Using COM and COM+ in .NET

-P/invoke: The mechanism to call unmanaged functions in Win32 DLLs from .NET

Ways in which .NET is better than COM:

-For distributed applications communication, .NET uses ".NET remoting" instead of DCOM. DCOM supports only one binary protocol, but .NET remoting supports different kinds of transport, different encoding (SOAP and binary) and different security protocols.
-SOAP (Simple Object Access Protocol) uses XML over HTTP to allow distributed communications (more firewall friendly)

COM interop enable the integration of .NET and COM code in both directions

Runtime Callable Wrappers RCW

-COM components cannot be used directly in .NET code
- RCW needs to be used, and it is responsible for lifetime and interface management. It can wrap around ActiveX control to allow usage in Designer. It also allows .NET code to use *reflection* to discover the COM object's abilities

COM Callable Wrapper CCW

- CCW can be used to make a .NET object look and behave like a COM component by providing COM interfaces such as IUnknown.
-COM holds raw pointers to things, so CCW needs to ensure the pointers remain valid even when .NET objects are moved around in the managed heap.
-System.Runtime.InteropServices.Marshal class provides a number of methods that help when interacting with unmanaged code.

Differences between COM and .NET

-Location of components
  -COM: registry + GUID (128-bits integer)
  -.NET: GAC (shared among programs) or same directory as the program (private assembly).
    -Assemblies are identified by its name, and a more fully qualified name including the namespace it resides, version and culture info. If assembly lives in GAC, it must be strongly signed to get a unique identification.

Other differences:

	COM	.NET
Constructor	No constructor; Custom initialization. When using CCW, the .NET component must have default constructor only	
Finalization	Manual release of resources	Non-deterministic. Need to call **ReleaseComObject** to manually release in managed code
Error handling	HRESULT	Exceptions
Visibility	Interface can contain public methods only. When using CCW, use attribute **[ComVisible(false)]**	

Apartment		Exported .NET types will have a registry ThreadingModel value of "Both". When access COM components, the client can be set to have the desired apartment with the following syntax:
		`class Class1`
		`{`
		`  [STAThread]`
		`  static void Main(string[] args)`

		{
		...
		}
		}

Events handling	ConnectionPoint	Delegates

Strength of COM+

-Transactional (only commit if successful, can rollback otherwise)
-Asynchronous data process with message queue. Client and server don't have to be available at the same time
-Security support on component, interface, and method. Role-based security.

Interop Assemblies (RCW and CCW)

-Three ways of producing interop assemblies, for representing a COM object to .NET and vice versa:
  -VS Studio
  -TlbImp.exe
  -System.Runtime.InteropServices.TypeLibConverter class
-PIA (Primary Interop Assemblies): Official release of the wrapper for the COM object from the vendor. This prevents multiple copies of the COM object represented by different interop assemblies (which .NET treat them as different), and prevents the problem of not being able to pass interface pointer between the two because they are regarded as different.
  -Marked with PrimaryInteropAssembly attribute
  -Strong name
  -Maintain the same GUID
  -Usually installed in GAC because the entire machine should use the same copy

TlbImp.exe (Creating RCW)

`-tlbimp libname.ext /out:Interop.libname.dll /namespace:libname /sysarray` performs the same thing as what Visual Studio does
`-/reference:Interop.SomeComponent.dll` is used to specify other COM objects referenced. If not specified, TlbImp will search through the registry and recursive create Interop assemblies for the COM objects referenced.
`-/transform:dispret` is needed if `dispinterface` is used in the COM object, and the wrapper should convert `out` parameters into return values

System.Runtime.InteropServices.TypeLibConverter

-Three methods: ConvertAssemblyToTypeLib, `ConvertTypeLibToAssembly` and `GetPrimaryInteropAssembly`

Strong Name

-Public key + digital signature(determined from private key)
  -Make the assembly unique/integrity check (source of the assembly is verifiable)
  -Other people cannot produce newer version without private key
-Clients that use a strong name-ed assembly will include the public key into the program manifest so that the client can check at run-time that it is using the right assembly

Signing Assemblies

1. Create a private/public key pair with `sn -k mykeys.snk`
2. Include `[assembly: AssemblyKeyFile(@"..\..\mykeys.snk")]` in AssemblyInfo.cs

Delayed Signing (Partial Signing)

-Without signing with the real private/public key during development

1. Extract the public key with `sn -p mykeys.snk mypublickey.snk`
2. Include `[assembly: AssemblyKeyFile(@"..\..\mykeys.snk")]` in AssemblyInfo.cpp

```
[assembly:AssemblyDelaySignAttribute(true)];
[assembly:AssemblyKeyFileAttribute("mypublickey.snk")];
```

gacutil and regasm

-gacutil /i myassembly.dll installs myassembly.dll into GAC. The assembly can be an interop assembly generated through any of the three methods

-An assembly with *PrimaryInteropAssembly* attribute (genereated by using /primary option with TlbImp.exe) does not have to live in GAC. .NET framework can use registry to locate it. To add the appropriate entries to registry, use `regasm myPrimaryAssembly.dll`

Note: The Assembly Registration (Regasm) tool reads the metadata within an assembly and adds the necessary entries to the registry. Once a class is registered, any COM client can use it as though the class were a COM class.

How COM is converted into assembly

-Specifying which namespace to place the assembly in by modifying the IDL

```
// The interface will be placed in the namespace
// MyCompany.MyComponent.IMyInterface
[
  object, dual,
  uuid(3a014c8a-3772-49bb-a8c8-b84d9bf6db72),
  custom(0F21F359-AB84-41E8-9A78-36D110E6D2F9,
                "MyCompany.MyComponent.IMyInterface")
]
interface IMyInterface : IUnknown {
  ...
};
```

COM events

-Connection Points

1. client (sink) queries source for IConnectionPointContainer
2. sink locates a particular IConnectionPoint, representing a specific event source interface
3. sink calls IConnectionPoint::Advise to register its sink interface pointer

-The following will be generated when COM interface with events is imported into .NET

-A .NET interface equivalent to the COM source interface, which has an _Event suffix. This interface contains the same members as the COM interface but is declared as a .NET interface.

-A delegate for each method on the source interface. These methods have an _EventHandler suffix.

-A class with a _SinkHelper suffix. This class implements the .NET interface so that .NET clients don't see the underlying COM connection-point mechanism.

-A class with an _EventProvider suffix. This class deals with talking to the COM component's IConnectionPointContainer interface and obtaining an IConnectionPoint interface.

Using .NET components in COM applications

Generating CCW

-GUID will be generated unless specified. To have a constant GUID, fix the AssemblyVersion to a constant in AssemblyInfo or explicitly provide a GUID attribute.
-Generated CCWs would include IUnknown, IDispatch, IProvideClassInfo, ISupportErrorInfo, IErrorInfo, and ITypeInfo. It may also include IDispatchEx, IConnectionPointContainer and IConnectionPoint, and IEnumVARIANT
-If the .NET component implements a .NET interface, then the interface is in turn implemented by CCW and can be accessed through COM
-If the .NET component directly implements without a .NET interface, then a *class interface* is generated (name: _ClassName)

-Problem:

-generated on the fly, violates the COM rule that COM interfaces should not change

=> To avoid the problem, specify ClassInterfaceType.AutoDispatch

Various Attributes that can be specified in .NET code

<Guid(f7291cfb-6b3a-40e7-a9a7-fc4c444a0a08), ProgId("MyStuff.Exported")>
<ComVisible(False)>

-Note: Applying ComVisible(false) to an assembly hides all the types within that assembly. If you do this, you can then use ComVisible to make selected types within the assembly visible. Applying ComVisible(false) to a type hides all the members of that type, and you cannot selectively make members visible using ComVisible(true). Applying ComVisible(false) to an interface means that the type doesn't support it as far as COM is concerned (that is, QueryInterface calls for the interface will fail).

<AutomationProxy(True)>

Steps to install the component for COM use

1. Strong name sign the .NET component
2. Register and generate the TypeLibrary for COM
   -Three ways:
   - `tlbexp assemblyFile [options]`
   - `regasm myfile.dll /tlb:myfile.tlb`
   - Using `TypeLibConverter` (details not shown here)
3. `gacutil -i assemblyName.dll`

Windows Form control as ActiveX control

-ActiveX control needs ActiveX container, just as Windows Forms is the container for Windows Form control
-It's easy to use Windows Form control in Internet Explorer (IE acting as an activeX container)

-Just create a regular Windows Form control
-Add the following code to HTML

```
<object id="TimeBox1"
classid="http:TimeBox.dll#TimeBox.TimeBox"height="300" width="500"
VIEWASTEXT>
</object>
```

Exposing .NET Events in COM

```
using System;
using System.Runtime.InteropServices;

namespace Bank
{
   // Interface to allow COM clients to use events.
   // Define it as a pure dispatch interface
   [
     Guid("ef81a831-fa49-4ede-b70b-08f2e9d602b2"),
     InterfaceType(ComInterfaceType.InterfaceIsIDispatch)
   ]
```

```
   public interface IAccountEvents {
      void NewRate(double val);
   }

   // Delegate for rate change event
   public delegate void RateDelegate(double val);

   // Export the Account class, exposing IAccountEvents
   // as a source interface
   [ComSourceInterfaces(typeof(IAccountEvents))]
   public class Account {
      private static double interestRate = 0;

      // Define the event
      public event RateDelegate NewRate;

      public Account() {
      }

      public double Rate {
         get {
            return interestRate;
         }
      }

      // Fire the event
      public void SetRate(double d) {
         NewRate(d);
      }
   }
}
```