COM Corner: COM+ Events

by Steve Teixeira

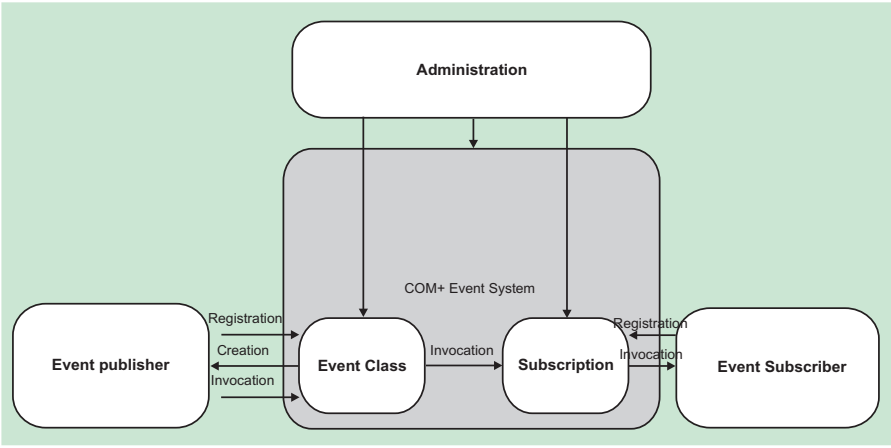
The typical picture we imagine when we think about the relationship between COM client and server objects is fairly linear: clients invoke methods on servers and servers do stuff in response to the client call and optionally provide some data back to the client in the form of a return value and out parameters. This relationship is probably an accurate representation of more than 90% of COM client/server interactions, but you don't have to be a COM guru to realize that this model is limited, particularly with regard to clients having the ability to be quickly updated when some server data changes.

The simplest way to obtain such a notification would be for clients to poll servers on a periodic basis in order to check whether the information which they're interested in has changed. However, the disadvantages of polling are pretty self-evident: clients waste a lot of cycles sending polls, servers likewise waste a lot of clock cycles responding to polls, extraneous network traffic may be generated, and the overall scalability of the system is diminished by the sum of all this increased load on client, server and wire.

More desirable, but still low-tech, is a system whereby clients can pass servers one or more pre-defined interfaces to call back on when the information in question changes. However, this system essentially has to be re-invented for every different interface you wish to use, and it is incumbent upon the server to write specialised code to track multiple client connections.

Traditional COM provides a more efficient and structured solution to this problem, which I've written about several times in this column, called *events*. This solution involves the use of the *connection points*, which provide servers with the ability to track clients that wish to be notified of information changes, as well as the means for servers to call client methods to make the notifications. Connection points are an example of what is known as a Tightly Coupled Event (TCE) system. In a TCE system, clients and servers are mutually aware of the other's identity. Additionally, TCE systems require that clients and servers be running simultaneously, and they provide no means for filtering of events. The connection point system also has the inherent disadvantages of being rather complex to implement and use, clients are also forced to implement entire event interfaces, even if they are only interested in a single method of the interface.

➤ *Figure 1: COM+ event system.*



COM+, which will be released in mid-February as a part of Windows 2000, contains a new event system that solves some of these problems and adds some nice features. The COM+ event model is known as a Loosely Coupled Event (LCE) system. It is referred to as such because there is no hard connection between servers (known as event *publishers*) and clients (known as event *subscribers*). Instead, publishers register with the COM+ catalog the events they wish to publish, and subscribers separately register with the COM+ catalog the events in which they are interested. When a publisher fires an event, the COM+ runtime reviews its database to determine which clients should receive an event notification and sends the notification to those clients. What's more, clients don't even have to be running when the event is fired; COM will activate clients upon invocation of the event. Additionally, the event registration model supports method level granularity. This means that subscribers aren't forced to implement methods for events in which they have no interest. Figure 1 provides an illustration of the COM+ event system.

As Figure 1 shows, the process begins when the publisher registers a new event class. This can be done using the Component Services administration tool or using the `ICOMAdminCatalog.InstallEventClass()` method. Once registered, the object that implements the event class will reside in the COM+ runtime. The publisher, or another object, can then call the `CoCreateInstance` COM API call to create an instance of this object and call methods on this object to fire events.

On the subscriber side, the subscriber can register for an event class permanently, using the Component Services administration tool, or in a transient manner using the COM admin catalog API. Permanent subscription means

```
unit PubMain;
interface
uses
  ComObj, ActiveX, Publisher_TLB, StdVcl;
type
  TEventObj = class(TAutoObject, IEventObj)
  protected
    function MyEvent(const EventParam: WideString): HResult; safecall;
  end;
implementation
uses ComServ;
function TEventObj.MyEvent(const EventParam: WideString): HResult;
begin
end;
initialization
  TAutoObjectFactory.Create(ComServer, TEventObj, Class_EventObj,
    ciMultiInstance, tmApartment);
end.
```

➤ *Listing 1*

that the subscribing component doesn't need to be active when the event fires: the COM+ runtime will automatically create the component before invoking the event. Transient subscriptions are intended for components which are already active and wish to receive event notifications only temporarily. When the publisher fires an event, COM+ will iterate over all the registered subscribers, invoking the event on each. Note that it is not possible to determine the order in which COM+ will iterate over the clients when invoking an event. However, it is possible to gain some control over the firing of events using *event filters*, which I will describe in more detail later.

Speaking practically, creating a COM+ Event can be boiled down into a 5-step process:

1. Creation of an event class server.
2. Registration and configuration of the event class server.
3. Creation of a subscriber server.
4. Registration and configuration of the subscriber server.
5. Publishing of events.

I'll take these steps one at a time in order to demonstrate a Delphi 5 implementation of COM+ events.

Event Class Server Creation

The first step to creating an event class server is to create an in-process COM server to which you will add a COM object. The important distinction to bear in mind between creating an event class server and creating a regular COM server is that *an event class server carries with it no implementation*, it only serves as a vehicle for definition of the event class.

I create an event class server in Delphi by using the ActiveX Library wizard to create a new COM server DLL and the Automation Object wizard to generate the event class and interface. I'll call this object `EventObj`. The wizards leave me in the Type Library Editor to complete the definition of the server, where I add a method to the `IEventObj` interface, called `MyEvent`, that will serve as the event method. The implementation file produced for this type library is shown in Listing 1.

That's all there is to creating the event class server. Note that it's not necessary to register this server: registration is handled specially, as I will discuss next.

Register And Configure

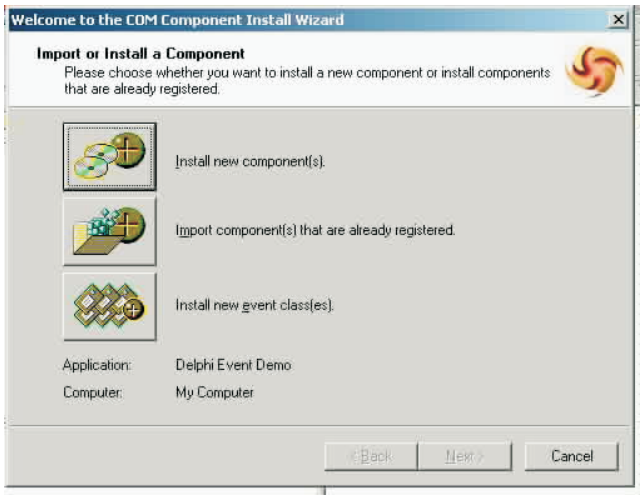
In this phase you take your first step towards becoming good friends with the Component Services administration tool. You'll use this tool often as you develop COM+ applications. You'll find it in the `Administrative Tools` group of the `Programs` section of the `Start` menu (I'm using Windows 2000 Server, Release Candidate 3). The first thing you'll need to do in the Component Services administration tool is create a new COM+ application. You can do this by selecting `New | Application` from the local menu of the `COM+ Applications` node in the tree view on the left. This will invoke the COM+ Application Install Wizard, as shown in Figure 2. In this wizard, I choose to create a new application from scratch and call it *Delphi Event Demo*.

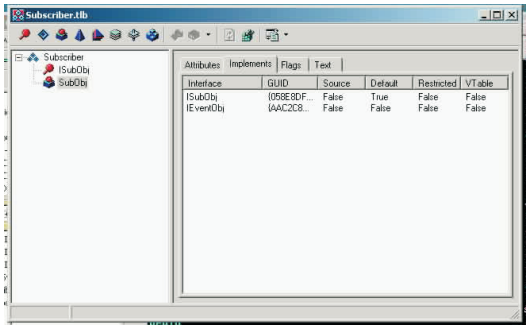
Once the COM+ application has been installed, I can install the event class server into the application, by selecting `New | Component` from the local menu of the `Components` node under the new application in the tree. This invokes the COM Component Install Wizard, part of which is shown in Figure 3.

In this wizard, I choose to install a new event class, and I select the

➤ *Below: Figure 2*

➤ *Right: Figure 3*





➤ *Figure 4*

file name of the event class server that I just created. With that done, it's time to move on to creation of the subscriber server.

Subscriber Server Creation

A subscriber server is essentially a standard Delphi Automation server. The only catch is that you need to implement the event interface that you defined when creating the event class server. I accomplish this by using the type library from the event class server in the subscriber server and adding the `IEventObj` interface to the implements list of the coclass.

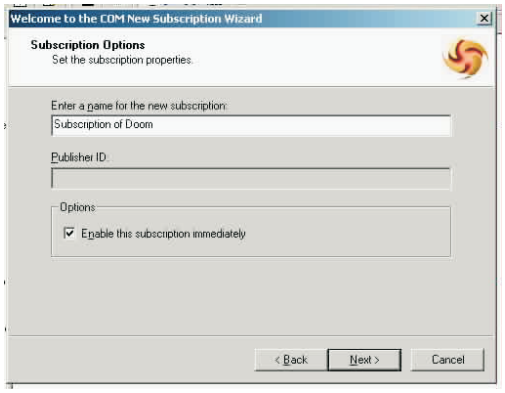
Figure 4 shows the `SubObj` coclass, containing both `ISubObj` and `IEventObj`, and the implementation file for this type library is shown in Listing 2.

You can see that the implementation of the event is quite earth-shattering: a message box is displayed showing a real, live, text string! Again, there is no need to register this server as you would a standard COM server. That housekeeping is handled in the next step.

Subscriber Server Registration And Configuration

To register the subscriber server, I reopen the Component Services administration tool, and choose `New | Component` from the local menu just as I did in for the event class server. The difference is that this time I choose to install a new component in the COM Component Install Wizard and select the subscriber DLL.

Once the subscriber server is installed, I can create a new subscription for the subscriber server I do this by selecting `New | Subscription` from the `Subscriptions` node under my new subscriber server. This brings up the New Subscription wizard, which allows me to define the correlation between the publisher and subscriber interfaces or methods. In this case, I select `IEventObj` for the subscriber method(s) and for



➤ *Figure 5*

the event class I choose `Publisher.EventObj`. I enter *Subscription of Doom* as the name of this subscription and choose to enable the server immediately, as shown in Figure 5.

Figure 6 shows the complete COM+ application definition as shown in the Component Services administration tool.

Publishing Events

The setup is now complete, so all that is left is to publish the event by creating an instance of the `EventObj` class and calling the `IEventObj.MyEvent` method. The simplest way to do this is in a small test application, as I've shown in Listing 3.

Figure 7 shows the result of pushing the magic button. Note that the event subscriber is created automatically by COM+ and the event handler code is executed.

You might notice that COM+ takes a few moments to invoke the event the first time through. This is due to the fairly substantial amount of internal infrastructure that needs to be loaded in order to fire COM+ events. The bottom line here is that you shouldn't depend on events being fired back to subscribers in real time. They'll get there soon, but not instantly.

Beyond The Basics

While this article provides a solid grounding in the fundamentals of the COM+ event model, there are a couple of powerful features that I'd like to mention.

The first is queued events. These are the synthesis of COM+ events and queued components (known as MSMQ components in pre-COM+ days). Essentially, this functionality provides the ability to fire events to disconnected components, and those events can be played back at a later time.

The other advanced topic worthy of mention is event filters, which come in two flavours: publisher filters and parameter filters. Publisher filters provide a means for publishers to control the order and firing of an event method by an event class. Parameter filters enable publishers to intercept events based on the value of the parameters of that event.

Summary

This article was intended to give you the background of why COM+ events are such a great step forward in the world of COM event notifications, an understanding of the theory of how COM+ events operate, and the knowledge needed to build applications that take advantage of COM+ events in Delphi. Using this technique for LCEs can help you solve some of the traditional disadvantages of TCEs and create more scalable and more efficient distributed applications.

Steve Teixeira is the VP of software development at DeVries Data System and co-author of *Delphi 5 Developer's Guide*. You can reach Steve by email at steve@dvdata.com

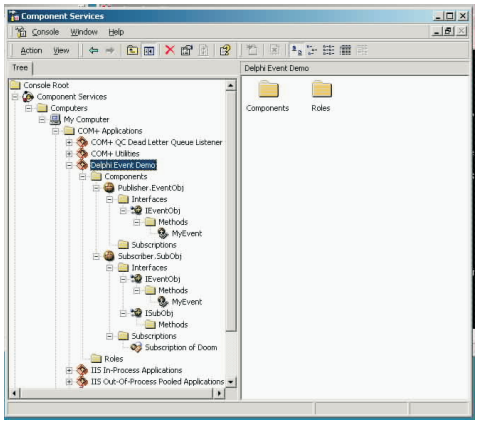
➤ *Above: Listing 2*

```
unit SubMain;
interface
uses
  ComObj, ActiveX, Subscriber_TLB, StdVcl, Publisher_TLB;
type
  TSubObj = class(TAutoObject, ISubObj, IEventObj)
  protected
    function MyEvent(const EventParam: WideString): HResult; safecall;
  end;
implementation
uses ComServ, Windows;
function TSubObj.MyEvent(const EventParam: WideString): HResult;
begin
  MessageBox(0, PChar(string(EventParam)), 'COM+ Event!', MB_OK);
  Result := S_OK;
end;
initialization
  TAutoObjectFactory.Create(ComServer, TSubObj, Class_SubObj,
    ciMultiInstance, tmApartment);
end.
```

➤ *Below: Listing 3*

```
unit TestU;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Publisher_TLB, StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    FEvent: IEventObj;
  end;
var Form1: TForm1;
implementation
uses ComObj, ActiveX;
{$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject);
begin
  OleCheck(CoCreateInstance(CLASS_EventObj, nil, CLSCTX_ALL, IEventObj, FEvent));
  FEvent.MyEvent('This is a clever string');
end;
end.
```

➤ *Figure 6*



➤ *Figure 7*