The Basics of COM

Understanding how COM works can be intimidating at first. One reason for this intimidation is the fact that COM uses its own vocabulary. A second reason is that COM contains a number of new concepts. One of the easiest ways to master the vocabulary and concepts is to compare COM objects to normal C++ objects to identify the similarities and differences. You can also map unfamiliar concepts from COM into the standard C++ model that you already understand. This will give you a comfortable starting point, from which we'll look at COM's fundamental concepts. Once we have done this, the examples presented in the following sections will be extremely easy to understand.

Classes and Objects

Imagine that you have created a simple class in C++ called xxx. It has several member functions, named MethodA, MethodB and MethodC. Each member function accepts parameters and returns a result. The class declaration is shown here:

```
class xxx {
public:
int MethodA(int a);
int MethodB(float b);
float MethodC(float c);
};
```

The class declaration itself describes the class. When you need to use the class, you must create an instance of the object. Instantiations are the actual objects; classes are just the definitions.

Each object is created either as a variable (local or global) or it is created dynamically using the new statement. The new statement dynamically creates the variable on the heap and returns a pointer to it. When you call member functions, you do so by dereferencing the pointer. For example:

```
xxx *px; // pointer to xxx class
px = new xxx; // create object on heap
px->MethodA(1); // call method
delete px; // free object
```

It is important for you to understand and recognize that COM follows this same objected oriented model. COM has classes, member functions and instantiations just like C++ objects do. Although you never call new on a COM object, you must still create it in memory. You access COM objects with pointers, and you must de-allocate them when you are finished.

When we write COM code, we won't be using new and delete. Although we're going to use C++ as our language, we'll have a whole new syntax. COM is implemented by calls to the COM API, which provides functions that create and destroy COM objects. Here's an example COM program written in pseudo- COM code.

```
ixx *pi // pointer to COM interface
CoCreateInstance(,,,,&pi) // create interface
pi->MethodA(); // call method
pi->Release(); // free interface
```

In this example, we'll call class ixx an "interface". The variable pi is a pointer to the interface. The method CoCreateInstance creates an instance of type ixx. This interface pointer is used to make method calls. Release deletes the interface. I've purposely omitted the parameters to CoCreateInstance. I did this so as not to

obscure the basic simplicity of the program. CoCreateInstance takes a number of arguments, all of which need some more detailed coverage. None of that matters at this moment, however. The point to notice is that the basic steps in calling a COM object are identical to the steps taken in C++. The syntax is simply a little different. Now let's take a step back and look at some of the bigger differences between COM and C++.

How COM Is Different

COM is not C++, and for good reason. COM objects are somewhat more complicated then their C++ brethren. Most of this complication is necessary because of network considerations.

There are four basic factors dictating the design of COM:

- C++ objects always run in the same process space. COM objects can run across processes or across computers.
- COM methods can be called across a network.
- C++ method names must be unique in a given process space. COM object names must be unique throughout the world.
- COM servers may be written in a variety of different languages and on entirely different operating systems, while C++ objects are always written in C++.

Let's look at what these differences between COM and C++ mean to you as a programmer.

COM can Run Across Processes

In COM, you as the programmer are allowed to create objects in other processes, or on any machine on the network. That does not mean that you will always do it (in many cases you won't).

However, the possibility means that you can't create a COM object using the normal C++ new statement, and calling its methods with local procedure calls won't suffice. To create a COM object, some executing entity (an EXE or a Service) will have to perform remote memory allocation and object creation. This is a very complex task. By remote, we mean in another process or on another machine. This problem is solved by creating a concept called a *COM server*. This server will have to maintain tight communication with the client.

COM Methods Can Be Called Across a Network

If you have access to a machine on the network, and if a COM server for the object you want to use has been installed on that machine, then you can create the COM object on that computer.

Of course, you must the proper privileges, and everything has to be set-up correctly on both the server and client computer. But if everything is configured properly and a network connection exists, activating a COM server on one machine from another machine is easy. Since your COM object will not necessarily be on the local machine, you need a good way to "point to" it, even though its memory is somewhere else.

Technically, there is no way to do this. In practice, it can be simulated by introducing a whole new level of objects. One of the ways COM does this is with a concept called a proxy/stub. We'll discuss proxy/stubs in some detail later. Another important issue is passing data between the COM client and its COM server. When data is passed between processes, threads, or over a network, it is called "marshaling". Again, the proxy/stub takes care of the marshaling for you. COM can also marshal data for certain types of interface using Type Libraries and the Automation marshaller. The Automation marshaller does not need to be specifically built for each COM server – it is a general tool.

COM Vocabulary

One of the problems we're going to have is keeping track of two sets of terminology. You're probably already familiar with C++.

Concept	Conventional (C++/OOP)	COM
Client	A program that requests services from a server.	A program that calls COM methods on a COM object running on a COM server.
Server	A program that "serves" other programs.	A program that makes COM objects available to a COM client.
Interface	None.	A pointer to a group of functions that are called through COM.
Class	A data type. Defines a group of methods and data that are used together.	The definition of an object that implements one or more COM interfaces. Also, "coclass".
Object	An instance of a class.	The instance of a coclass.
Marshaling	None.	Moving data (parameters) between client and server.