

COM Interoperability

Source <http://www.dotnetcoders.com/web/Articles/ShowArticle.aspx?article=55>

What is the need for Interoperability?

COM components have different internal architecture from .NET components hence they are not innately compatible. Most organizations, which have built their enterprise applications on COM objects for their middle tier services, cannot write off the investments on these solutions. These legacy components ought to be exploited by managed code in .NET framework. This is where Interoperability pitches in; it's a Runtime Callable Wrapper (RCW) that translates specific calls from managed clients into COM specific invocation requests on unmanaged COM components. The method call on RCW will make .NET components believe that they are talking to just another .NET component.

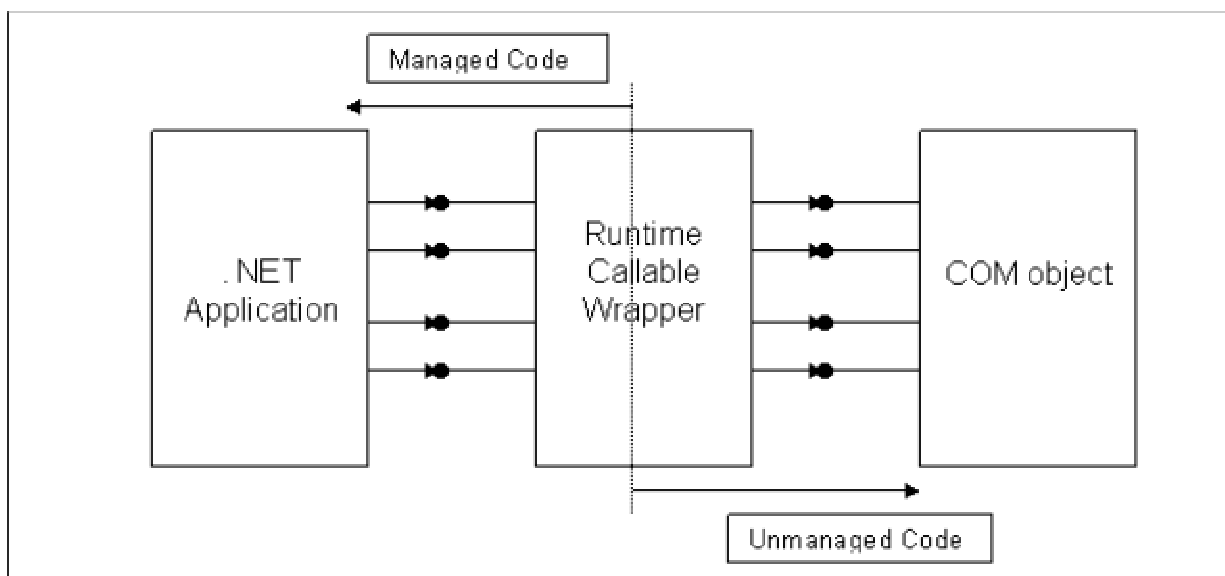
Before we move on to the core concepts, lets have small primer on COM.

What is COM?

COM stands for Component Object Model, which is binary specification for software code re-use. It imposes a standard for the interfaces through which client code talks to component classes. The component's IUnknown interface helps to maintain a reference count of the number of clients using the component. When this count drops down to zero, the component is unloaded. All components should implement the IUnknown interface. The reference count is maintained through IUnknown::AddRef() & IUnknown::Release() methods, interface discovery is handled through IUnknown::QueryInterface().

What is Runtime Callable Wrapper?

.NET applications communicate with a COM component through a managed wrapper of the component called a Runtime Callable Wrapper. It acts as managed proxy to the unmanaged COM component.



When we make a method call, it goes onto RCW and not the object itself. RCW manages the lifetime management of the COM component.

How does Component Binding work?

Binding refers to information on methods, properties, events etc that client needs to know about the object. We still can use the good old technique of bindings in .NET interoperability viz., Early & Late bindings.

- Early Binding: Clients obtain compile time type information from the component's type library
- Late Binding: Clients lack rich type information of the object, it is known at runtime.

How do we implement COM Interoperability?

We can implement it through the following steps:

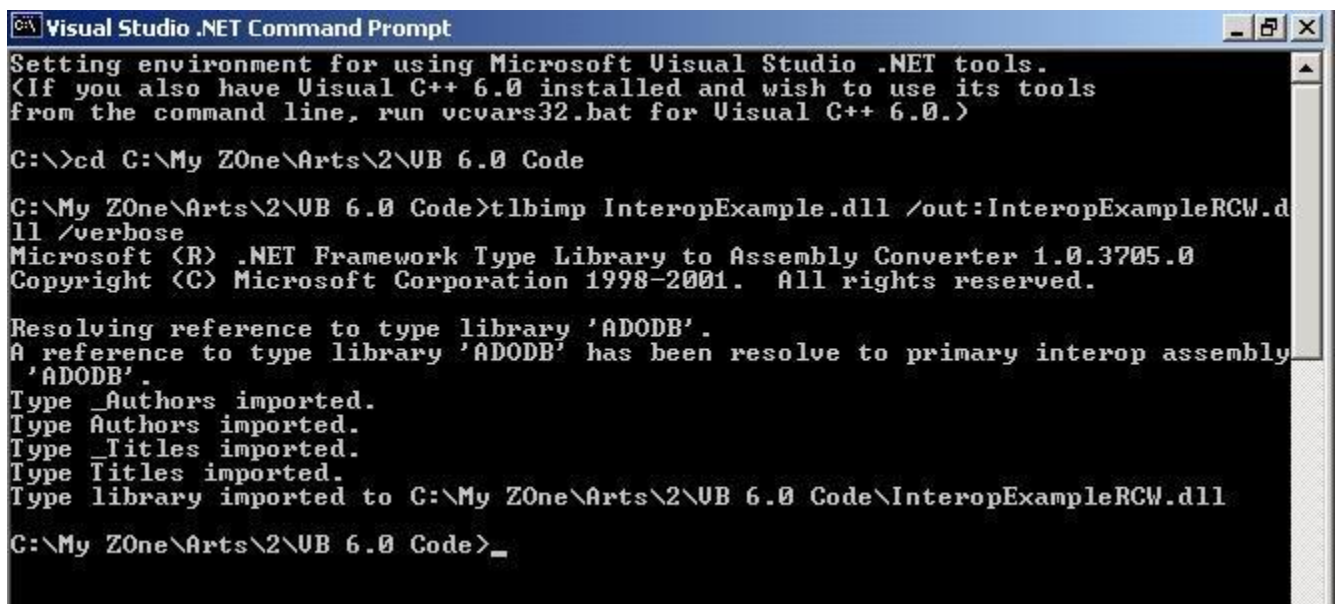
1. Create Runtime Callable Wrapper out of COM component.
2. Reference the metadata assembly Dll in the project and use its methods & properties.

Step 1:

There are two ways to generate managed metadata wrapper:

- Using Type Library Importer utility
- VS.NET IDE

Type Library Importer (tlbimp.exe) is command line syntax, which converts COM specific type definition in a COM type library into equivalent definitions for .NET wrapper assembly. By default utility gives wrapper assembly same name as COM dll.



```
Visual Studio .NET Command Prompt
Setting environment for using Microsoft Visual Studio .NET tools.
<If you also have Visual C++ 6.0 installed and wish to use its tools
from the command line, run vcvars32.bat for Visual C++ 6.0.>

C:\>cd C:\My ZOne\Arts\2\VB 6.0 Code

C:\My ZOne\Arts\2\VB 6.0 Code>tlbimp InteropExample.dll /out:InteropExampleRCW.d
ll /verbose
Microsoft (R) .NET Framework Type Library to Assembly Converter 1.0.3705.0
Copyright (C) Microsoft Corporation 1998-2001. All rights reserved.

Resolving reference to type library 'ADODB'.
A reference to type library 'ADODB' has been resolve to primary interop assembly
'ADODB'.
Type _Authors imported.
Type Authors imported.
Type _Titles imported.
Type Titles imported.
Type library imported to C:\My ZOne\Arts\2\VB 6.0 Code\InteropExampleRCW.dll

C:\My ZOne\Arts\2\VB 6.0 Code>_
```

The above example generates metadata assembly with the name "InteropExampleRCW.dll" out of COM component "InteropExample.dll" using the following syntax at VS.NET command prompt.

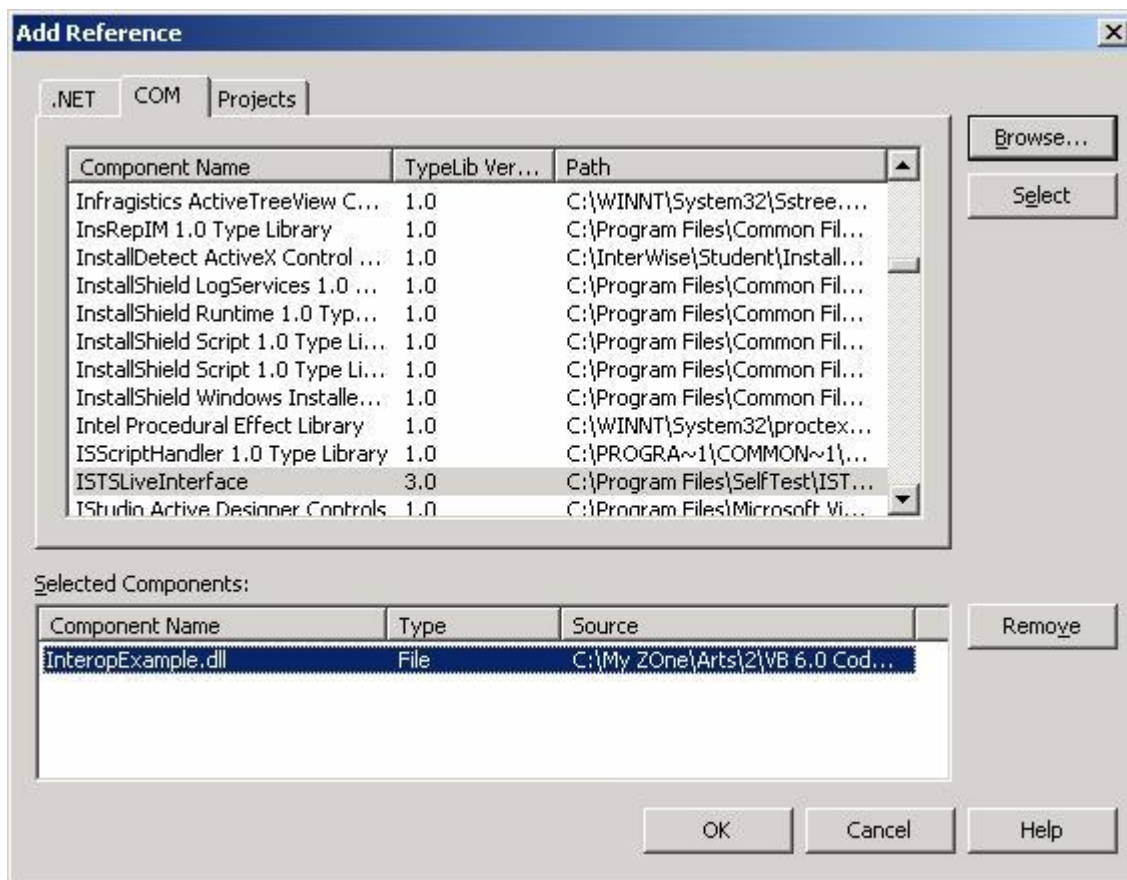
```
tlbimp InteropExample.dll /output:InteropExampleRCW.dll /verbose
```

Note: it internally resolves ADODB references in the COM. Through "out" argument we can specify desired assembly name.

Type library importer interrogates COM dll's type library and translates the information therein into .NET format. The metadata assembly so generated contains wrapper classes that can be used any .NET client e.g. C# windows clients. RCW is created on the fly whenever component is created & it acts like managed types to COM specific data types.

VS.NET IDE also helps us generate metadata assembly:

Click on Project -> Add reference -> COM tab

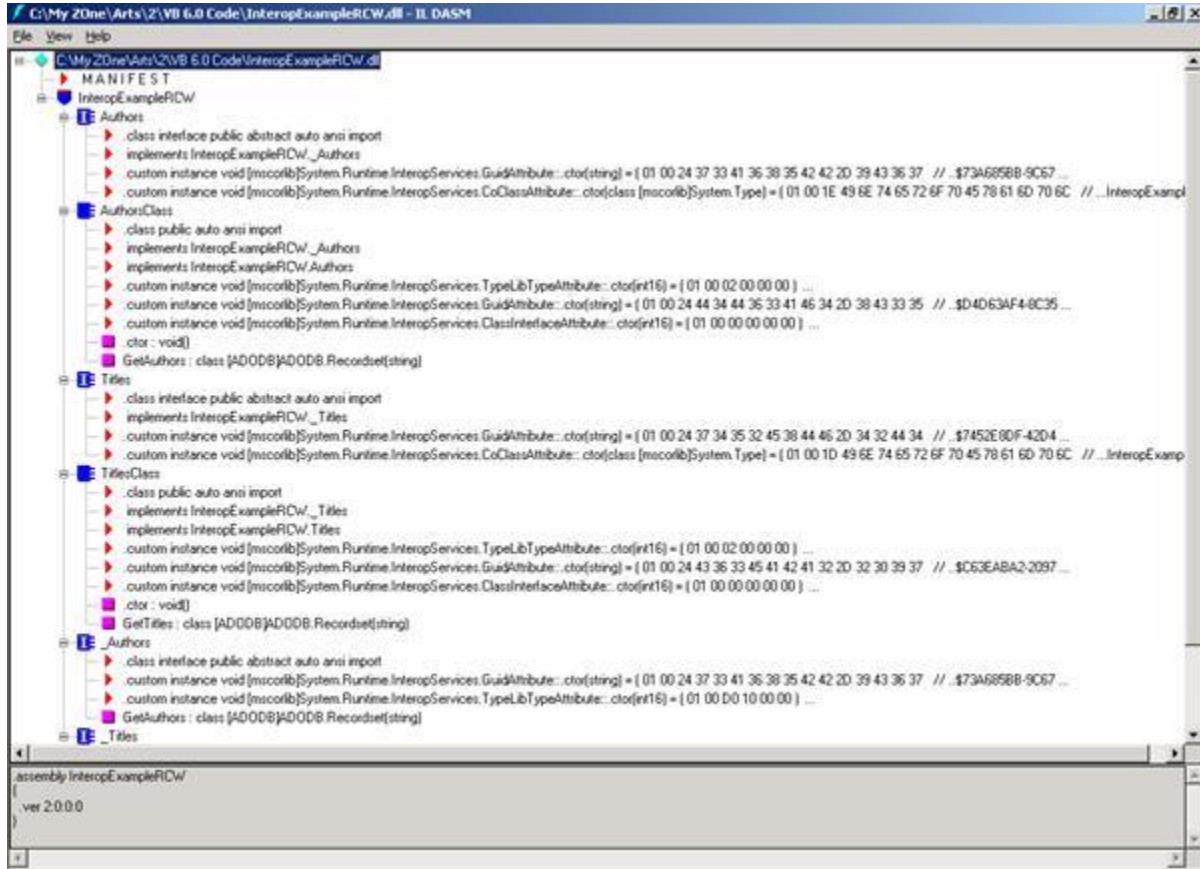


The tab lists registered components on local machine, select the desired COM dll and add to list of selected components. VS.NET automatically generates metadata assembly putting the classes provided by that component into a namespace with the same name as COM dll.

What is the structure of the Wrapper Assembly?

For each class imported into wrapper assembly, two wrapper classes are generated. COM specific information can be viewed using MSIL Disassembler utility (ildasm .exe) at VS.NET command prompt:

```
ildasm InteropExampleRCW.dll
```



InteropExample.dll (developed in VB6.0) component has the following public classes:

- Authors
- Titles

The generated assembly has four classes:

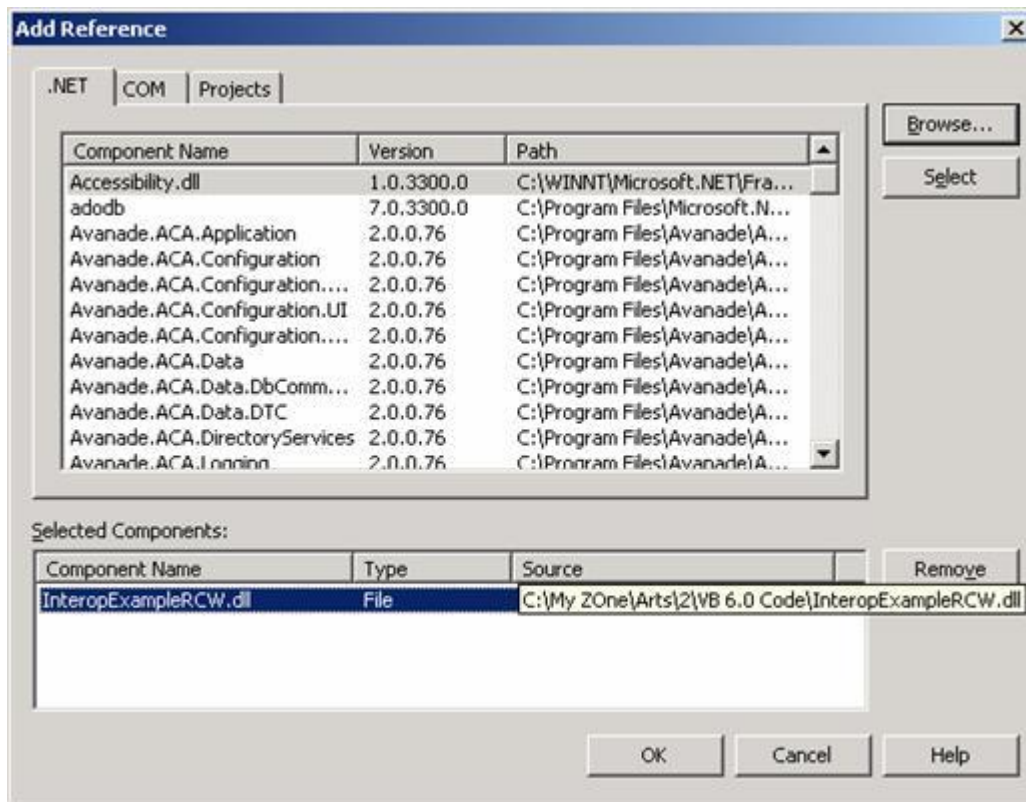
1. Authors
2. AuthorsClass
3. Titles
4. TitlesClass

First is the interface having the same GUID as the original COM class, second is concrete class and whose instance is to be created. Concrete class is suffixed with the word "Class". Concrete class implements all the interfaces that are supported by the original COM class.

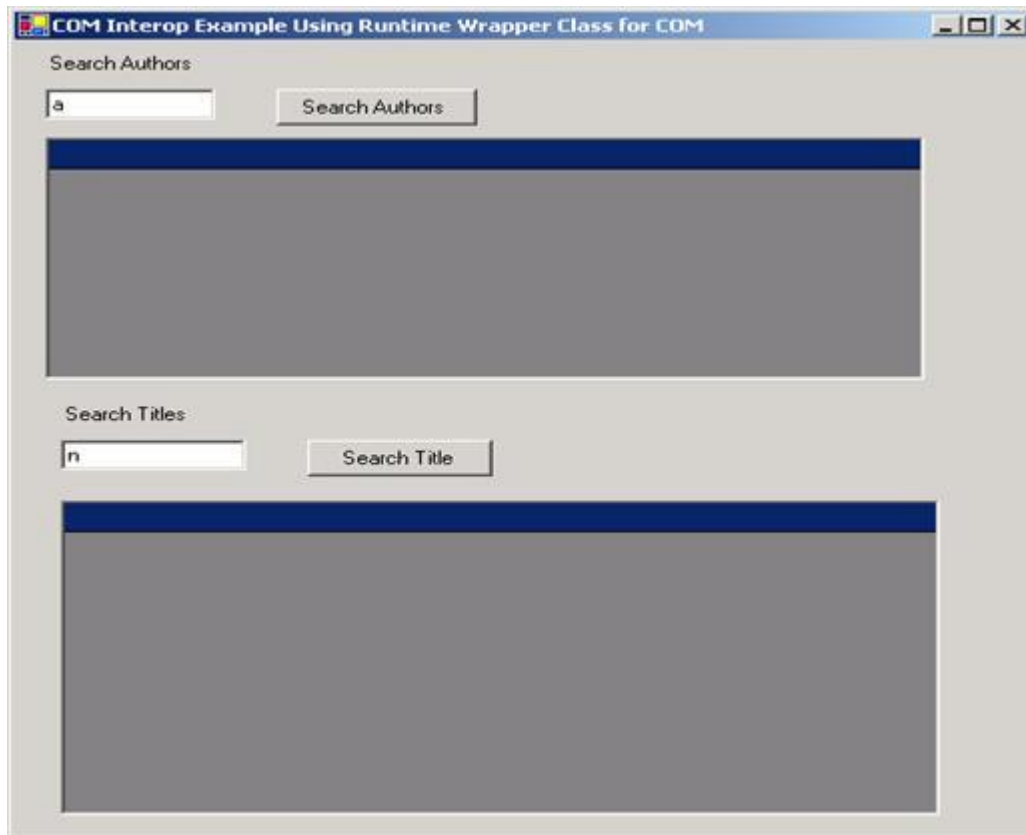
All the generated types are place under the single namespace InteropExampleRCW

Step 2:

Reference the metadata assembly Dll in the project and use its methods & properties:



Create a new C# or VB.NET Windows application project. Drag following label, textbox, datagrid controls on the form, and name them accordingly, the form will appear as follows:



Import the required assemblies:

```
using System.Data;  
using System.Data.OleDb;
```

Include following code in appropriate click events:

To use Authors search from AuthorClass, have code for Search Author button click event as:

```
private void btnSearchAuthors_Click(object sender, System.EventArgs e)  
{  
    //create an instance of AuthorsClass from wrapper assembly  
    InteropExampleRCW.AuthorsClass myAuthorRCW=new  
    InteropExampleRCW.AuthorsClass();  
  
    DataSet dsAuthorList=new DataSet("Authors");  
    OleDbDataAdapter daAuthRecs=new OleDbDataAdapter();  
    ADODB.Recordset rsAuthors=new ADODB.Recordset();  
  
    rsAuthors=myAuthorRCW.GetAuthors(txtAuthors.Text.ToString());  
  
    //invoke method from the RCW  
    daAuthRecs.Fill(dsAuthorList,rsAuthors,"Authors");  
  
    dataGridAuthors.SetDataBinding(dsAuthorList,"Authors");  
}
```

To use Titles search from TitleClass, code for Search Title button click event:

```
private void btnSearchTitle_Click(object sender, System.EventArgs e)
{
    InteropExampleRCW.TitlesClass myTitlesRCW = new
    InteropExampleRCW.TitlesClass();
    DataSet dsTitleList = new DataSet("Titles");

    OleDbDataAdapter daTitleRecs = new OleDbDataAdapter();
    ADODB.Recordset rsTitles = new ADODB.Recordset();
    rsTitles=myTitlesRCW.GetTitles(txtTitle.Text.ToString());

    daTitleRecs.Fill(dsTitleList,rsTitles,"Titles");
    dataGridTitle.SetDataBinding(dsTitleList,"Titles");
}
```

COM Interop Example Using Runtime Wrapper Class for COM

Search Authors

green Search Authors

	au_id	au_fname	au_lname	phone	address	city
▶	213-46-8915	Marjorie	Green	415 986-7020	309 63rd St.	Oakland
	527-72-3246	Morningstar	Greene	615 297-2723	22 Graybar H	Nashville
*						

Search Titles

computers Search Title

	title_id	title	type	price
▶	BU1111	Cooking with Computers: Surreptitious	business	11.95
	BU7832	Straight Talk About Computers	business	19.99
*				

How Do I Release COM objects?

Runtime Callable Wrapper is managed creation itself; hence its lifetime is controlled by Common Language Runtime. The COM component is freed from memory when the garbage collector calls the `Finalize()` method on RCW. Internally RCW calls `Release()` of `IUnknown` interface on COM object.

To explicitly remove COM objects from memory invoke the static method on Marshal class in `System.Runtime.InteropServices` namespace:

```
using System.Runtime.InteropServices;
...
Marshal.ReleaseComObject(myAuthorRCW);
```

~ ~ ~ End of Article ~ ~ ~