COM(+) Programming with C++

Instructor Guide

Table of Contents

Course Philosophy and Design Goals

Overview

The primary objective of the COM(+) Programming with C++ course (*the course*) is to teach the fundamental concepts and techniques of development using the Microsoft Component Object Model (COM).

The course is pitched at an introductory level. Consequently, it is suitable for developers with C++ programming experience, but no prior exposure to COM.

It should be noted that the course covers COM, and does not venture into the realms of higher-level technologies, such as ActiveX controls, OLE or structured storage.

The course also endeavours to explain not just the *how* of COM, but also the *why*. This is reflected in the extended first chapter of the course where we attempt to refocus the delegates' minds towards interface-based component development.

Where possible, this philosophy is reflected in each individual chapter, with a discovery sequence that flows as follows:

- Statement of problem
- Explanation of solution / COM technology that addresses problem
- (In later chapters) Explanation of how ATL simplifies the task
- Conclusion

At the end of the course, a reasonable delegate should feel comfortable if tasked to create a moderately complex in-process COM object using ATL.

They will have been introduced to apartments and marshalling, and should therefore be able to build and deploy proxy/stub DLLs, but will probably not have the experience to design efficient COM interfaces (especially those delegates with little or no prior experience in client/server development).

The delegates will also have some feeling about what COM+ is for, although we cannot expect them to have fully grasped all of this in the short amount of time that is available to them.

About the Instructor Guide

Teaching any course is a personal experience, and this Instructor Guide is simply a reflection of one instructor's approach. Feel free to use (or ignore) as much or as little of this material as you want.

The timings that are provided below are *suggestions*. You may well dwell on certain chapters more than others, and balance the timings accordingly. There is, however, one thing to be aware of – there is virtually no slack in the timings at all. Avoid getting behind!

This guide highlights anecdotes and examples that you can use during the sessions, as well as identifying areas that typically cause the delegates the most problems or raise the most questions.

Demonstrations are described, but not in key/mouse clicking detail. Instead, the basic sequence and goals of each demo are described.

During the descriptions of the chapters, each slide is identified by name and is represented in italics e.g. *Introduction to COM*.

When referring to exercises (AKA labs or delegate practical sessions), I have referred to them by number. However, the exercises on the disk have the same name as the chapter to which they apply.

Finally, the course is descended (using single inheritance, of course), from the old COM Programming with C++ course (code WNCOMPG-1). The next chapter outlines the major changes between the two courses.

Changes from WNCOMPG-1

Major and structural changes

The initial chapter on *Component Development* has been removed, with some of the slides being absorbed into *From C++ to COM*. The reason for this was to trim the start up of the course, to improve the initial flow and to remove some duplication of explanation.

The entire chapter on *ADO Programming* has been removed from the course. This chapter was not core to COM and had a less than satisfactory exercise. This chapter has been demoted to supporting material at the back of the student guide.

A new chapter, *Introduction to COM+*, has been added to the final day of the course. This provides an overview of the facilities of the COM+ environment.

Some of the supporting material chapters from the back of the student guide have been removed. Many of them had no supporting notes or were very dated.

Connection Points has been moved to an earlier slot in the teaching sequence. This works better when brought forward before the chapters on more advanced subjects.

The exercises for *Distributed COM* and *Threads and Apartments* have been simplified. The first of these two exercises involved unnecessary complications such as surrogates, while the second was too obfuscated for most delegates.

All exercises and solutions have been thoroughly checked, corrected and debugged. All workspaces have been modified so that they deliver IntelliSense within Visual Studio 6.

The exercises have all been renamed, as have all of the solutions. This should also have the added benefit of assisting delegates when they have both the exercise and the solution open.

The exercise for *COM Fundamentals (Server)* has been rewritten to make it easier to work with. In particular, the lock and object counting has been made simpler to use.

New notes have been written for the *Automation* chapter. The notes and slides in *Threads and Apartments* have been upgraded to discuss the TNA.

Finally, the *Way Ahead* mentions attributed C++ and ATL 7.

Minor changes

Code examples in the slides have been checked and corrected.

All notes have been read and modified for consistency of style and accuracy of content.

COM Fundamentals (Client) now contains information on writing a Visual Basic client (after all, doing so is part of the following exercise).

The *Introduction to IDL* chapter now contains a little more information and includes a slide on the different pointer attributes (again, we expect them to use these attributes in the exercise). Additionally, the incorrect description of `pointer_default()` has now been corrected.

The exercise on writing client code for IDispatch has been made optional.

There has been some minor re-sequencing of slides in the chapter on *Connection Points*, to provide a better flow and to remove duplication.

Course set-up issues

The course now runs on Windows 2000, not Windows NT 4. This is as a consequence of developing with COM+.

You must ensure that a Windows Platform SDK is installed that has the correct support for COM+. The course has been tested with the February 2001 SDK, so any subsequent SDK should be acceptable.

Suggested Course Timings

Day 1	Duration (mins)	Suggested timing
Course Introduction	30	09.30 – 10:00
From C++ to COM	150	10:00 – 12:30
Exercise 01	75	13:30 – 14:45
COM Fundamentals (Client)	60	14:45 – 16:00
Exercise 02	60	16.00 – 17.00

Day 2	Duration (mins)	Suggested timing
*Review session**	15	09:00 – 09:15
COM Fundamentals (Server)	90	09.15 – 10.45
Exercise 03	105	10.45 – 12:30
Introduction to IDL	75	13:30 – 15:45
Exercise 04	75	15.45 – 17:00

Day 3	Duration (mins)	Suggested timing
*Review session**	15	09:00 – 09:15
Building a COM object using ATL	105	09.15 – 11.00
Exercise 05	30	11.00 – 11.30
Strings, Variants and SafeArrays	90	11.30 – 14.00
Exercise 06	75	14.00 – 15.15
Automation	60	15.15 – 16.15
Exercise 07a and 07b (07b is optional)	45	16.15 – 17.00

Day 4	Duration (mins)	Suggested timing
Language Integration	60	09.00 – 10.00
Exercise 08	30	10.00 – 10.30
Events and Callbacks	60	10.30 – 11.30
Exercise 09	30	11:30 – 12:00
COM EXE Servers	60	12.00 – 14.30
Exercise 10	60	14.30 – 15.30
Distributed COM	60	15:30 – 16:30
Exercise 11	30	16:30 – 17:00

Day 5	Duration (mins)	Suggested timing
Threads and Apartments	75	09.00 – 10.15
Exercise 12	45	10.15 – 11.00
Introduction to COM+	120	11.00 – 14.00
Exercise 13	60	14.00 – 15.00
The Way Ahead	30	15:00 – 15.30
Course closes		15.30

Note that all timings **include** allowances for morning and afternoon breaks of fifteen minutes and one hour for lunch.

Chapter Notes

Chapter 0: Course Introduction

Objectives

- Introduce the course
- Get the delegates to introduce themselves to you and each other
- Obtain an idea of the delegates' learning objectives for the course

Teaching notes

Build the delegates' confidence in you, the instructor, and in their ability to be able to cope with the course.

Manage their expectations on the level of COM+ that will be covered, as the majority of the course is spent preparing the fundamentals of COM.

Explain the importance of starting from the "why" of COM, before progressing to the "how".

This is a fairly standard QA introductory chapter.

Chapter 1: From C++ to COM

Objectives

- Introduce the concepts of component based development
- Transition the delegates from thinking about classes to working with interfaces and components
- Solidify and evaluate their C++ experience, in particular relating to virtual functions
- Explain the motivations behind COM

Teaching Notes

Overall, this chapter is designed to teach them COM programming without touching COM at all. In effect, it takes the delegates through the design of COM step-by-step.

The chapter is long (some 27 slides) and will take considerable time, but this is time well spent, as the delegates will grasp COM much more quickly if they understand this chapter.

It is often advisable to begin with a review of C++ on the whiteboard whilst still displaying the title slide. This sharpens the delegates up after the introductions, and gets them ready to work.

> The exercise that I find works well is to start with a simple class problem (such as a Car), add a method (**not virtual**) and then write a small amount of client code to that instantiates the object. You can throw in some deliberate bugs (missing ;'s, make the method private) and so on to wake the delegates up.
>
> Then derive a new class (say a Ferrari from the Car), and add a new implementation for the method. Rewrite the client so that you are using a base class pointer, but are instantiating a derived class object
>
> e.g. Car *pCar = new Ferrari();
>
> Get them to tell you which method will be called. Then get them to fix the problem (make the method virtual), and explain how it works. Finally, go the final mile and make the base class abstract.

You can then lead onto the slides.

Completing this exercise so early in the course often settles the delegates nerves about their C++ knowledge, and makes them think about inheritance and overriding functions.

Component Based Development and *Component Deployment* provide the opportunity to explain the motivations behind CBD.

So how about C++? offers you the opportunity to point out the problems that exist with the language.

In particular, you can emphasise the weaknesses involved with shipping a C++ header file (recipient can change the accessibility attributes) and the problems of close coupling (what happens if a new private member is added – a cascading rebuild).

COM to the Rescue! is the time to emphasise the discipline side of COM programming.

And COM+? should be kept short, as it will be dealt with in the final chapter of the course. You can, if you choose, point out the COM+ is only available in W2K and beyond, and that it is the merging of MTS and the COM runtime.

Migrating from C++ to COM is an introductory slide that displays the traditional Account class. This is the start point for moving the delegates from being pure C++ developers to being COM developers.

Interface Based Programming is a bit of fun to get the delegates thinking in interfaces. Stress the immutability (what would happen if you pressed Play and your favourite video was overwritten!) and the logical grouping of functions.

This leads directly into *Decoupling the Interface*. You can whiteboard the handle class mechanism if you want (I tend not to, but it is interesting for some classes to compare and contrast).

Using and Abstract Base Class should be a refresher, especially after the first whiteboard exercise. This will lead straight into the *"vtable" Mechanism*. One option you have here is to do the exercise described above here.

The next four slides deal with the problem of construction (language neutrality) and destruction of objects. Get the delegates to tell you how to fix the problems.

Extending an Interface and *Extending a Component* set the scene for the need for QueryInterface(). Explain the problems of changing methods, adding methods (old server being called from a new client) and finding out what an object can do.

The next four slides then deal with the problem and answer to using RTTI and dynamic_cast<>. You might want to point out the weaknesses of the DynamicCast() function as provided (void * not typesafe and plain textual interface name). At the end of this block, you can naturally drop into lifetime management problems (on which pointer should you call Delete()), so we lead into reference counting for three slides.

Finally, we see *Revised Client Code*, with all the problems "solved", and then a back patting slide on how we have designed COM in one chapter!

Lead into **Exercise 1** from the *Summary* slide.

Chapter 2: COM Fundamentals (Client)

Objectives

- Introduce client side COM programming
- Familiarise the delegates with the major COM client side APIs so that they would be able to write client code
- Introduce IUnknown, HRESULTs and GUIDs
- Familiarise delegates with simple VB client programming so that they would be able to call a COM object from VB using early binding

Teaching notes

COM Basics introduces the notation of a COM object. Emphasise the fact that you cannot determine the implementation detail from the diagram, and that every COM object must support a standard interface called IUnknown.

Interfaces lets you pick out the complete encapsulation model of COM, and again emphasise the fact that an interface is defined by its vtable (order and nature of methods) and that all interfaces have a common root interface called IUnknown, which provides a clear hook into the next slide.

The IUnknown Interface allows you to introduce many interesting concepts, such as GUIDs and HRESULTs (and even remoting if you're prepared to explain why we don't really on the values of AddRef() and Release()).

In *HRESULTs*, you should introduce the macros SUCCEEDED() an FAILED(), as well as explain that the return code should be greater than 0x200.

Interface IDs is an opportunity to demo GUIDGEN.EXE and UUIDGEN.EXE. Mention that without a MAC address, the GUID that is generated is only valid for use on that single machine.

Using QueryInterface() and *Reference Counting* just require careful explanation.

Class Factories is a difficult slide for many delegates. They are not really sure why you want them (extensibility, abstraction, single mechanism for client coders to remember, etc.) . Don't go into overwhelming detail, as they are covered fully in the next chapter.

The next three slides cover the creating of a COM object using IClassFactory and CoGetClassObject(), and the lead out of the *Using CoGetClassObject* slide is the sheer volume of code that is needed to effectively replace a simple new statement.

The next two slides on CoCreateInstance() show a simpler mechanism, but make sure that the delegates are able to discern when to use which approach.

Lead them into the next two slides on other clients, and **demo** the use of VB6 to instantiate and use a simple COM object.

Typically, create a simple VB application with a button on it.

Add a reference to the type library, show them the object browser and then write the button_Click event handler with four lines of code

```
Dim obj as libname.coclassname

Set obj = new libname.coclassname

obj.Foo

set obj = nothing
```

You can then add in four more lines of C++ code so that they can see the similarities, e.g.

```
Dim obj as libname.coclassname
' IXXX obj = 0;

Set obj = new libname.coclassname
' CoCreateInstance( ... );

obj.Foo
' obj->Foo();

set obj = nothing
' obj->Release();
```

After the summary, let them loose on **Exercise 2**.

Chapter 3: COM Fundamentals (Server)

Objectives

- Enable a delegate to be able to understand and write a simple COM server in C++ (without self-registration support)

Teaching notes

After the standard overview, and a refresher diagram on interfaces, you into three slides on implementing IUnknown.

Implementing QueryInterface() and *QueryInterface() Rules* are critical slides. The first of these shows a simple example. Emphasise the fact that the code does not require RTTI. Also, you might want to get the delegates to add defensive code (some assert and release mode checks at the top of the function. When covering the rules, flip backwards and forwards between the two slides to get the delegates to see whether the implementation would work.

When talking through the rules, I always tend to do identity last, and I diagram the need for it on the whiteboard.

Reference Counting is easily explained, but many of the delegates will not have seen InterlockedIncrement() and InterlockedDecrement() before, so explain why they are needed.

The next five slides deal with packaging components into a single DLL, and explain the motivation behind DllGetClassObject(). Stress the fact that the decision on which interfaces a class object support should be made in QueryInterface(), and not DllGetClassObject() – the only if ... else if ... else clause in this latter function should be on the CLSID.

This is then followed by two slides on loading and unloading DLLs, with two more slides on. It is worth diagramming the counting problems on the whiteboard, explaining that each object is responsible for only its own count, but that the DLL has to maintain an overall count of all the objects and the class objects.

The last three slides are just summary slides, before leading into Exercise 3.

Chapter 4: Introduction to IDL

Objectives

- Introduce Interface Definition Language to the delegates.
- Give them enough knowledge to be able to read and create simple interface definitions.

Teaching notes

Before starting the slides, it is worth explaining the motivations of IDL. A good example on the whiteboard is to use the 'C' strcpy() function as an example of a bad declaration of a function. For example, what would stop someone calling strcpy as follows

```
char c;
strcpy( &c, &'A');
```

This would meet the requirements for the call as specified in the header, but is obviously fatally flawed. You can then explain that if we want free networking code, plus ease of use from VB and other development languages, we must have a better declaration language – IDL.

In general, each slide is quite self-explanatory.

On *A Simple Example* beware of being drawn on pointer_default(), as this just leads to a road of pain. Save this one for later.

The concept of direction is often hard for delegates to grasp, making *Function Declarations* a little difficult and retval is important given that all functions must return an HRESULT.

With *Typedefs, Constants and Enumerations* you might want to mention structs, and explain that they have C like declarative behaviour, so you must either use a typedef or place the struct keyword in front of all uses.

Pointers and IDL always causes problems, as this will likely be the first time that delegates ever think of pointers in this way. Explain the three scenarios, and tread lightly over pointer_default(). Explain that *no matter what pointer_default()* says, it only applies to return pointers (shouldn't they be HRESULTs and nested pointers.

The MIDL Compiler is a simple slide, but you might want to point out how little diagnostic information is provided by the compiler – which means compile often.

With *Specifying a Property in IDL* it is important to emphasise that the get property must have a retval parameter.

You should definitely demonstrate the OLEVIEW tool towards the end of the chapter.

Exercise 4 can be quite tough for the delegates, so it is often desirable to whistle through the IDL solution at the end.

Chapter 5: Building a COM Object using ATL

Objectives

- To introduce ATL and its standard wizards.
- To give the delegates the ability to create a simple COM object in ATL.

Teaching notes

We're staying on ATL 3.0 for this – 7.0 only warrants a small mention.

Always check with the delegates that they understand templates, otherwise the rest of the chapter will just be a mess. A simple walk through of the max() macro from C to the overloaded max() function in C++ to the real templatised max() explains their use quite nicely – and then follow up with a template vector class.

Talk thru the first couple of slides, and then do everything up to *A Closer Look* as a demonstration. Make sure that you cover the wizards, adding properties and methods, and implementing subsequent interfaces.

Take a light touch through *Class Relationships* as it can be quite hard to explain that ATL derives a class from the class that we've written, but then get them into the lab as soon as possible.

Remember, from here on in they will be using ATL to write COM objects – ooh rah!

Chapter 6: Strings, Variants and SafeArrays

Objectives

- Introduce these fundamental types to delegates.
- Explain how to use the ATL/C++ wrapper classes to simplify these somewhat nasty types.

Teaching notes

This chapter is nasty! Nobody loves these data types, and after the high and the fun of ATL, this is definitely a come down.

There are five slides on strings, four on variants and three on SafeArrays, and the emphasis in all cases is that we need these for language interaction.

Most delegates will not have come across smart pointer (or wrapper) classes, so treat them gently. You might need to explain the concepts (I've often found delegates on the course who are unsure of when a destructor fires!).

Other than that, I'm afraid that this chapter is a straight "you must do this like this because..." chapter. I just walk them thru the slides, explaining each sequence as I go.

I also drop in (when dealing with strings) the conversion macros from ATL, and the benefits of using the TCHAR and t***() functions for portability.

Exercise 6 is simply no fun whatsoever, so take the time to explain the second of the two parts (not many delegates can cope with the concept of a variant holding an array!) and walk them thru the nasty lack of wizard support for part three.

Chapter 7: Automation

Objectives

- Explain the Automation architecture and how IDispatch works.
- Introduce the concept of dual interfaces.

Teaching notes

I find that the best way to explain the motivations behind Automation is to

(a) ask the question "What does automation sound like?". Hopefully you'll get some sensible form of response on which you can build an answer around the process of macros and having programs automate other programs.

(b) Talk through *What is Automation?*, explain about scripting clients, how lightweight they are and how they can only support a very narrow (i.e. two) range of interfaces.

From this you can get the delegates to build the interface IDispatch up on the whiteboard by discussion. Tell them that they only have seven methods (and that three of those are the IUnknown methods), and then ask them questions along the lines of

"Given that the script developer has called a function Rotate(), do you think that this is common to all implementations of IDispatch?". They should say no, in which case this will naturally lead into GetIDsOfNames() and Invoke() (they are unlikely to get the other two).

After building the interface up, show them *How Does it Work?* and then lead into the next two slides, where the obvious emphasis needs to go onto DISPIDs and the different IDL layout.

When talking through *GetIDsOfNames()* you might want to highlight the first parameter. You can talk through the parameters, but make sure you highlight the LCID as multi-*national* language support is important.

(**If** the delegates are up to it, I sometimes ask the question about whether they think that reserved parameters are good in methods on COM interfaces. Personally, I think that they are not, as there is no mechanism in standard COM for determining whether or not you want to use them.)

Do the same with *Invoke()* and again mention LCIDs (needed for named parameters).

A Simple C++ Client and *A More Complex C++ Client* provide you the ideal opportunity to explain how nasty it is to talk from C++ to IDispatch, and to explain the fact that IDispatch is orders of magnitude slower than raw custom interfaces.

This provides the perfect link into *Dual Interfaces*, and most delegates rapidly get the benefits of supporting both types of client. I often ask the question at the end of this slide

"Dual interfaces sound like the perfect answer to all our problems?" which, as we all know, is not the case. You can lead straight into *Automation-Compatible Types* which shows one of the problems, plus you can whiteboard the problem of multiple dual interfaces (a scripting client can only talk to the methods of one of the dual interfaces without additional work).

The next four slides are straight talk thrus, with the exception of *IDispatchImpl<>*, where you might like to highlight the fact that the major and minor typelib version numbers are defaulted, so a change to these numbers will require a mod to the wizard generated code.

Exercise 7 consists of some fun with a scripting client, plus an **optional** section on calling IDispatch from C++.

Chapter 8: Language Integration

Objectives

- Primary objective is to solidify the fact that they are writing COM components to be consumed by clients outside of C++. Therefore, it raises the importance of using IDL to create type libraries.
- Introduce and explain the smart pointer classes provided by the C++ compiler.
- Explain the importance of and mechanism to provide rich error information.

Teaching notes

From *Using Type Libraries* to *A Simple VB Client* should be revision by now. Handle these slides briskly, but stress the importance of the type library as a primary means of communicating.

I never talk through the slides on using smart pointers (*Simplifying a Visual C++ Client* to *Using #import and Smart Pointers*). This is prime demo material, and you can either craft a new COM object just for this purpose, or reuse the one that you wrote in the ATL chapter.

When doing the demo, I tend to deliberately allow CoUninitialize() to be called prior to the smart pointer class' destructor firing to ensure that the delegates get the message about the fact that these "smart" pointers aren't necessarily all that smart.

Then it's time for *Error Handling*. This warrants diagramming the process of the creation of the error object and the process through which a client determines the presence of an error object.

Talk thru the slides and then its into **Exercise 8**. You might need to watch out for delegates that haven't ever used exception handling.

Chapter 9: Events and Callbacks

Objectives

- Explain the standard COM connection point mechanism.

Teaching notes

Note for instructors familiar with WNCOMPG1. There has been a reordering of the slides in this chapter, and the slide on enumeration has been removed.

This chapter should be tackled briskly. Connection points are interesting, but unless you are working with, say, ActiveX controls, this approach is a little dated.

I always diagram the basic issues involved with establishing bi-directional communication to explain why a special mechanism can be useful. Circular references are always fun.

The slide sequence is then quite specific.

The focus is on connectable objects from *What are Connectable Objects For?* thru to *Using IConnectionPointContainer*.

Then we're into connection points for the next three slides before we talk about sinks and then the connection identifier. During this section you'll often be asked about "lightweight" COM objects. Some delegates will find it hard to grasp this concept of a COM object without a CLSID, so watch out for that.

From *Code Anatomy: Connectable Object* to *ATL Support Classes* is ideal live demo territory. DON'T FORGET TO MENTION THE

BUG IN THE WIZARD THAT SOMETIMES MISSES THE D IN THE DIID_xxxx WHEN IMPLEMENTING CONNECTION POINTS.

I always demo the client in VB (it's simpler), so point out the joys of the WithEvents keyword.

Other Stuff is just a quick overview of some Do's and Don'ts, and then leap into **Exercise 9.**

Chapter 10: COM EXE Servers

Objectives

- To introduce the basic COM marshaling mechanisms.
- To enable the delegate to be able to author a basic COM EXE Server.

Teaching notes

It should be noted that the modern approach to COM development is to author DLL servers, and not EXE servers. This is as a result of the requirements of COM+ and also the added flexibility that DLLs offer. We also use the UK spelling for marshalling and marshalled throughout.

Pros and Cons of an EXE Server just gets the delegates thinking. I always start this chapter with a somewhat esoteric analogy, as follows:

> "You invite Hannibal Lecter round to tea. Everything's going fine, until you happen to ask the innocent question "Fancy a snack?" At this point Hannibal starts chasing you around the house with a large carving knife, hacking and slaying at everything in sight. Hmmm – not too desirable, really. That's what it's like when you invite a COM object in process.
>
> Wouldn't it be nice to be able to invite Hannibal round to tea, but put him in your *neighbour's* house? Then, when you ask him the same question, he hacks the other house down. We now have some fault tolerance.
>
> The problem with this is that if we are looking at an impressionist painting on the wall, we now have to scan it and fax it to the other house for Hannibal to be able to see it. If he modifies it, he has to send you back his copy, and you have to replace the one on your wall. This is obviously slower."

From *Crossing the Process Boundary* to *Registering a Proxy/Stub DLL* is all about marshaling. I always tend to diagram the concepts on the whiteboards (its often advantageous if you have two whiteboards to represent the different processes).

The next five slides are almost just a straightforward recap of why IDL is so important. Stress how important [in] and [out] are, and

explain in *Interface Pointers* the problems that occur with passing interface pointers about.

I always diagram this last point, because far too many C++ developers are used to doing dodgy casting up and down the inheritance chain, and they need to be broken of that habit.

The rest of the chapter on writing EXE servers is basically demo time again.

> Again, I always write a simple COM object with a **custom** interface, write my usual four line, single button VB client and then show them the error that occurs when the proxy/stub DLL has been built and registered.
>
> Then I build the proxy/stub DLL, register it and then voila, working cross process COM.

Then it's in to **Exercise 10**.

Chapter 11: Distributed COM

Objectives

- Provide a basic overview of DCOM to the delegates.
- Familiarise the delegates with DCOMCNFG.EXE.
- Introduce the extended CoCreateInstanceEx() API.

Teaching notes

This chapter has to be kept quite short, for

DCOM Security Overview has a build on it that lets you walk through the issues that surround creating a COM object on a remote machine.

Then it's down to a straight demo through DCOMCNFG.EXE all the way through to *Moving Server to a Remote Machine*. Then it’s a straight chat through *Remoting Programmatically, MULTI_QI* and *Using CoCreateInstanceEx()*.

There is a build on the *Remote Activation Steps*, and then into **Exercise 11**.

Note that this exercise has been somewhat simplified from the exercise in WNCOMPG-1, as there was too much that could go wrong.

Chapter 12: Threads and Apartments

Objectives

- To explain how COM objects with different threading requirements can work together.
- To introduce apartments and explain their role as a synchronization mechanism.

Teaching notes

Note that this subject matter is normally fairly tough for delegates. Often, the delegates don't even have a basic understanding of threads, which makes it even more demanding.

I always start this chapter with a simple walk though along the lines "What is a thread?". I then whiteboard thru a bit of code like this

Imagine the function

```
int g_x = 0;

void foo()
{
    g_x++:
}
```

I then use two different coloured pens to represent the state of the registers and the g_x variable given two threads that go through this function, one of which is interrupted within the g_x++ line.

Again, watch out for some developers who will think that g_x++ is a single machine instruction, because it is a single line of C++ code!!

At the end of this, g_x will have the value 1. I then ask the delegates if they agree that the code has been executed twice (they normally can cope with that), before changing the function foo() to AddRef(). I then ask them what they think of that!

This wakes them up to threading issues, making the rest of the chapter a tad easier.

Launch into *General Threading Issues* and then on into Apartments.

It can be hard to explain the concept of an apartment, especially when you hear the phrases "enter an apartment" or "lives in an apartment", so try to get across the fact that an apartment is NOT a

sub-division of a process, but is more a thread (and proxy) associated item.

When explaining the different types of apartment, I tend to use the analogy of the training room as the apartment "boundary".

> The first example has the delegates being COM objects and me being the single thread. I make a point of walking to the door to collect a message for a delegate, and then physically walk up to the delegate. From there, it becomes easy to see that whilst I'm busy talking to one delegate, all the others can't receive any messages, and that only one call will be dispatched at a time.
>
> I then reverse the roles, with me as the COM object and the delegates as the threads. It then becomes fairly obvious that I would have to be able to cope with multiple threads talking to me at the same time.

There are then four slides (with builds) that explain the needs for apartments, followed by a summary truth table (*Apartment Synchronization*).

Entering an Apartment needs one extra little fact explained – you need to #define _WIN32_DCOM or _WIN32_WINVER >= 0x400 for CoInitializeEx() to be available.

In-proc Servers lets you explain the ThreadingModel registry key, and then you're into CoMarshalInterThreadInterfaceInStream()/CoGetInterfaceAndReleaseStream(). I tend to whiteboard this with objects and threads in different apartments, and explain why things are needed.

Finally, you have a nice run of walkthroughs that really help to cement the knowledge.

In the summary, I always reinforce the fact that apartments are just a COM provided synchronization mechanism that try to help ensure that concurrency requirements are met.

Leap into **Exercise 12** which is now much simpler than before.

Chapter 13: Introduction to COM+

Objectives

- Introduce the services offered by COM+.
- Explain the Component Services MMC snap-in.

Teaching notes

This is an introductory chapter only. Detailed technical issues are NOT covered in this section – refer any delegates to MS course 2099 if they want more training on items such as the SPM or fiddling with context in great detail.

This chapter will be fairly heavy going for many delegates. Bear in mind that four days ago we were introducing basic concepts such as components and interfaces. A light (demo driven) touch is probably required here.

COM in the Enterprise just introduces the basic requirements for why we need some form of clever management environment. Stress the importance on scalability and reliability.

Services Provided by COM+ just lists the main technologies to play with, and then we lead straight into the next slide that asks the question *How does COM+ provide the services?* This is just an introduction so that we can explain interception and set up the concept of a configured component with the delegates.

COM+ Attributes introduces the concept of setting bits in a store somewhere, and then there are two slides on configuring applications and the component. It makes sense to demo this with a simple COM object that you have already built during the week in one of the other demos.

There follows four core slides on interception, contexts and apartments. The key learning point here is that there is a finer unit than the apartment, the context, and that objects live in contexts that are configured to suit the objects' requirements.

The next three slides deal with synchronisation. We start off looking at the TNA, then lead into the need for the synchronisation attribute, with a final slide on how to set the synchronisation requirements.

Then it's in to transactions. Delegates are likely to have a mixed background here, so you might need to explain why we need them, ACID properties, and so forth.

Explain how easy they are to set up with COM+, and then talk thru' the voting mechanism.

The next four slides cover basic activation issues, such as object pooling, constructor strings and JITA. You can explain the main benefits of each of these, and a little on the pitfalls, before demoing the Activation tab. You *might* want to talk about the component supporting statistics, and show them a pooled object in the snap-in.

There follows three slides on security, but keep this fairly light. The main thing to point out is that certain features are only available if you enable them on the application, and if you use a named account for the application.

There follows a quick slide on queued components, a quick slide on loosely coupled events and then a miniscule slide on deployment.

Then it’s the summary before hitting **Exercise 13**.

Chapter 14: The Way Ahead

Objectives

- Standard course close down chapter.
- Get delegate review forms and hand out certificates.
- Ensure delegates know where to go for further information.

Teaching notes

Just run through the slides. Try not to demoralise them by talking about .NET too much, and then wrap up the course.

There is no exercise for this chapter. If the delegates want to take home their work, get them to run the cleanall.bat batch file to remove all the extraneous files.

They can then run winzip and it should fit on a floppy.