COM Wrapper

Tutorial on COM Servers for Custom Objects

by Chandar Kumar Oddiraju

Version 2.0 - August 2009, Marat Mirgaleev

Autodesk Developer Services

Autodesk®

Table of Contents

Introduction

In this tutorial you will see how to make your ObjectARX custom objects available to COM. Doing so brings two primary benefits:

- You can leverage the ease of use of languages such as Visual Basic, Visual Basic for Applications, Java and Delphi to manipulate your custom objects.
- The Properties window of AutoCAD will be able to interact with your object intelligently.

Creating a COM wrapper for your custom object requires a reasonable amount of work. Autodesk provides two tools to make this task easier: an ATL-based framework that implements most of the interfaces that you need on a COM wrapper and the ObjectARX Wizards that let you configure this framework through easy to use dialogs. For more information on the ATL based framework, see the "Using COM for ObjectARX Development" chapter in the ObjectARX Developer Guide or consult the online documentation.

To create a COM wrapper you need to have an ObjectARX custom object. In the first part of the tutorial, we will build a simple custom object then in the second half of the tutorial we will create a COM wrapper for our custom object.

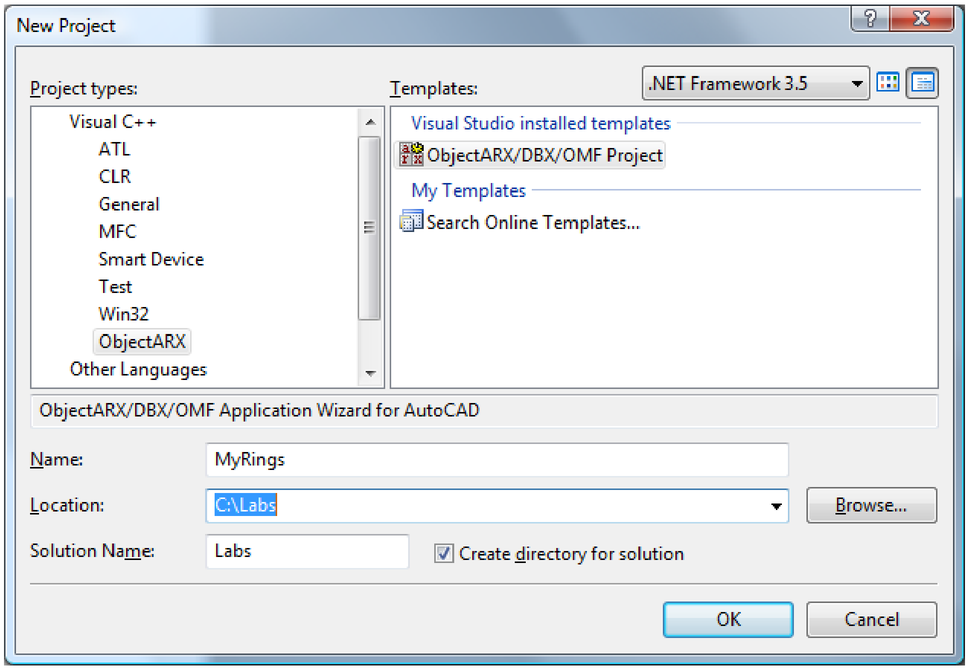
1 Creating the custom object module (DBX)

Let's create a project that defines a custom entity. We will use the ObjectARX Application Wizard to create this project. Make sure that you have downloaded the latest version of the ObjectARX Wizard from the ADN web site.

1.1 *Create a custom object project*

On the **File** menu, click **New** then **Project**.

Select **ObjectARX and ObjectARX/DBX/OMF Project**.



ObjectARX/DBX/OMF Application Wizard for AutoCAD

Name: MyRings

Location: C:\Labs

Solution Name: Labs

Create directory for solution: ✓

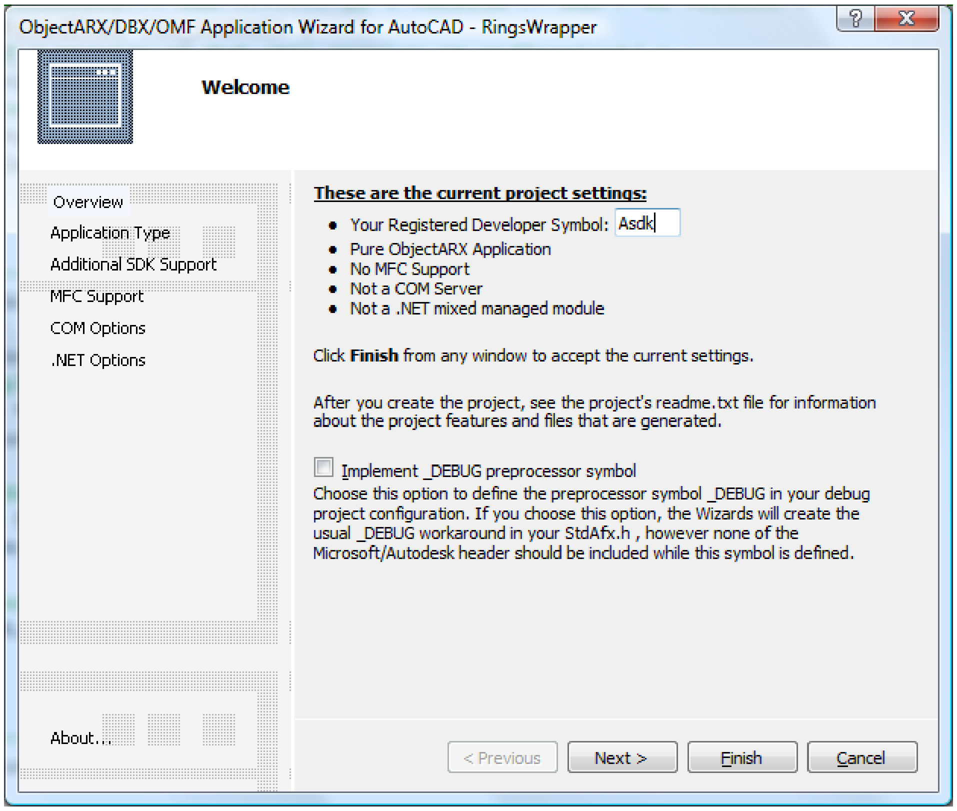
Type "`MyRings`" in the **Name** box.

Select the project location.

Type "`Labs`" in the **Solution Name** box.

Click **OK**.

ObjectARX/DBX/OMF Application Wizard will start (see the picture below).



ObjectARX/DBX/OMF Application Wizard for AutoCAD - RingsWrapper

Welcome

These are the current project settings:

- Your Registered Developer Symbol: Asdk
- Pure ObjectARX Application
- No MFC Support
- Not a COM Server
- Not a .NET mixed managed module

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.

☐ Implement _DEBUG preprocessor symbol

Choose this option to define the preprocessor symbol _DEBUG in your debug project configuration. If you choose this option, the Wizards will create the usual _DEBUG workaround in your StdAfx.h , however none of the Microsoft/Autodesk header should be included while this symbol is defined.

Here you can enter your Registered Developer Symbol, but we will use `Asdk` symbol in this tutorial. (To register you own developer symbol please go to the ADN website.)

Click **Next**.

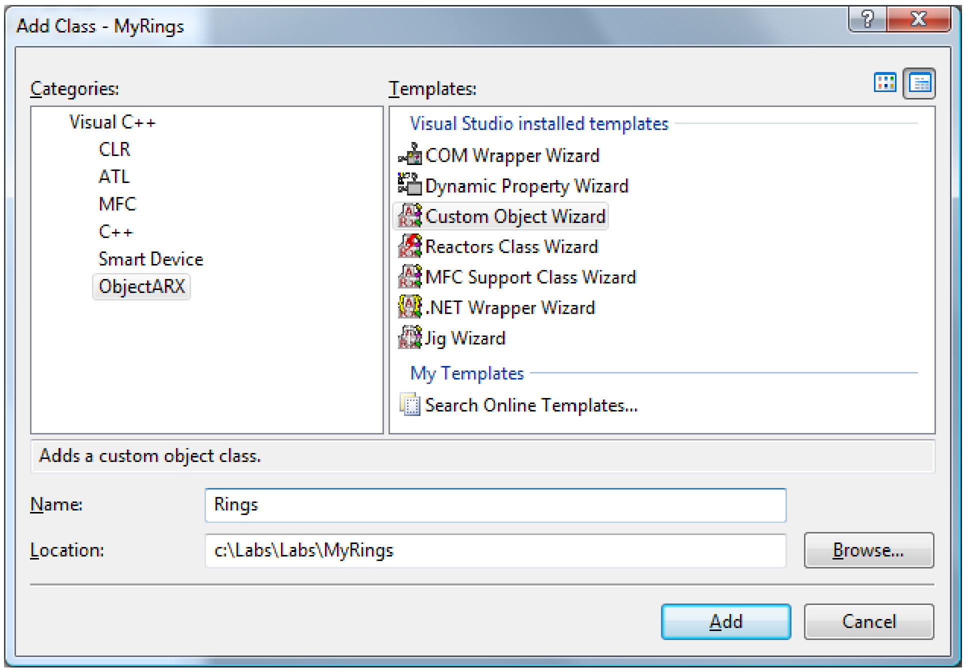
Choose **ObjectDBX (Custom Object Enabler)**.

Click **Finish**.

1.2 Adding Custom object

Let's define a custom entity and add it to our project.

Select **Project** – **Add Class** from the main menu. Click **ObjectARX** and **Custom Object Wizard**.


Add Class - MyRings

Categories:
- Visual C++
  - CLR
  - ATL
  - MFC
  - C++
  - Smart Device
  - ObjectARX

Templates:

Visual Studio installed templates
- COM Wrapper Wizard
- Dynamic Property Wizard
- Custom Object Wizard
- Reactors Class Wizard
- MFC Support Class Wizard
- .NET Wrapper Wizard
- Jig Wizard

My Templates
- Search Online Templates...

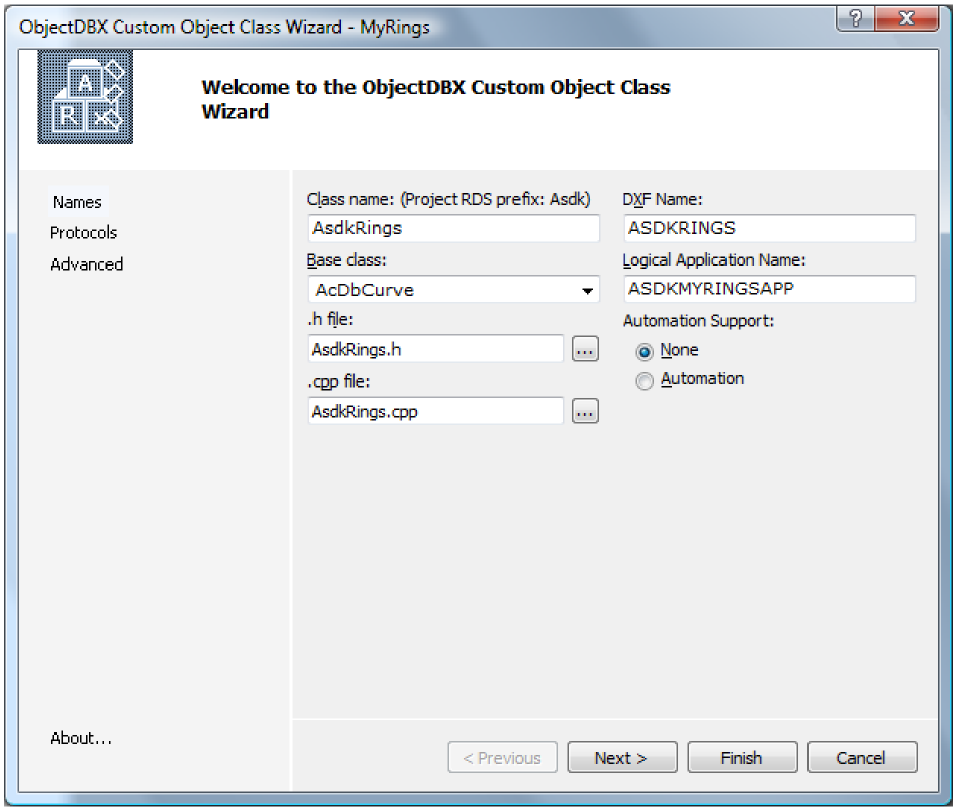
Adds a custom object class.

Name: Rings

Location: c:\Labs\Labs\MyRings

In **Name** enter "`Rings`". Then click **Add**.

You will be presented with the **ObjectDBX Custom Object Class Wizard** dialog.


ObjectDBX Custom Object Class Wizard - MyRings

Welcome to the ObjectDBX Custom Object Class Wizard

Class name: (Project RDS prefix: Asdk) AsdkRings

Base class: AcDbCurve

.h file: AsdkRings.h

.cpp file: AsdkRings.cpp

DXF Name: ASDKRINGS

Logical Application Name: ASDKMYRINGSAPP

Automation Support: [✓ None] Automation

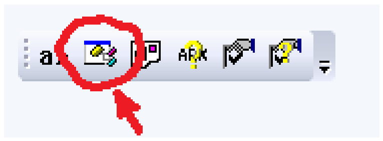
Enter "`AsdkRings`" in **Class name**.

Select **AcDbCurve** in the **Base class** dropdown list.

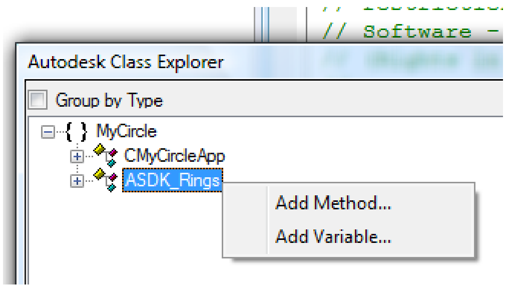
Click **Finish**.

1.3 Define Custom Object Attributes

Next, we will add some Attributes to our Custom Object. Click the **Autodesk Class Explorer** icon in the **ObjectARX AddIn** toolbar:

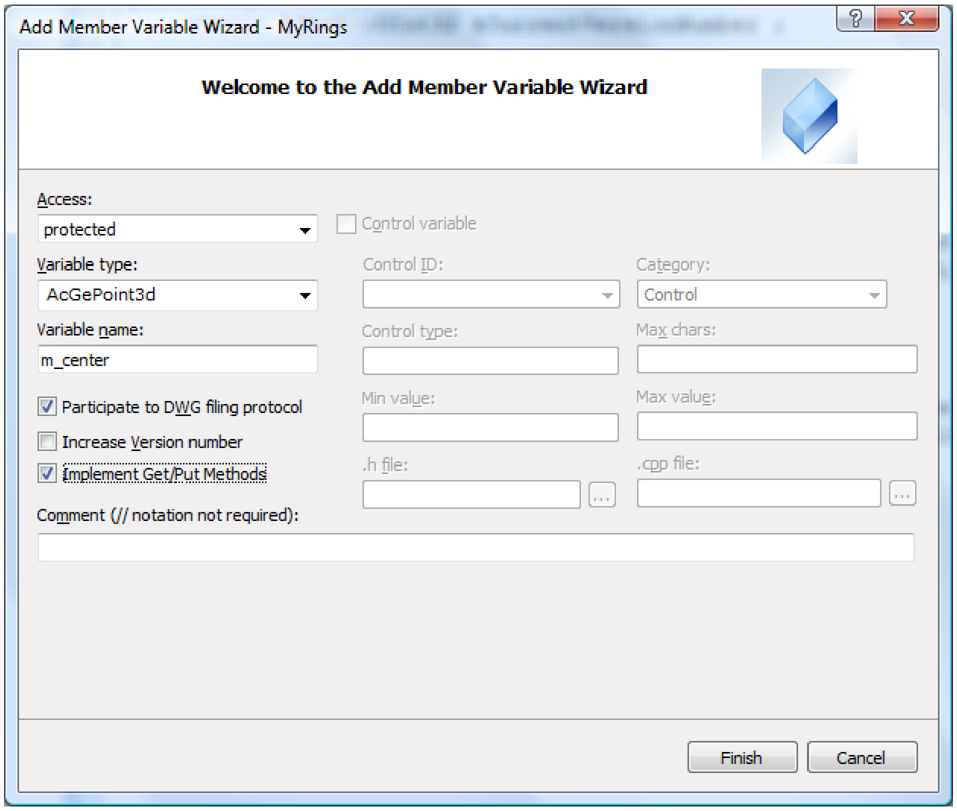


then right-click **AsdkRings** in the **Autodesk Class Explorer**:



and click the **Add Variable** item.

In the Add Member Variable Wizard, select **protected** in **Access**.


Add Member Variable Wizard - MyRings

Welcome to the Add Member Variable Wizard

Access: protected

Variable type: AcGePoint3d

Variable name: m_center

[✓] Participate to DWG filing protocol

[ ] Increase Version number

[✓] Implement Get/Put Methods

[ ] Control variable

Control ID:

Category: Control

Control type:

Max chars:

Min value:

Max value:

.h file:

.cpp file:

Comment (// notation not required):

From the **Variable type** select **AcGePoint3d**.

Enter `"m_center"` in the **Variable name**.

Uncheck **Increase Version number**.

Check **Implement Get/Put Methods**.

Press **Finish**.

Similarly, create all the remaining variables from the table:

Name	Type
m_center	AcGePoint3d
m_normal	AcGeVector3d
m_radius	double
m_rings	Adesk::UInt8

You can check the put- end get-methods in the *AsdkRings.h* and *.cpp* files, for example:

```
double AsdkRings::get_m_radius(void) const
{
    assertReadEnabled () ;
    return (m_radius) ;
}
```

I prefer to rename it to `get_radius` and all other methods similarly.

Try to build the project. On my computer it gives a compile error: `'NaN' : undeclared identifier.` Just replace NaN with 0, it should be Ok. Also don't forget to specify the ObjectARX include and library paths in the project settings. So, it builds now, let's go ahead.

1.4 *Functions you override*

As our `AsdkRings` class derives from `AcDbCurve`, we may need to override some methods of the base class. These methods include `subGetClassID()`, `subWorldDraw()` and a few others.

The Wizard also adds some other overridden methods to the class definition.

We are going to change the graphic representation of our custom entity. To do this we will modify the `subWorldDraw()` function created by the Wizard, as shown below, in order to draw a series of circles with the same center:

```
Adesk::Boolean AsdkRings::subWorldDraw (AcGiWorldDraw *mode)
{
    assertReadEnabled();
    double step = m_radius / m_rings; // Calculate the increment for each ring
    double radius = step; // Initialize the first radius

    // Draw a circle for each radius
    for(int i = 0; i < m_rings ; i++, radius += step)
          mode->geometry().circle(m_center, radius, m_normal);

     return true;
}
```

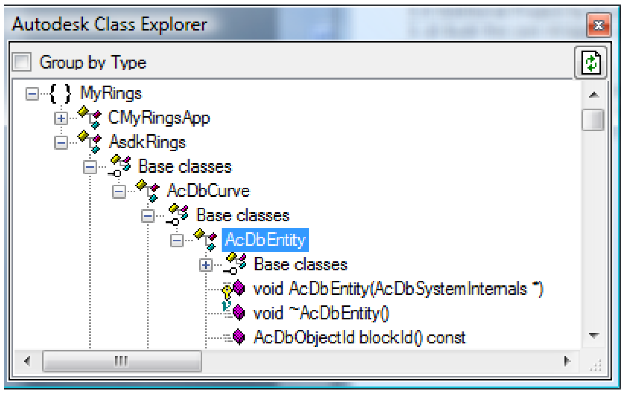
Copy the code above and paste it instead of the existing `subWorldDraw()`.

1.5 *Implementation of subGetClassID() in Custom Object*

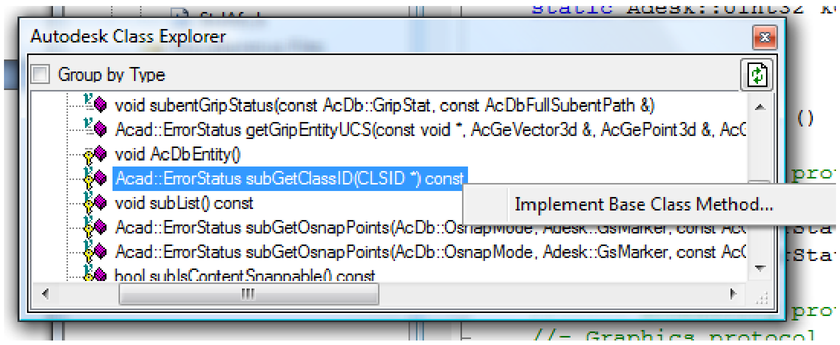
The important function that has to be overridden by the custom object is `subGetClassID()`. This function is used by AutoCAD whenever AutoCAD needs to retrieve the COM wrapper for a custom object given a pointer or id of the custom object. This happens, for example, when the user selects the custom object while the AutoCAD Properties window is active.

To override the `subGetClassID()` method:

Expand the **AsdkRings** branch in the **Autodesk Class Explorer**, then expand **Base classes**, then **AcDbCurve** –> **Base classes** –> **AcDbEntity** (see the picture below).



Find **Acad::ErrorStatus subGetClassID(CLSID *)** const in the list and right-click it:


Autodesk Class Explorer
Group by Type
void subentGripStatus(const AcDb::GripStat, const AcDbFullSubentPath &)
Acad::ErrorStatus getGripEntityUCS(const void *, AcGeVector3d &, AcGePoint3d &, AcC
void AcDbEntity()
Acad::ErrorStatus subGetClassID(CLSID *) const
void subList() const
Acad::ErrorStatus subGetOsnapPoints(AcDb::OsnapMode, Adesk::GsMarker, const AcC
Acad::ErrorStatus subGetOsnapPoints(AcDb::OsnapMode, Adesk::GsMarker, const AcC
bool subIsContentSnappable() const
Implement Base Class Method...

Click **Implement Base Class Method** in the menu.

The `subGetClassID()` method will be added to the *AsdkRings.h* and *.cpp* files.

To implement this function correctly we need the CLSID of our COM wrapper, something that we do not have at this point. We will come back to this function in Section 0.

1.6 *Build AsdkRings.dbx*

Now we are ready to build the project. After a successful build, copy the `AsdkMyRings.dbx` file into "`C:\Program Files\Common Files\Autodesk Shared\`" directory. If you want, you can use a different directory, however you are advised to put the directory in System Path. Otherwise, you are required to load the `.DBX` file manually into AutoCAD. Our COM server implementing the COM wrapper will have a load-time dependency on our custom object. The operating system must be able to resolve this dependency at load time and the best way to make sure that it is possible is to put our DBX on the operating system path. For more information on how the OS resolves DLL dependencies, see DLL topics on MSDN.

2 Creating the wrapper class for the custom object

Now we will provide COM support for our `Rings` Object.

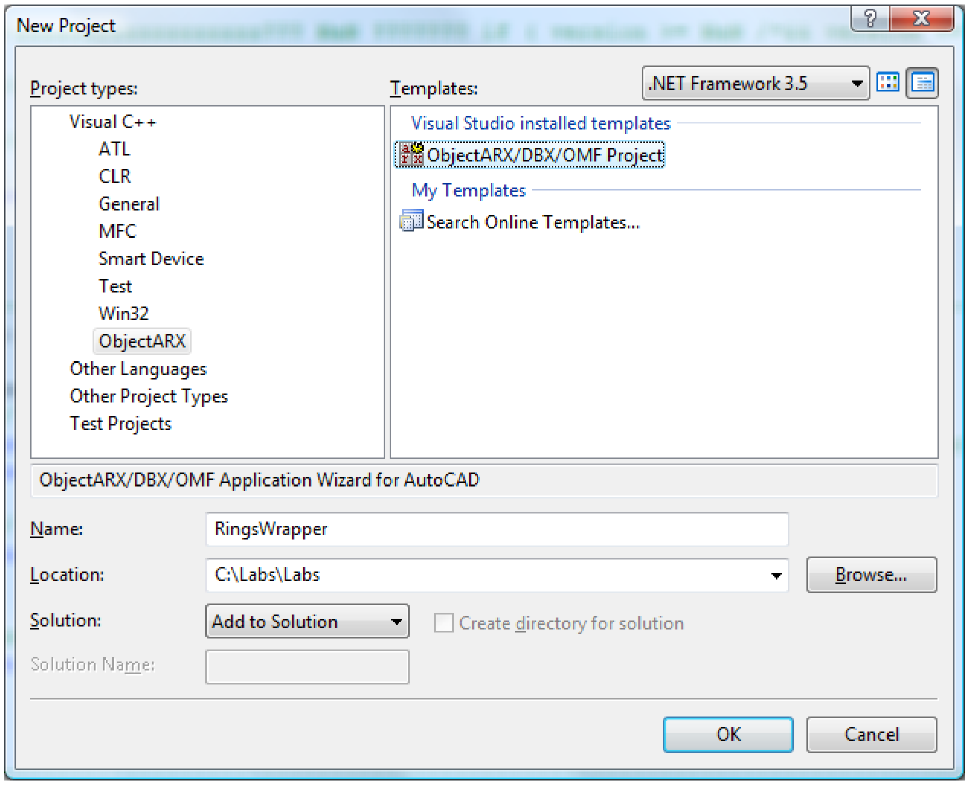
2.1 *Create a project*

Select **File -> New -> Project** from the main menu.

In the New Project dialog box select **ObjectARX** and **ObjectARX/DBX/OMF Project**.

Enter "`RingsWrapper`" for the **Name**.

Make sure that you have selected **Add to Solution** from the **Solution** drop-down list.



ObjectARX/DBX/OMF Application Wizard for AutoCAD

Name: RingsWrapper

Location: C:\Labs\Labs

Solution: Add to Solution

Click the **OK** button.

The next dialog you are presented with is the ObjectARX Wizard. First, you have to enter your registered developer symbol as before. Use `Asdk` for this sample.

Press **Next** four times (leaving the default settings on those tabs of the Wizard).

Check the **Implement a COM Server using ATL** radio button. This will activate the **Use ObjectARX ATL Extensions for Custom Objects** check box, make sure it is checked.

Select the **Finish** button.

2.2 *ObjectARX Wizard created source code*

Now we have a project that contains the skeleton for an ObjectARX application which uses ATL. The project contains a number of files.

Open the *RingsWrapper.cpp* file. You can see that it implements the necessary functions needed for ObjectARX, COM and ATL:

```
//-----------------------------------------------------------------------------
//- RingsWrapper.cpp : Initialization functions
//-----------------------------------------------------------------------------
#include "StdAfx.h"
#include "resource.h"
#include <initguid.h>
#include "RingsWrapper.h"
#include "RingsWrapper_i.c"
#include "acadi_i.c"


//-----------------------------------------------------------------------------
class CRingsWrapperModule : public CAtlDllModuleT<CRingsWrapperModule> {

public :
    DECLARE_LIBID(LIBID_AsdkRingsWrapperLib)
    DECLARE_REGISTRY_APPID_RESOURCEID(IDR_RINGSWRAPPER,
        "{5C6BB8CF-D672-4CF0-9393-6704D074E47E}")
} ;

CRingsWrapperModule _AtlModule ;

//-----------------------------------------------------------------------------
//- DLL Entry Point
extern "C"
BOOL WINAPI DllMain (HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved) {
    //- Remove this if you use lpReserved
    UNREFERENCED_PARAMETER(lpReserved) ;

    if ( dwReason == DLL_PROCESS_ATTACH ) {
        _hdllInstance =hInstance ;
    } else if ( dwReason == DLL_PROCESS_DETACH ) {
    }
    return (_AtlModule.DllMain (dwReason, lpReserved)) ;
}

//-----------------------------------------------------------------------------
//- Used to determine whether the DLL can be unloaded by OLE
STDAPI DllCanUnloadNow (void) {
    HRESULT hr =(_AtlModule.GetLockCount () == 0 ? S_OK : S_FALSE) ;
    return (hr) ;
}

//-----------------------------------------------------------------------------
//- Returns a class factory to create an object of the requested type
STDAPI DllGetClassObject (REFCLSID rclsid, REFIID riid, LPVOID *ppv) {
    return (_AtlModule.GetClassObject (rclsid, riid, ppv)) ;
}

//-----------------------------------------------------------------------------
//- DllRegisterServer - Adds entries to the system registry
STDAPI DllRegisterServer (void) {
    //- Registers object, typelib and all interfaces in typelib
    return (_AtlModule.RegisterServer (TRUE)) ;
}

//-----------------------------------------------------------------------------
//- DllUnregisterServer - Removes entries from the system registry
STDAPI DllUnregisterServer (void) {
    return (_AtlModule.UnregisterServer (TRUE)) ;
}

//-----------------------------------------------------------------------------
#ifndef _WIN64
#pragma comment(linker, "/EXPORT:DllCanUnloadNow=_DllCanUnloadNow@0,PRIVATE")
#pragma comment(linker, "/EXPORT:DllGetClassObject=_DllGetClassObject@12,PRIVATE")
#pragma comment(linker, "/EXPORT:DllRegisterServer=_DllRegisterServer@0,PRIVATE")
#pragma comment(linker, "/EXPORT:DllUnregisterServer=_DllUnregisterServer@0,PRIVATE")
```

```
#else
#pragma comment(linker, "/EXPORT:DllCanUnloadNow=DllCanUnloadNow,PRIVATE")
#pragma comment(linker, "/EXPORT:DllGetClassObject=DllGetClassObject,PRIVATE")
#pragma comment(linker, "/EXPORT:DllRegisterServer=DllRegisterServer,PRIVATE")
#pragma comment(linker, "/EXPORT:DllUnregisterServer=DllUnregisterServer,PRIVATE")
#endif
```

This file defines an instance of the `CRingsWrapperModule` class as well as the `DllMain()` function which initializes and terminates the instance.

Windows calls the `DllCanUnloadNow()` function to test if it can unload your application, and `DllGetClassObject()` to create an instance of one of the COM objects your application defines.

The `DllRegisterServer()` function registers our COM server application in the Windows registry, and the `DllUnregisterServer()` function removes the entries from the Windows registry if the application should be removed.

Also we have a standard ObjectARX application class in the *acrxEntryPoint.cpp* file which is called by AutoCAD when our application is loaded.

There is one point that must be clear at this time. Although the DLL we have created is a COM server, it can only be loaded into the AutoCAD process. This is because the DLL is an ObjectARX application that has a load-time dependency on **acad.exe**. We provide an `IMPLEMENT_ARX_ENTRYPOINT` macro, which creates an `acrxEntryPoint()` function, in this DLL, so that the user can load it on the AutoCAD command line using the ARX command. In the `RegisterServerComponents()` virtual method the DLL calls `DllRegisterServer()` thus registering itself in the registry. Note that once the DLL is registered, COM will be able to load this DLL via its own activation mechanism. It is important to understand that when COM loads the DLL, the `acrxEntryPoint()` function is NOT called. Therefore, it is advisable to do only self-registration when `acrxEntryPoint()` is called and all other necessary initialization is done in `DllMain()`.

Since this ARX is not the usual ObjectARX application, it is recommended NOT to register any commands in it.

You can build this project now. To do this:

Go to the main menu -> **Build** -> **Configuration Mana**ger and set the checkbox in the **Build** column for the **RingsWrapper** project.

Then, the main menu -> **Build** -> **Build Solution**.

On my computer I get the following error:

```
.\RingsWrapper.rc(102) : error RC2135 : file not found: AsdkRingsWrapper.tlb
```

I simply double-click on the error message, that brings me to the *RingsWrapper.rc* file and to the following line:

```
1 TYPELIB "AsdkRingsWrapper.tlb"
```

I replace it with

```
1 TYPELIB "x64/Debug/AsdkRingsWrapper.tlb"
```

(your path may differ) and everything builds successfully.

Load the created .ARX file in AutoCAD, but currently it does nothing.

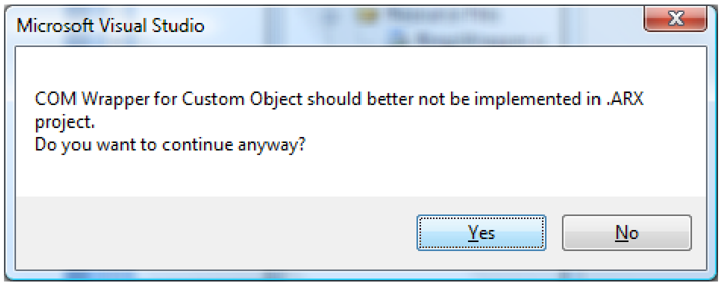
2.3 *Adding a new Custom object wrapper*

In the **Solution Explorer**, right-click the **RingsWrapper** project and select **Add**, then **Class**.

In the Add Class dialog choose **ObjectARX** and **COM Wrapper Wizard**.

Enter "Rings" for the **Name** and press **Add**.

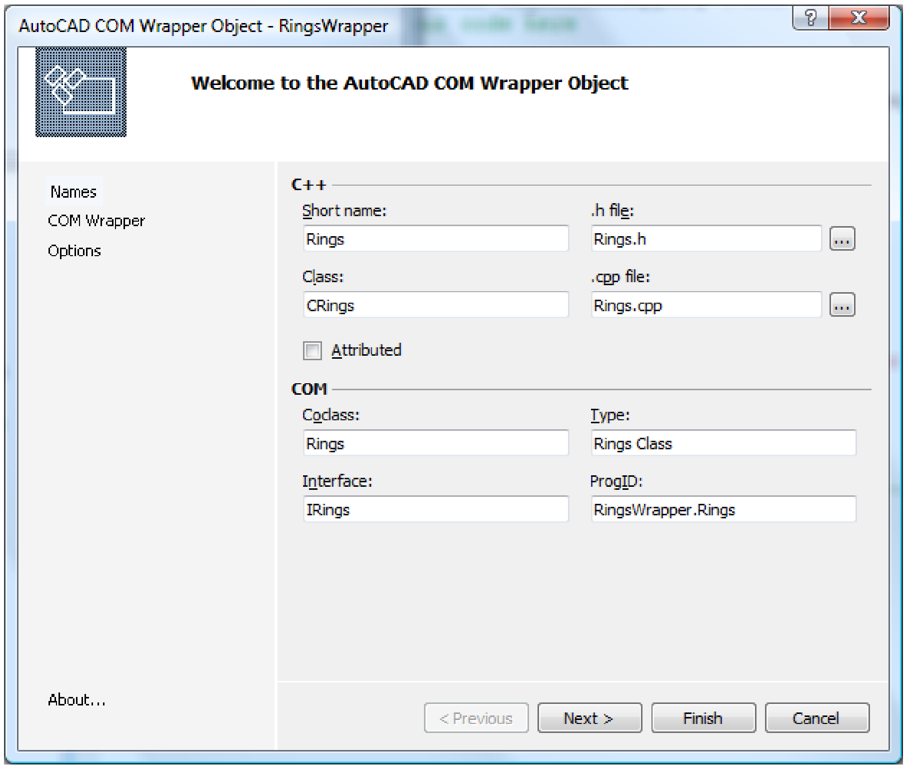
Answer "`Yes`" in the following message box:



Microsoft Visual Studio

COM Wrapper for Custom Object should better not be implemented in .ARX project.
Do you want to continue anyway?

You will see the AutoCAD COM Wrapper Object dialog. Here you can define the new COM object. In the **Short Name** field enter "`Rings`". The Wizard fills the other fields automatically:



AutoCAD COM Wrapper Object - RingsWrapper

Welcome to the AutoCAD COM Wrapper Object

C++

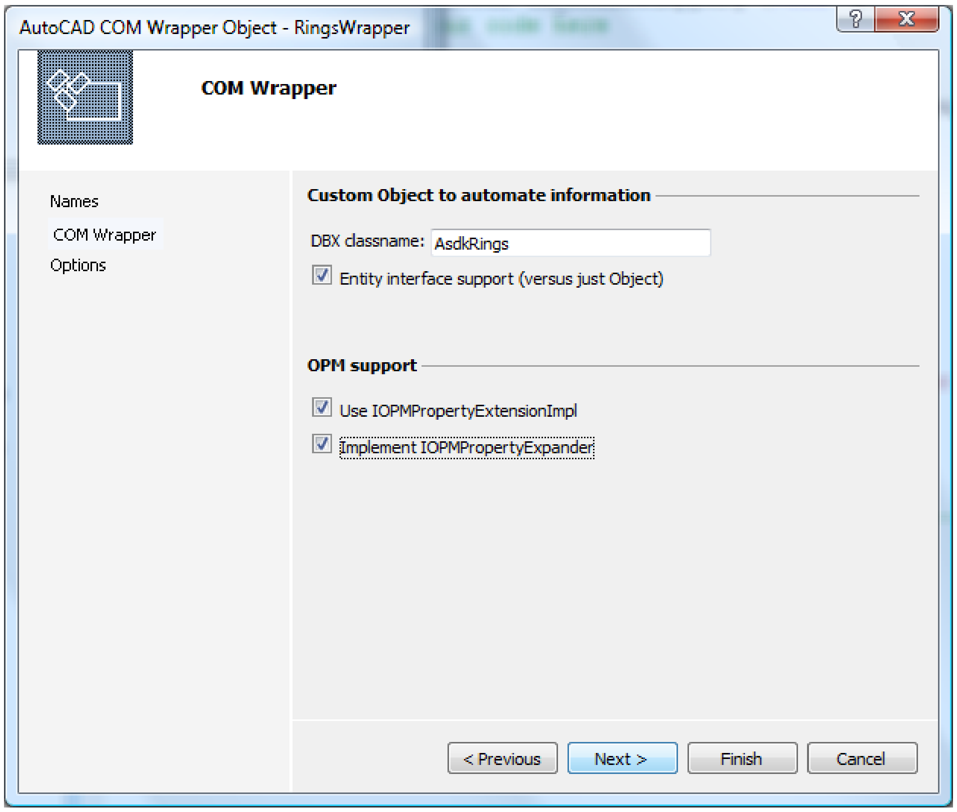
Short name: Rings
.h file: Rings.h
Class: CRings
.cpp file: Rings.cpp
Attributed: ☐

COM

Coclass: Rings
Type: Rings Class
Interface: IRings
ProgID: RingsWrapper.Rings

Click **Next**.

Type "`AsdkRings`" in **DBX classname**. This is the name of the custom entity we defined in our DBX module.


AutoCAD COM Wrapper Object - RingsWrapper

COM Wrapper

Custom Object to automate information

DBX classname: AsdkRings

☑ Entity interface support (versus just Object)

OPM support

☑ Use IOPMPropertyExtensionImpl

☑ Implement IOPMPropertyExpander

Check the **Entity interface support (versus just Object)** checkbox.

We also want to support the Property Inspector API (also known as the Object Property Manager, OPM), thus, check **Use IOPMPropertyExtensionImpl** and **Implement IOPMPropertyExpander** in the **OPM support** section. We will discuss this functionality in Section 4.

Click **Finish**.

The Wizard generates *Rings.cpp* and *Rings.h* files, which have the `CRing` class implementation and definition respectively. Read on for further discussion of the contents of these files.

Since we are wrapping around the AsdkRings custom object, it would be a good idea to check whether the custom object module is loaded or not, but we omit this in our sample.

Note that the COM server DLL has a load-time dependency on the DBX application implementing the custom object so the Operating System tries to load the DBX module into memory when the COM server DLL is loaded.

2.4 Implementation of subGetClassID() in Custom Object Revisited

Now we have everything we need to properly implement the `subGetClassID()` function for `AsdkRings` that we stubbed out in Section 1.5.

Modify the implementation in *AsdkRings.cpp* as follows:

```
Acad::ErrorStatus AsdkRings::subGetClassID(CLSID* pClsid) const
{
    assertReadEnabled();
    *pClsid = CLSID_Rings;
    return Acad::eOk;
}
```

The `CLSID_Rings` is defined in *RingsWrapper_i.c* file. This file is generated from the *RingsWrapper.idl* file when you build the RingsWrapper project. Add the following line to the AsdkRings.cpp after others #include's:

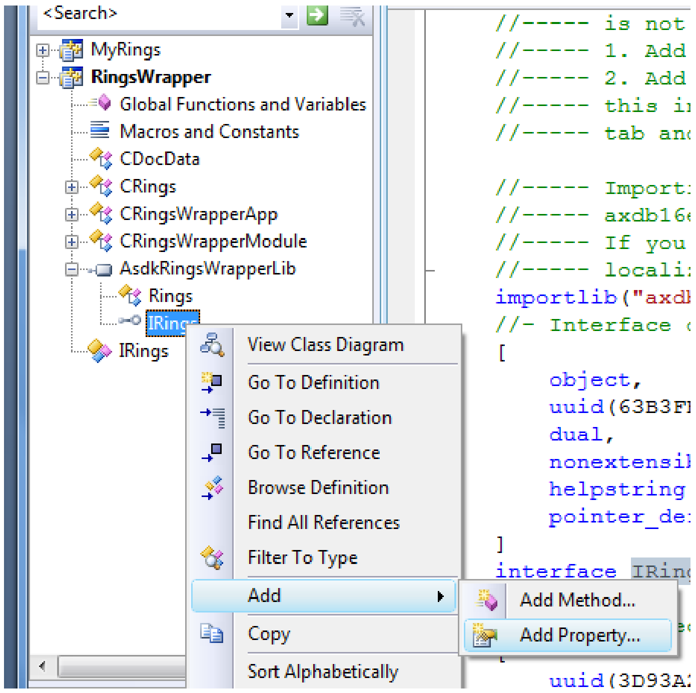
```
#include "../RingsWrapper/RingsWrapper_i.c"
```

And vice versa, in the RingsWrapper project, *Rings.cpp* file, we need to refer to another project:

```
#include "../MyRings/AsdkRings.h"
```

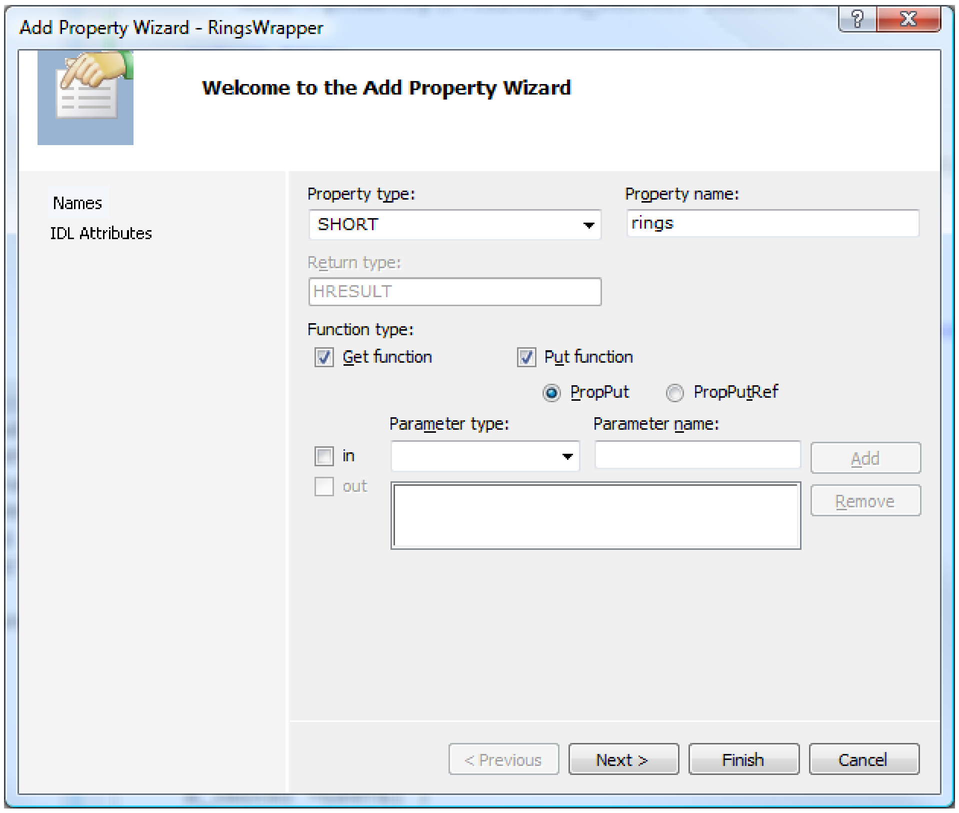
2.5 Adding new properties

Go to the **Class View** in the **Solution Explorer** and expand the **RingsWrapper** project, then expand the **AsdkRingsWrapperLib** branch and right-click on the **IRings** interface to get the popup context menu (see the picture). Select **Add -> Add Property**.



```
//----- is not
//----- 1. Add
//----- 2. Add
//----- this i
//----- tab an

//----- Import
//----- axdb16
//----- If you
//----- locali
importlib("axd
//- Interface
[
    object,
    uuid(63B3F
    dual,
    nonextensi
    helpstring
    pointer_de
]
interface IRin

    uuid(3D93A
```

You will be presented with the **Add Property Wizard**.


Add Property Wizard - RingsWrapper

Welcome to the Add Property Wizard

Property type: SHORT

Property name: rings

Return type: HRESULT

Function type:

[✓] Get function [✓] Put function

[● PropPut] PropPutRef

Enter "`rings`" in the **Property Name** field, type **short** in the **Property type** combo box. Ensure **Get Function** and **Put functions** are checked. Also select **PropPut** radio button (these are the defaults).

Click **Finish** to add this property to the Rings object.

Repeat the same process for adding the remaining properties from the table:

Property Type	Property Name
`short`	`rings`
`VARIANT`	`center`
`double`	`radius`
`VARIANT`	`normal`

2.6 *ObjectARX AppWizard generated functions*

First let us look into the *RingsWrapper.idl* file. You will see the following in the IRings interface section:

```
interface IRings : IAcadEntity
{
   [propget, id(1), helpstring("property rings")] HRESULT rings([out, retval] short *pVal);
   [propput, id(1), helpstring("property rings")] HRESULT rings([in] short newVal);
   [propget, id(2), helpstring("property center")] HRESULT center([out, retval] VARIANT *pVal);
   [propput, id(2), helpstring("property center")] HRESULT center([in] VARIANT newVal);
   [propget, id(3), helpstring("property radius")] HRESULT radius([out, retval] double *pVal);
   [propput, id(3), helpstring("property radius")] HRESULT radius([in] double newVal);
   [propget, id(4), helpstring("property normal")] HRESULT normal([out, retval] VARIANT *pVal);
   [propput, id(4), helpstring("property normal")] HRESULT normal([in] VARIANT newVal);
};
```

Here every property is assigned with a unique id value, also each property is assigned with two distinct functions, one of them is `'propput'` for setting the value and other is `'propget'` for retrieving the value. AppWizard will generate functions based on these function types. For `'propput'` you will have a `put_XXX()` function and for `'propget'` you will have a `get_XXX()` function, where `XXX` is the property name. You can change the `helpstring` according to your requirements.

Now look at the *Rings.cpp* and *Rings.h* files that have the actual implementation of our Custom Object wrapper class.

In the *Rings.h* file we see that the `CRings` class is multiple inherited from `CComObjectRootEx`, `CComCoClass`, `ISupportErrorInfo` and `IAcadEntityDispatchImpl`. Also it is inherited from `IOPMPropertyExtensionImpl` and `IOPMPropertyExpander.`

`CComObjectRootEx` and `CComCoClass` are provided by ATL. Please see the ATL documentation for details. `ISupportErrorInfo` is a standard COM interface that has only one method: `InterfaceSupportsErrorInfo()`, which is implemented in *Rings.cpp*.

`IAcadEntityDispatchImpl` represents the ATL-based framework provided on the ObjectARX SDK. It implements a number of interfaces that are required from a COM wrapper. This class is a template class, thus the full source code of this class is available to you on the ObjectARX SDK (see *axboiler.h*).

Now look at the `CreateNewObject()` function in *Ring.cpp*. In this function you create the custom object and append it to its owner in the database. The owner is passed to you as a parameter of this function.

Note that the framework calls this function in response to a call to the `IAcadBaseObject::SetObjectId()` method.

2.7 Modifications you need to do

All the modifications you do manually are in the *Rings.cpp* file.

In case of a single-threaded server like AutoCAD, all Automation calls go through the message loop and are dispatched to a hidden window that calls the method. Since we are driven off the message loop and we are in the application context, we need to explicitly lock the document before doing any operation on its database. We do this by instantiating an `AcAxDocLock` object and passing our objectId to it. For details on the `AcAxDocLock` class see the ObjectARX online help.

```
AcAxDocLock lock(m_objId);
```

You also need to check the lock status, if you don't succeed in locking the document, you are not allowed to do modifications, you must simply return from function.

```
if(lock.lockStatus()) // if something wrong return from here..
      return E_ACCESSDENIED;
```

The next step is to access the custom object that this COM object wraps. The framework stores the object ID of the underlying database object in the `m_objId` member variable. You can use this member variable anytime to gain access to the custom object. The best way is to use a 'smart pointer' to open the object:

```
AcDbObjectPointer<AsdkRings> pC(m_objId,AcDb::kForRead);
```

In your get methods you open the object for read while in your set methods you open the object for write. Then make sure that open operation succeeds:

```
If(pC.openStatus()!=Acad::eOk)
      return E_ACCESSDENIED;
```

Now, you can delegate to the database object to actually set or get the property.

In `get_xxxx(type *pVal)` functions you call the object method to retrieve the required property and set into the `pVal` argument.

In `put_xxx(type newVal)` functions you call the object method to set the required property to a value given in the `newVal` argument.

This task is simple as long as you are dealing with basic types. However, points and vectors are passed as `VARIANTs` on the COM interface, so, you will need to convert `VARIANTs` to `AcGePoint3d` or to `AcGeVector3d` and vice versa. To deal with `AcGePoint3d` you use the `AcAxPoint3d` class, which is specially designed for this purpose. `AcAxPoint3d` is wrapped around the `AcGePoint3d` class to implement the COM-specific functionality. See ObjectARX online help for further details.

To convert from `VARIANT` to `AcGePoint3d` you do as follows:

```
STDMETHODIMP CRings::put_center(VARIANT newVal)
{
    . . .
    AcDbObjectPointer<AsdkRings> pC(m_objId,AcDb::kForWrite);
    if(pC.openStatus()!=Acad::eOk)
        return E_ACCESSDENIED;
    AcAxPoint3d pt(newVal);
    pC->put_center(pt);
    . . .
}
```

Since `AcAxPoint3d` is derived from `AcGePoint3d`, you can safely pass this object in place of `AcGePoint3d` object as an argument.

For converting from `AcGePoint3d` to `VARIANT`, you call the `setVariant()` method defined in `AcAxPoint3d`.

```
STDMETHODIMP CRings::get_center(VARIANT *pVal)
{
        . . .
        AcDbObjectPointer<AsdkRings> pC(m_objId,AcDb::kForRead);
        if(pC.openStatus()!=Acad::eOk)
              return E_ACCESSDENIED;
        AcAxPoint3d pt;
        pC->center(pt);
        pt.setVariant(*pVal);
        . . .
}
```

Since the `AcAxPoint3d` constructor, which takes `VARIANT` type as argument, throws an exception, you must handle it. This is achieved by wrapping the above two statements with a `try-catch` block.

2.8 Final code of put- and get-methods

Please, copy and paste the following code.

To *Rings.h*:

```
public:
    //IRings
     STDMETHODIMP get_rings(short *pVal);
     STDMETHODIMP put_rings(short newVal);
     STDMETHODIMP get_center(VARIANT *pVal);
     STDMETHODIMP put_center(VARIANT newVal);
     STDMETHODIMP get_radius(double *pVal);
     STDMETHODIMP put_radius(double newVal);
     STDMETHODIMP get_normal(VARIANT *pVal);
     STDMETHODIMP put_normal(VARIANT newVal);
```

To *Rings.cpp*:

```
STDMETHODIMP CRings::get_rings(short *pVal)
{
    AcAxDocLock lock(m_objId);
    if(lock.lockStatus())
        return E_ACCESSDENIED;
    AcDbObjectPointer<AsdkRings> pC(m_objId,AcDb::kForRead);
    if(pC.openStatus()!=Acad::eOk)
        return E_ACCESSDENIED;

    (Adesk::UInt8&)*pVal = pC->get_rings();
    return S_OK;
}


STDMETHODIMP CRings::put_rings(short newVal)
{
    AcAxDocLock lock(m_objId);
    if(lock.lockStatus())
        return E_ACCESSDENIED;
    AcDbObjectPointer<AsdkRings> pC(m_objId,AcDb::kForWrite);
    if(pC.openStatus()!=Acad::eOk)
        return E_ACCESSDENIED;
    pC->put_rings(newVal);
    return S_OK;
}


STDMETHODIMP CRings::get_center(VARIANT *pVal)
{
    try
    {
        AcAxDocLock lock(m_objId);
        if(lock.lockStatus())
            return E_ACCESSDENIED;
        AcDbObjectPointer<AsdkRings> pC(m_objId,AcDb::kForRead);
        if(pC.openStatus()!=Acad::eOk)
            return E_ACCESSDENIED;
        AcAxPoint3d pt = pC->get_center();
        pt.setVariant(*pVal);
    }
    catch(const HRESULT hr)
    {
        Error("An error occurred. Check the input params.");
        return hr;
    }
    return S_OK;
}


STDMETHODIMP CRings::put_center(VARIANT newVal)
{
    try
    {
        AcAxDocLock lock(m_objId);
        if(lock.lockStatus())
            return E_ACCESSDENIED;
        AcDbObjectPointer<AsdkRings> pC(m_objId,AcDb::kForWrite);
        if(pC.openStatus()!=Acad::eOk)
            return E_ACCESSDENIED;
        AcAxPoint3d pt(newVal);
        pC->put_center(pt);
    }
    catch(const HRESULT hr)
    {
        Error("An error occurred. Check the input params.");
        return hr;
    }
    return S_OK;
}


STDMETHODIMP CRings::get_radius(double *pVal)
{
    AcAxDocLock lock(m_objId);
    if(lock.lockStatus())
        return E_ACCESSDENIED;
    AcDbObjectPointer<AsdkRings> pC(m_objId,AcDb::kForRead);
    if(pC.openStatus()!=Acad::eOk)
        return E_ACCESSDENIED;

    *pVal = pC->get_radius();
    return S_OK;
}


STDMETHODIMP CRings::put_radius(double newVal)
{
    AcAxDocLock lock(m_objId);
    if(lock.lockStatus())
        return E_ACCESSDENIED;
    AcDbObjectPointer<AsdkRings> pC(m_objId,AcDb::kForWrite);
    if(pC.openStatus()!=Acad::eOk)
        return E_ACCESSDENIED;

    pC->put_radius(newVal);
    return S_OK;
}


STDMETHODIMP CRings::get_normal(VARIANT *pVal)
{
    try
    {
        AcAxDocLock lock(m_objId);
        if(lock.lockStatus())
            return E_ACCESSDENIED;
        AcDbObjectPointer<AsdkRings> pC(m_objId,AcDb::kForRead);
        if(pC.openStatus()!=Acad::eOk)
```

```
            return E_ACCESSDENIED;

         AcAxPoint3d pt;
         AcGeVector3d v = pC->get_normal();
         pt.set(v.x,v.y,v.z);
         pt.setVariant(*pVal);
      }
      catch(const HRESULT hr)
      {
         Error("An error occurred. Check the input params.");
         return hr;
      }
      return S_OK;
   }


   STDMETHODIMP CRings::put_normal(VARIANT newVal)
   {
      try
      {
         AcAxDocLock lock(m_objId);
         if(lock.lockStatus())
            return E_ACCESSDENIED;
         AcDbObjectPointer<AsdkRings> pC(m_objId,AcDb::kForWrite);
         if(pC.openStatus()!=Acad::eOk)
            return E_ACCESSDENIED;

         AcAxPoint3d pt(newVal);
         pC->put_normal(pt.asVector());
      }
      catch(const HRESULT hr)
      {
         Error("An error occurred. Check the input params.");
         return hr;
      }
      return S_OK;
   }
```

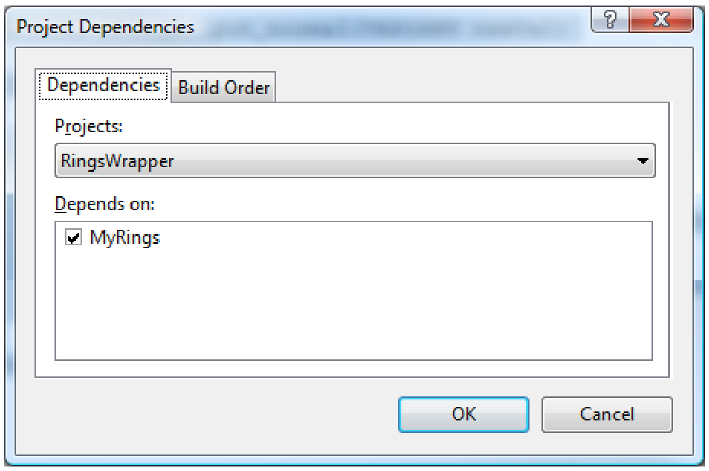
2.9 Additional Project Settings for the Wrapper module

Since we are calling member functions of our custom object, this module will need to link with the import library of the DBX module which implements the custom object.

As we have both projects in our solution, we can simply set a dependency between the projects. That will ensure that whenever we modify the custom object project, Visual Studio will compile the DBX project first and link the generated library into your COM project.

In the **Solution Explorer** right-click the **RingsWrapper** project and select **Project Dependencies**.

On the **Dependencies** tab check the **MyRings** item:



Click **OK**.

2.10 Build and load the COM Wrapper module

Now you are ready to build the project. Go to the main menu -> **Build** -> **Build Solution**.

After successfully building the project, open the AutoCAD application and load the DBX module first. Check if there are any error messages in the Command area.

It is necessary to load the ARX module once manually in order to let it register itself calling `DllRegisterServer()`. So, please, load the ARX module. Once the COM server is registered, it is no longer necessary to load the ARX module manually each time when you want to call it via COM. The runtime system of COM will make sure that the ARX is loaded. Be careful, however, when your ARX module is loaded by the COM runtime your `On_kInitAppMsg()` is not executed.

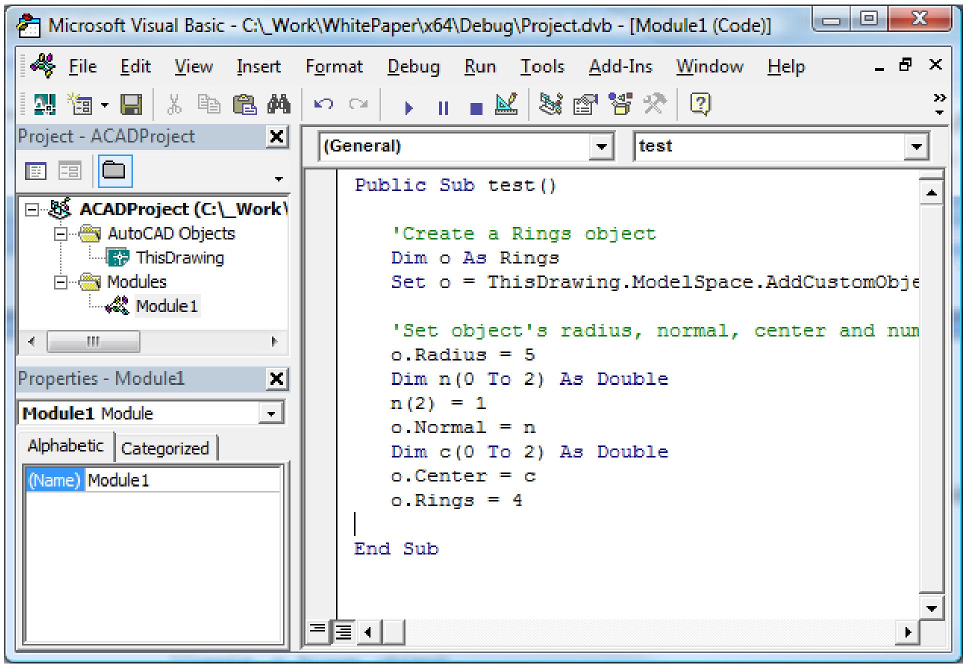
3 Testing the COM wrapper

If everything goes well, let's try to create our Rings object using Visual Basic for Applications.

Enter "`vbaide`" in the AutoCAD command prompt. Visual Basic application is launched by that command.

Now from the **Insert** pull-down menu select **Module**. You will be presented with a module section. Set the cursor in the module window and copy-paste the following macro into it:

```
Public Sub test()

   'Create a Rings object
   Dim o As Rings
   Set o = ThisDrawing.ModelSpace.AddCustomObject("AsdkRings")

   'Set object's radius, normal, center and number of rings
   o.Radius = 5
   Dim n(0 To 2) As Double
   n(2) = 1
   o.Normal = n
   Dim c(0 To 2) As Double
   o.Center = c
   o.Rings = 4

End Sub
```

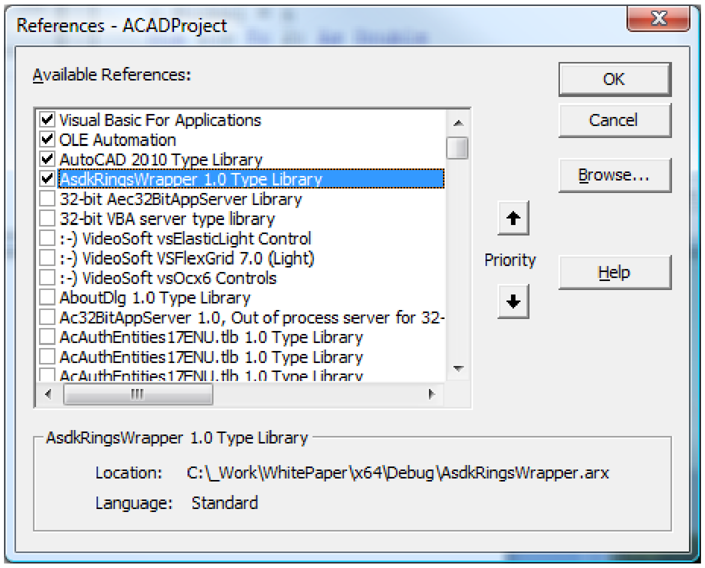


```
Public Sub test()

    'Create a Rings object
    Dim o As Rings
    Set o = ThisDrawing.ModelSpace.AddCustomObje

    'Set object's radius, normal, center and num
    o.Radius = 5
    Dim n(0 To 2) As Double
    n(2) = 1
    o.Normal = n
    Dim c(0 To 2) As Double
    o.Center = c
    o.Rings = 4

End Sub
```

This VBA macro exercises the `Rings` object.

Before executing this macro, create a reference to the `AsdkRingsWrapper`. To do that, click the **Tools** menu and select the **Reference** menu item.

You will be presented with the following dialog:



- [✓] Visual Basic For Applications
- [✓] OLE Automation
- [✓] AutoCAD 2010 Type Library
- [✓] AsdkRingsWrapper 1.0 Type Library
- [ ] 32-bit Aec32BitAppServer Library
- [ ] 32-bit VBA server type library
- [ ] :-) VideoSoft vsElasticLight Control
- [ ] :-) VideoSoft VSFlexGrid 7.0 (Light)
- [ ] :-) VideoSoft vsOcx6 Controls
- [ ] AboutDlg 1.0 Type Library
- [ ] Ac32BitAppServer 1.0, Out of process server for 32-
- [ ] AcAuthEntities17ENU.tlb 1.0 Type Library
- [ ] AcAuthEntities17ENU.tlb 1.0 Type Library
- [ ] AcAuthEntities17ENU.tlb 1.0 Type Library

AsdkRingsWrapper 1.0 Type Library

Location: C:_Work\WhitePaper\x64\Debug\AsdkRingsWrapper.arx

Language: Standard

Find the `AsdkRingsWrapper` library and check it. Click **OK**.

Now you are ready to run the macro. Run the macro and, hopefully, you will get our Rings object in the AutoCAD window.

4 Adding OPM feature

This section describes how to implement the Property Inspector API in our COM wrapper, if you checked the respective checkboxes when you were creating a COM Wrapper in section 2.3.

4.1 Development of OPM properties

If a custom object does not implement a COM object wrapper for itself, the `AcAxGetIUnknownOfObject()` function will generate a default wrapper that implements the methods of `IAcadEntity` or `IAcadObject`, depending on if the underlying object can be cast to an `AcDbEntity`. OPM then uses this object to display the Color, Layer, Line type, and Line weight properties, also known as the entity common properties. `ICategorizeProperties`, `IPerPropertyBrowsing`, and `IOPMPropertyExtension` are the 'flavoring' interfaces. For more details about these interfaces refer to the "Property Inspector and Properties Palette APIs" section in the ObjectARX Developer Guide.

The main purpose of `IOPMPropertyExpander` interface is to allow one property to be broken out into several properties in the Property Inspector.

When we created a COM wrapper, checking of the check boxes in the **OPM support** section created some additional functions in our implementation class (i.e. *Rings.cpp*), like this one:

```
//IOPMPropertyExpander
STDMETHODIMP CRings::GetElementValue(
   /* [in] */ DISPID dispID,
   /* [in] */ DWORD dwCookie,
   /* [out] */ VARIANT * pVarOut)
{
    //TO DO: Implement this function.
    return E_NOTIMPL;
}
```

In the *Rings.h* header file you can notice that our `CRings` class is additionally inherited from `IOPMPropertyExtensionImpl<CRings>` and `IOPMPropertyExpander`.

```
class ATL_NO_VTABLE CRings :
      public CComObjectRootEx<CComSingleThreadModel>,
      public CComCoClass<CRings, &CLSID_Rings>,
      public ISupportErrorInfo,
      public IOPMPropertyExtensionImpl<CRings>,
      public IOPMPropertyExpander,
   ...
```

There are also a few more COM interfaces in the map:

```
BEGIN_COM_MAP(CRings)
      COM_INTERFACE_ENTRY(IRings)
      COM_INTERFACE_ENTRY(IDispatch)
      COM_INTERFACE_ENTRY(ISupportErrorInfo)
      COM_INTERFACE_ENTRY(IConnectionPointContainer)
      COM_INTERFACE_ENTRY(IOPMPropertyExtension)
      COM_INTERFACE_ENTRY(ICategorizeProperties)
      COM_INTERFACE_ENTRY(IPerPropertyBrowsing)
      COM_INTERFACE_ENTRY(IOPMPropertyExpander)
      COM_INTERFACE_ENTRY(IAcadBaseObject)
      COM_INTERFACE_ENTRY(IAcadBaseObject2)
      COM_INTERFACE_ENTRY(IAcadObject)
```

```
        COM_INTERFACE_ENTRY(IAcadEntity)
        COM_INTERFACE_ENTRY(IRetrieveApplication)
    END_COM_MAP()
```

And there is some other OPM-related code.

In addition, there are prototype declarations for useful functions like `GetElementValue()`, `SetElementValue()`, `GetElementStrings()`, `GetElementGrouping()` and `GetGroupCount()`. These functions are called for each of the variables being processed. The `DISPID` parameter is passed to each of these functions, this `DISPID` is the id of the property. For details about these functions refer to the `IOPMPropertyExpander` interface in the ObjectARX help.

FUNCTION NAME	PURPOSE
`GetGroupCount()`	Get the number of item in each element
`GetElementGrouping()`	Get the number of elements in the property
`GetElementStrings()`	Get the element string names
`GetElementValue()`	Get the element value
`SetElementValue()`	Set the element value

The following algorithm (in pseudo code) is used to display the point and vertices values in the Property Inspector (`DispId_of_Property` is the `DISPID` for the property, `Receive_VertexCount` is the variable that receives the number of vertices from the COM wrapper).

```
If(SUCCEEDED == GetElementStrings(DispId_of_Property, &opmlpstr, &opmdword))
{
    GetElementGrouping(DispId_of_Property, &grouping);

    // If there is no grouping we just expand the property into opmlpstr.cElems
    // properties, otherwise we create a spinner control with the property name
    //and opmlpstr.cElems properties below it.
    if(0 == grouping )
    {
       count = opmlpstr.cElems;
    }
    else
    {
       addCount = grouping;
       create_spin_control();
       GetGroupCount(DispId_of_Property, &Receive_VertexCount);
    }
    Display_each_element();
}
```

4.2 *Modify the OPM override functions*

Modify the code in Rings implementation (*Rings.cpp*) for these functions as follows:

```
//IOPMPropertyExpander
STDMETHODIMP CRings::GetElementValue (DISPID dispID,
                                      DWORD dwCookie, VARIANT *pVarOut)
{
    if (dispID == 2) // Center
    {
        CComVariant var;
        get_center(&var);
        AcAxPoint3d pt(var);
        pVarOut->vt = VT_R8;
        pVarOut->dblVal = pt[dwCookie];
        return S_OK;
    }
    else if (dispID == 4) // Normal
    {
        if ( pVarOut == NULL )
            return (E_POINTER) ;
        CComVariant var;
        get_normal(&var);
        AcAxPoint3d pt(var);
        pVarOut->vt = VT_R8;
        pVarOut->dblVal = pt[dwCookie];
        return S_OK;
    }
    return (E_NOTIMPL) ;
}


STDMETHODIMP CRings::SetElementValue (DISPID dispID,
                                      DWORD dwCookie, VARIANT VarIn)
{
    if (dispID == 2)
    {
        CComVariant var;
        get_center(&var);
        AcAxPoint3d pt(var);
        pt[dwCookie] = VarIn.dblVal;
        pt.setVariant(var);
        put_center(var);
        return S_OK;
    }
    else if (dispID == 4)
    {
        CComVariant var;
        get_normal(&var);
        AcAxPoint3d pt(var);
        pt[dwCookie] = VarIn.dblVal;
        pt.setVariant(var);
        put_normal(var);
        return S_OK;
    }
    return (E_NOTIMPL) ;
}
```

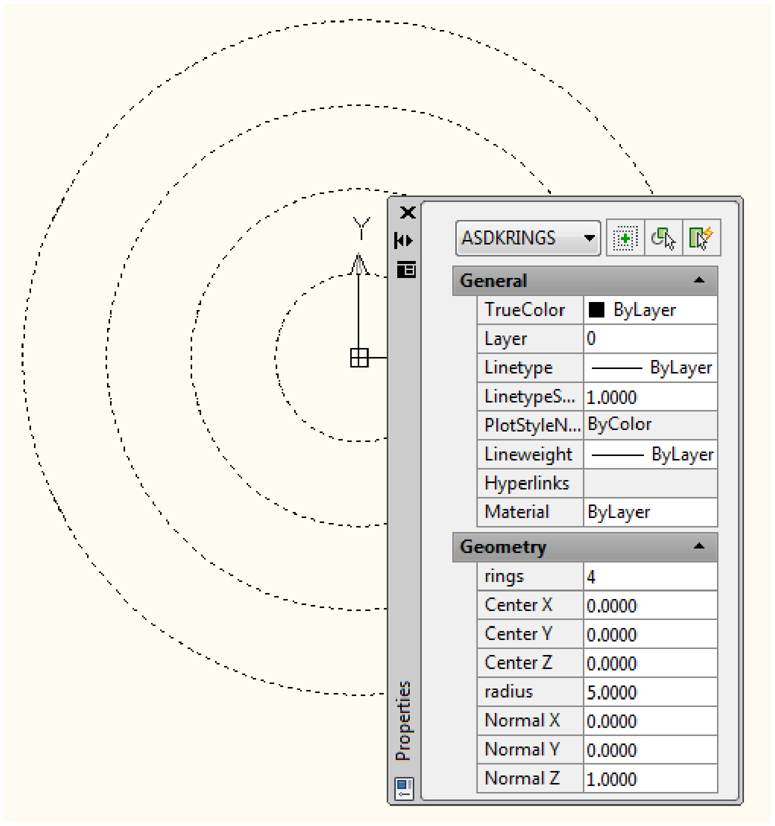
```
STDMETHODIMP CRings::GetElementStrings (DISPID dispID,
                                        OPMLPOLESTR __RPC_FAR *pCaStringsOut,
OPMDWORD __RPC_FAR *pCaCookiesOut)
{
    if ( pCaStringsOut == NULL || pCaCookiesOut == NULL )
        return (E_POINTER) ;
     if (dispID == 2)
    {
        pCaStringsOut->cElems = 3;
        pCaStringsOut->pElems = (LPOLESTR*)CoTaskMemAlloc(sizeof(LPOLESTR)*3);
        pCaStringsOut->pElems[0] = SysAllocString(L"Center X");
        pCaStringsOut->pElems[1] = SysAllocString(L"Center Y");
        pCaStringsOut->pElems[2] = SysAllocString(L"Center Z");
        pCaCookiesOut->cElems = 3;
        pCaCookiesOut->pElems = (DWORD*)CoTaskMemAlloc(sizeof(DWORD)*3);
        for (int i=0;i<3;i++)
            pCaCookiesOut->pElems[i] = i;
        return S_OK;
    }
     else if (dispID == 4)
    {
        pCaStringsOut->cElems = 3;
        pCaStringsOut->pElems = (LPOLESTR*)CoTaskMemAlloc(sizeof(LPOLESTR)*3);
        pCaStringsOut->pElems[0] = SysAllocString(L"Normal X");
        pCaStringsOut->pElems[1] = SysAllocString(L"Normal Y");
        pCaStringsOut->pElems[2] = SysAllocString(L"Normal Z");
        pCaCookiesOut->cElems = 3;
        pCaCookiesOut->pElems = (DWORD*)CoTaskMemAlloc(sizeof(DWORD)*3);
        for (int i=0;i<3;i++)
            pCaCookiesOut->pElems[i] = i;
        return S_OK;
    }
     return E_NOTIMPL;
}


STDMETHODIMP CRings::GetElementGrouping (DISPID dispID,
                                         short *groupingNumber)
{
    if ( groupingNumber == NULL )
        return (E_POINTER) ;
    if (dispID == 2 || dispID == 4)
    {
        *groupingNumber = 0;
        return S_OK;
    }
    return E_NOTIMPL;
}


STDMETHODIMP CRings::GetGroupCount (DISPID dispID, long *nGroupCnt)
{
    if ( nGroupCnt == NULL )
        return (E_POINTER) ;
     if (dispID == 2 || dispID == 4)
    {
        *nGroupCnt = 3;
        return S_OK;
    }
    return E_NOTIMPL;
}
```

In Addition, modify the code in *Rings.h* at the OPM property map section as follows:

```
// IOPMPropertyExtension
BEGIN_OPMPROP_MAP()
   OPMPROP_ENTRY(0, 0x00000001, PROPCAT_Geometry, 0, 0, 0, "", 0, 1, IID_NULL, IID_NULL, "")
   OPMPROP_ENTRY(0, 0x00000002, PROPCAT_Geometry, 0, 0, 0, "", 0, 1, IID_NULL, IID_NULL, "")
   OPMPROP_ENTRY(0, 0x00000003, PROPCAT_Geometry, 0, 0, 0, "", 0, 1, IID_NULL, IID_NULL, "")
   OPMPROP_ENTRY(0, 0x00000004, PROPCAT_Geometry, 0, 0, 0, "", 0, 1, IID_NULL, IID_NULL, "")
END_OPMPROP_MAP()
```

That's all you have to do for supporting the Property Inspector in our Custom Entity.

I hope, now you can see our Custom Entity in AutoCAD and also see and modify it using the Properties window:



General	
TrueColor	■ ByLayer
Layer	0
Linetype	——— ByLayer
LinetypeS...	1.0000
PlotStyleN...	ByColor
Lineweight	——— ByLayer
Hyperlinks	
Material	ByLayer
Geometry	
rings	4
Center X	0.0000
Center Y	0.0000
Center Z	0.0000
radius	5.0000
Normal X	0.0000
Normal Y	0.0000
Normal Z	1.0000

Thus, we have shown how COM Wrappers are created.

Good luck in your development!