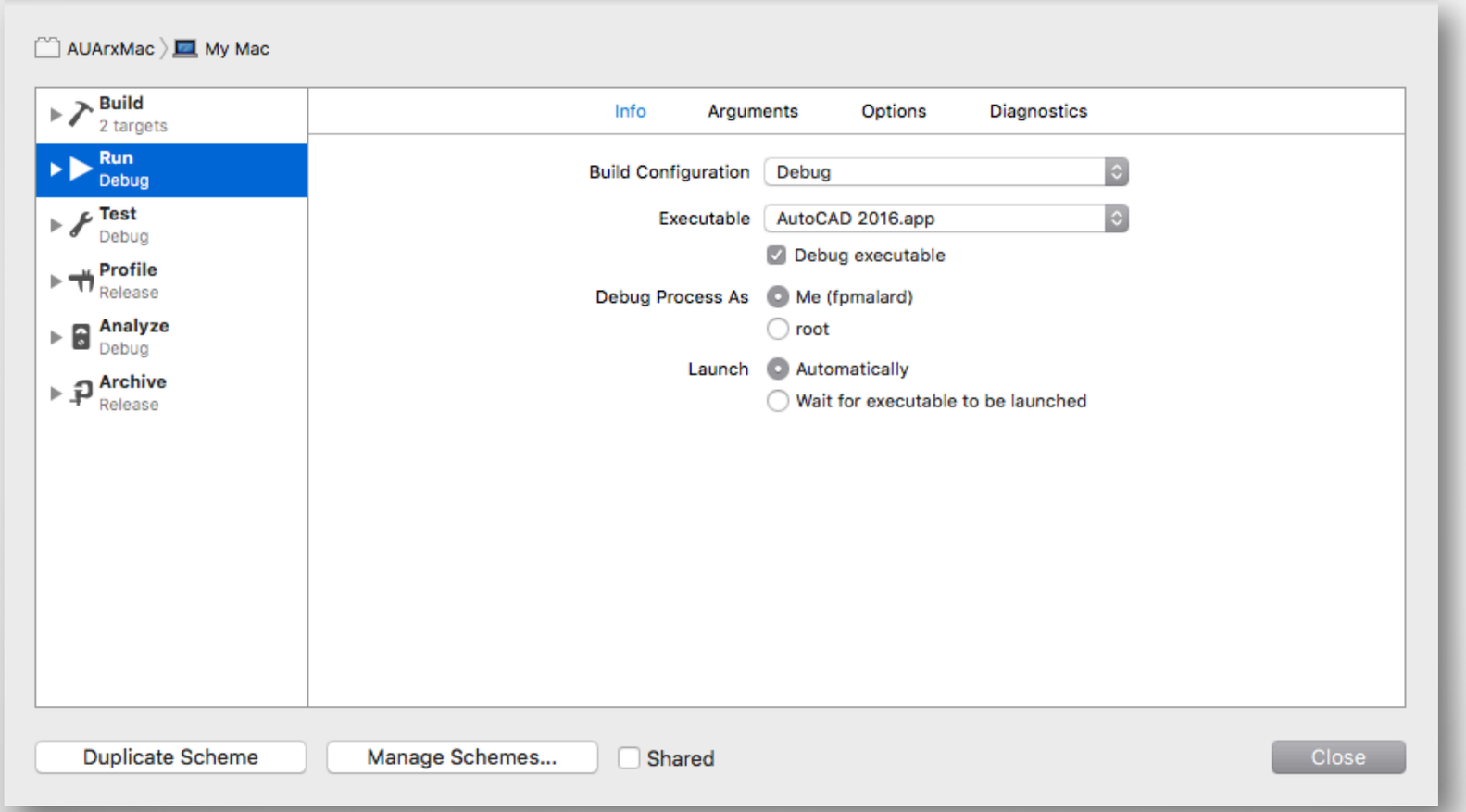




Creating MacOSX Project

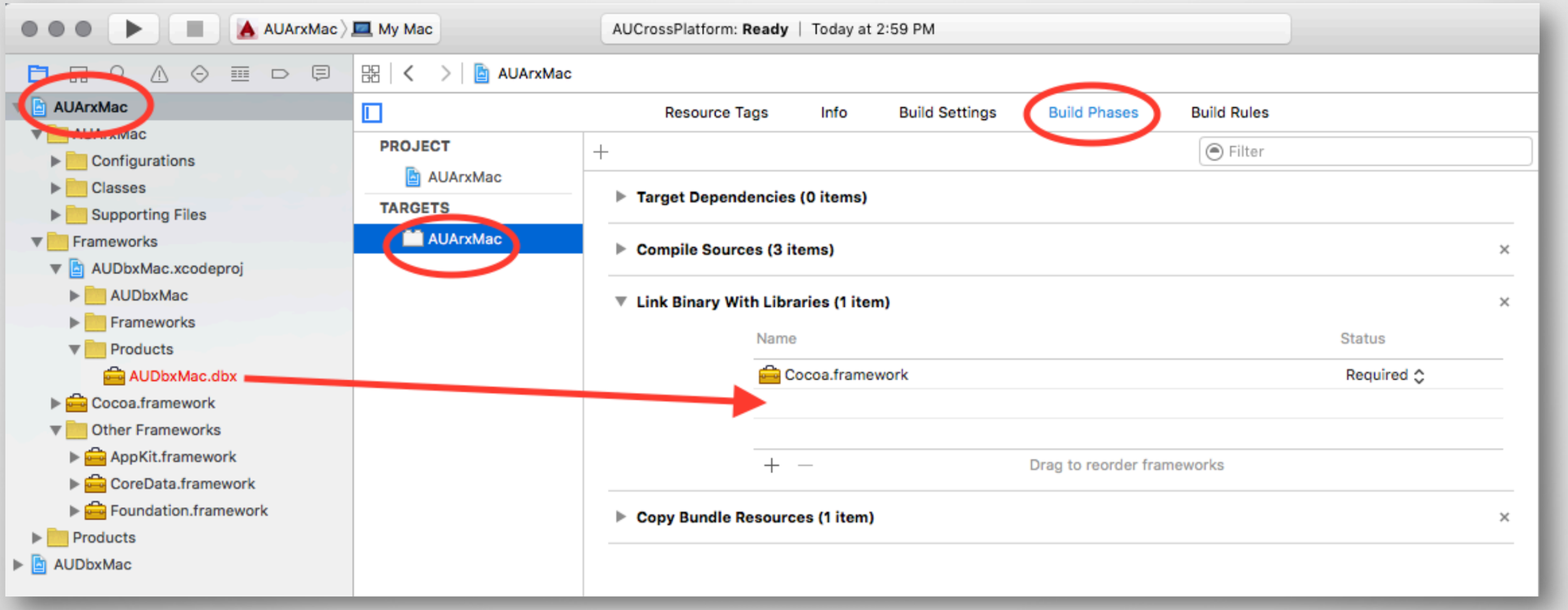
Adjusting the Schemes and Debugging the Project

- Edit **Scheme** to point to AutoCAD 2016.app



Creating MacOSX Project
Adding the shared Windows files

- Don’t add **Windows** specific files
- The **ARX** (bundle) module will **link to the DBX** module



Creating MacOSX Project
Adding the shared Windows files

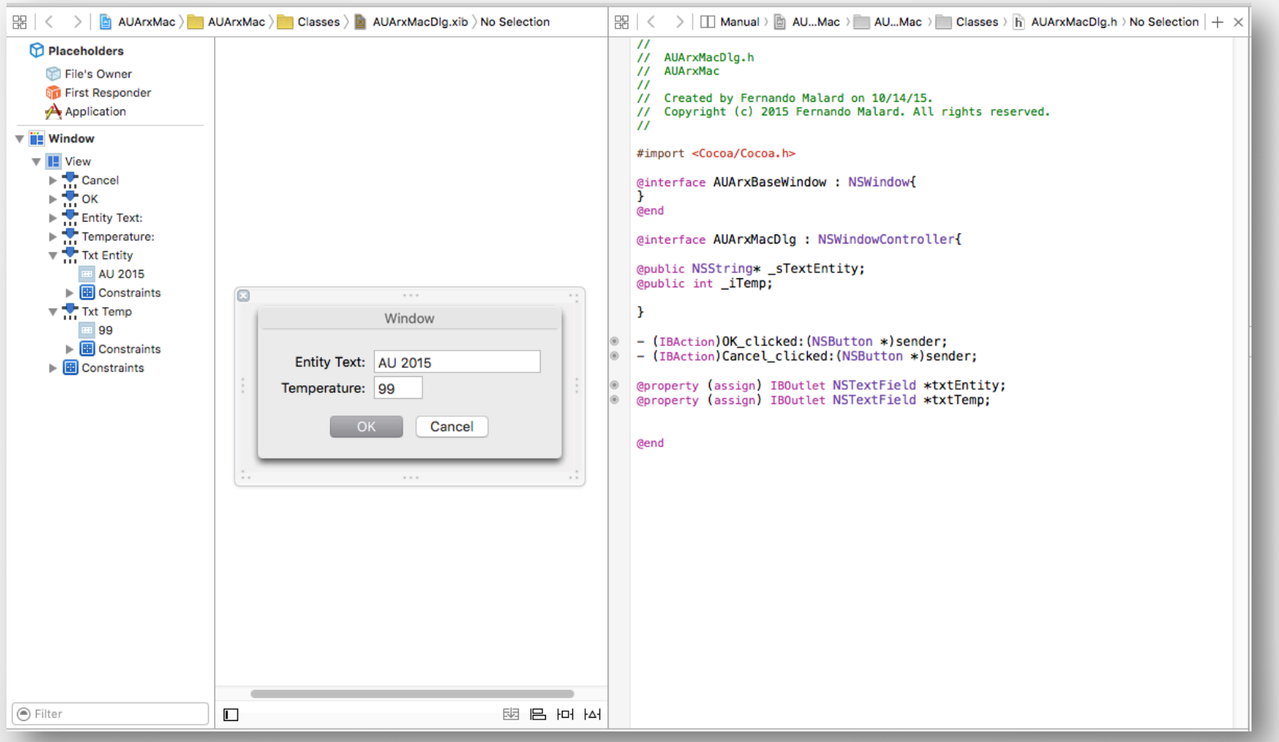
- Use **_ADESK_MAC_** definition when necessary
- Make sure you separate **specific platform** routines
- Adjust the **#include** paths (use forward slashes)

```
#ifdef _ADESK_MAC_
    #include "../AUDbxWindows/AUDbxEntity.h"
     extern bool ShowAUArxCocoa(AcString& txtEntity, int& iTemp);
#else
    #include "resource.h"
    #include "..\AUDbxWindows\AUDbxEntity.h"
    #include "AUArxWindowsDlg.h"
#endif
```

Creating the Mac UI with Cocoa and Objective-C

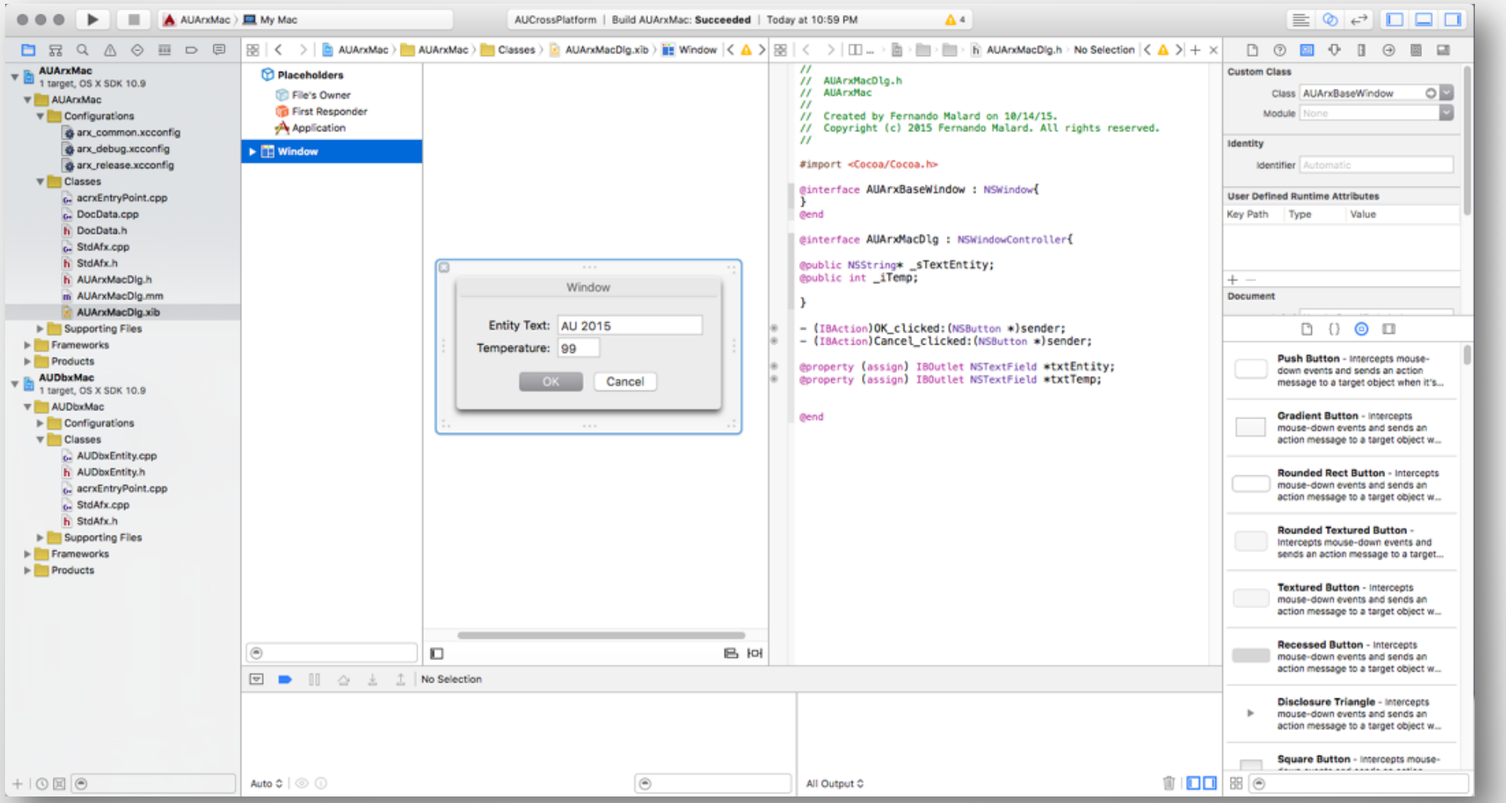
Creating the Mac UI with Cocoa and Objective-C

- Cocoa interfaces are driven by **XIB** files
- Each Window will have its **Objective-C Controller** class



Creating the Mac UI with Cocoa and Objective-C

- **Controller** class will map window controls to class members (**IBOutlet**) and events (**IBAction**)



Creating the Mac UI with Cocoa and Objective-C

- **.mm** extension, as an **Objective-C++** file, will allow a mix of C++ code and Objective-C
- **NSWindow** default class will need to be overridden due the **canBecomeMainWindow** event we need to adjust
- "**NO**" should be returned
- Change the Window **base class** to this new class

AUArxMacDlg.h:

```
@interface AUArxBaseWindow : NSWindow{
}
@end
```

AUArxMacDlg.mm:

```
// Base Window class
@implementation AUArxBaseWindow
- (BOOL)canBecomeMainWindow
{
    return NO;
}
@end
```

Creating the Mac UI with Cocoa and Objective-C

- To show the Window, we need to use **Objective-C** code
- Watch for **x64** details like the size of **wchat_t**
- The window needs to be initialized with the **NIB** (**XIB**) name

```
bool ShowAUArxCocoa(AcString& txtEntity, int& iTemp)
{
    AUArxMacDlg* pWnd = [AUArxMacDlg alloc];
    const wchar_t* pText = txtEntity;

    pWnd->_sTextEntity = [[[NSString alloc] initWithBytes:pText
       length:wcslen(pText)*sizeof(wchar_t)
       encoding:NSUTF32LittleEndianStringEncoding] autorelease];
    pWnd->_iTemp = iTemp;

    [pWnd initWithWindowNibName:@"AUArxMacDlg"];
    int iRes = [NSApp runModalForWindow:pWnd.window];

    iTemp = [pWnd txtTemp].intValue;

    NSData* d = [[pWnd txtEntity].stringValue
              dataUsingEncoding:NSUTF32LittleEndianStringEncoding];
    std::wstring sRet = std::wstring((wchar_t *)[d bytes],
       [d length]/sizeof(wchar_t));
    txtEntity = sRet.c_str();

    [pWnd release];
    return (iRes > 0);
}
```

Creating the Mac UI with Cocoa and Objective-C

- With **_ADESK_MAC_** the code to open both **MFC** and **Cocoa** interfaces

```
static void AuMyGroupAUCROSSDLG()
{
    bool bCreate = false;
    AcString txtEntity = _T("Autodesk");
    int iTemp = 14;

     #ifdef _ADESK_MAC_
        bCreate = ShowAUArxCocoa(txtEntity, iTemp);
     #else
        AUArxWindowsDlg dlg(CWnd::FromHandle(adsw_acadMainWnd()));
        bCreate = (dlg.DoModal() == IDOK);
        txtEntity.format(_T("%s"),dlg._sText.GetBuffer());
        iTemp = dlg._iTemp;
     #endif

    if (bCreate)
    {
        ads_point pti, ptj;
        if (acedGetPoint(NULL, _T("\nClick at the start point:"), pti) != RTNORM)
            return;
        if (acedGetPoint(pti, _T("\nClick at the end point:"), ptj) != RTNORM)
            return;

        CreateCustomEntity(pti, ptj, (ACHAR*)txtEntity.constPtr(), iTemp);
    }
    else
        acutPrintf(_T("\nDialog cancelled."));
}
```

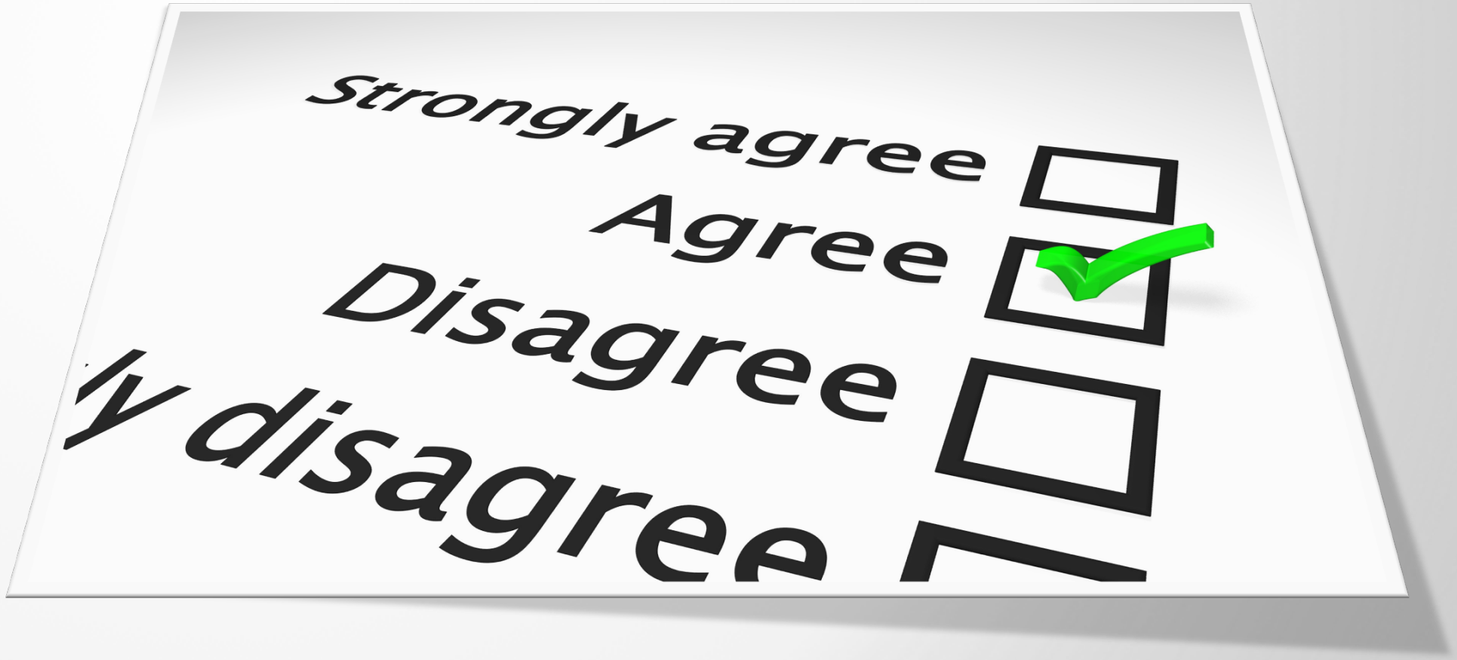
Conclusion

Conclusion

- Cross Platform applications can **share source code**
- Specific definitions should be used like **_ADESK_MAC_**
- Interfaces are totally different, **Cocoa** is recommended
- Multiple AutoCAD releases will have **different SDKs**
- **Windows** is different from **Mac OSX**
- **Objective-C** code can be mixed with **C++**
- **DBX** and **ARX** code can be shared
- **Xcode** and **Visual Studio** should be mastered

Be heard! Provide AU session feedback.

- Via the Survey Stations, email or mobile device.
- AU 2016 passes awarded daily!
- Give your feedback after each session.
- Give instructors feedback in real-time.



Forget to take notes? No problem!

After AU visit:

AutodeskUniversity.com

Click on **My AU** to find:

- Class Recordings
- Presentations
- Handouts

All of your sessions will be there to enjoy again and again.