Pain-Free COM+ Events With Helper Objects

by Juval Lowy

Use the COM+ events helper objects to transform your interaction with the COM+ catalog from a nightmare to a joy.

WHAT YOU NEED

Windows 2000
Visual C++ 6.0
Microsoft Platform SDK

Loosely coupled events (LCE) is an exciting new service introduced by COM+, and it has evolved to address classic COM problems of notifying and receiving events. LCE provides additional capabilities such as transaction support, security, asynchronous publishing, and asynchronous event delivery.

The COM+ Component Services Explorer provides easy visual administration for some of the LCE features, such as setting up persistent subscribers and adding a new event class. But important LCE features such as adding or removing a transient subscription, implementing, adding, and removing a publisher filter, and transient subscriptions filtering are only available programmatically.

The developer is required to program against a cumbersome set of COM dual interfaces exposed by the COM+ Explorer called the catalog interfaces. The catalog interfaces have many limitations. For one, they are not type-safe. A BSTR is used instead of a "normal" string, and it represents GUID, IID, and CLSID. Properties values are packaged in amorphous variants. Another limitation is that the COM+ interfaces and the underlying programming model and objects hierarchy require a lot of generic code for iterating over the catalog, even for simple tasks. Finally, the resulting code is tedious and error-prone.

This article presents you with an easy-to-use helper object that wraps the COM+ catalog, saving you the agony of programming directly against the catalog. This reduces hundreds of lines of code into a mere line or two.

The wrapper COM object encapsulates the catalog objects and interfaces, and instead exposes simple custom interfaces (with type safety) that do all the hard work (see Figure 1). The interfaces are your one-stop-shop for easy transient subscriptions management and publisher filtering, providing you with the same functionality as the catalog interfaces with a fraction of the code.

I will also give you a generic implementation of a publisher filter you can use to implement your own publisher filtering. You provide the filtering logic, and the generic filter provides all the required plumbing and interaction with the COM+ catalog.

Note that this article assumes you are familiar with the basic concepts of COM+ events such as event classes, publishing events, persistent subscribers, transient subscribers, and publisher-side filtering. See the Resources box for articles that cover the basics of COM+ events. If you are unfamiliar with these terms, I recommend you read those articles first to take full advantage of this article.

Transient subscriptions are essential to COM+ events, and they are the only way an existing object (a component instance) can receive COM+ events. Like a persistent subscriber, the object must implement interfaces it wants to receive the events on (these interfaces are sometimes called "sink" interfaces). The transient subscriber can choose to subscribe to all the events a particular event class can publish, to a particular interface supported by the event class, or even to a particular method on a particular interface.

The transient subscription must be registered with the COM+ catalog and removed from it when the object wants to unsubscribe. Here's how you register a transient subscription:

Create the catalog object CLSID_COMAdminCatalog and get a pointer to ICOMAdminCatalog. Call ICOMAdminCatalog::GetCollection() to retrieve a collection called "TransientSubscription" and retrieve the ICatalogCollection interface pointer. Call ICatalogCollection::Add() to get ICatalogObject. Call ICatalogObject::put_Value() to set the desired transient subscription properties, such as the event class to subscribe to, subscribing interfaces, the subscription name, and whether the subscription should be enabled. Call ICatalogCollection::SaveChanges(). Finally, release everything.

You are required to perform a similar sequence to remove the transient subscription. The catalog wrapper encapsulates the tedious code required for registering a transient subscription by exposing the interface ITransientSubscription, which allows you to subscribe easily to all the interfaces of a specified event class or to a particular interface on that class:

```
interface ITransientSubscription : IUnknown
{
HRESULT Add([in,string]LPCWSTR pwzName,
        [in]CLSID clsidEventClass,
        [in]REFIID iidInterface,
        [in]IUnknown *pSink);
  HRESULT Remove([in,string]LPCWSTR pwzName);
  HRESULT AddFilter([in,string]LPCWSTR pwzSubName,
            [in,string]LPCWSTR pwzCriteria);
  HRESULT RemoveFilter([in,string]LPCWSTR pwzSubName);
};
```

You create the catalog wrapper using the class ID CLSID_CatalogWrapper (download Listing 1 from the *VCDJ* Web site; see the Go Online box for details). When you add a subscription, you provide the catalog wrapper with the subscription name: a string identifying the subscription. The name identifies the subscription when you want to remove it later or associate a filter with it.

The two other methods of ITransientSubscription let you easily install a subscriber-side filter for your transient subscription.

Subscriber-Side Filtering: What's it Good For?

Not all subscribers perform meaningful operations in response to every published event. For example, you might want to take action only when your favorite stock is trading, or maybe only when it is trading above a certain mark.

One possible course of action is to accept the event, examine the parameters, and then decide whether to process or discard the event. This process is inefficient, however, when the subscriber is not interested in the event because it forces a context switch to allow the subscriber to examine the event and requires redundant network round trips. You would also have to write extra examination code that can't introduce defects, bugs, and additional testing. Event examination and processing policies will likely change over time and between customers; you'll end up chasing your tail trying to satisfy all your customers.

Here's a better idea: Place the filtering logic outside the subscriber's scope. The filtering logic should be configurable administratively, potentially different for each customer site. This is exactly what a COM+ subscriber-side filter provides (see Figure 2).

For example, if you subscribe to an event that notifies you when a new user is added to your portfolio management system and the method signature is:

```
HRESULT OnNewUser([in]BSTRbstrName,
                  [in]BSTR bstrStatus);
```

you can specify filtering criteria such as:

```
bstrName = "Bill Gates"  AND  bstrStatus = "Rich"
```

The event is delivered to your object only when the user name is "Bill Gates" and his current status is "Rich."

Currently, the filtering criteria only filters string values. The filter criteria string recognizes relational

THE CATALOG WRAPPER OBJECT

Figure 1 | The catalog wrapper helper object exposes two easy-to-use interfaces (with type safety) that encapsulate the details of interacting with the COM+ catalog. Clients who use these objects will have elegant, concise, and low-maintenance code.

RESOURCES

- "A Hands-On Look at COM+ Events" by Jeff Prosise, *VCDJ* January/February 2000, covering the motivation for using COM+ events, event classes, publishers, persistent subscribers, and COM+ Explorer configuration.
- "Tap Into Transient Subscriptions" by Jeff Prosise, *VCDJ* March 2000, covering transient subscriptions—what they are and how you add or remove them—and the basics of programming the COM+ catalog.
- "Write Publisher Filters" by Jeff Prosise, *VCDJ* April 2000, covering publisher filters—how to set up and remove them.

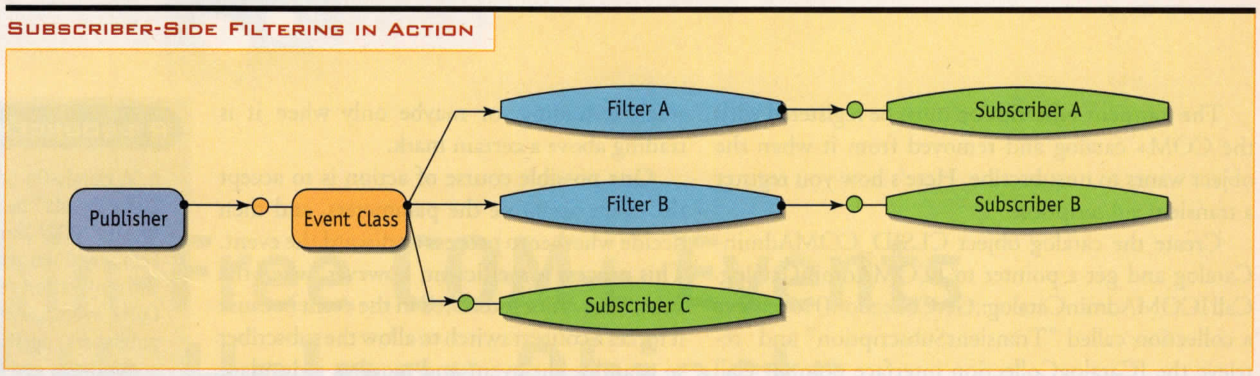

Figure 2 | Subscribers who want to be notified only when an event meets a certain criteria—not when every event is published to them—can specify filtering criteria. The filter will affect only that subscriber, and allows the separation of event filtering from event handling.

operators for checking equality (= =, !=), nested parentheses, and the logical keywords AND, OR, and NOT.

COM+ evaluates the expression and makes the call only when the criteria is true. Be aware that if you include the wrong parameters, introduce spelling errors or typos, or change the parameter names, the subscriber will never be notified.

Specifying a subscriber filter administratively is available only for persistent subscribers. To do this, display the persistent subscription properties, select the Options tab, and specify the Filter criteria. Transient subscribers must program against the catalog to set a transient subscription filter criteria, following similar steps to those used when registering a transient subscription. The catalog wrapper ITransientSubscription interface allows you to add a subscriber-side filter to a transient subscription with the AddFilter() and RemoveFilter() methods. The methods accept the subscription name and a filtering string. For example:

```
LPCWSTR pwzCriteria = L"bstrUser = \"BillGates\"";
pTransientSubscription->AddFilter(L"MySubs",pwzCriteria);
```

Manage the Publisher-Side Filter

Publisher filtering is a powerful mechanism that gives the publisher fine-grained control of events delivery. You can use a filter to choose not to publish to certain subscribers, control the order in which the subscribers get the event, and find out which subscriber did not receive the event. The publisher-side filter intercepts the publisher's call on the event class, applies filtering logic on it, and publishes accordingly (see Figure 3). If you associate a filter with an event class, all events published using this class go through the filter first.

A publisher-side filter class ID is stored in the COM+ catalog as a property of the event class it filters. When a publisher fires events on the event class, COM+ creates the publisher object behind the scenes and lets it perform the filtering. The bad news is that in order to install the filter, you must program it against the COM+ catalog.

Here's how you add a publisher filter to a particular event class:

Create the catalog object using CLSID_COMAdminCatalog and get a pointer to ICOMAdminCatalog. Then get the applications collection. Next, call ICOMAdminCatalog::GetCollection() to retrieve a collection called "Applications" and get back the ICatalogCollection interface pointer. The Application collection allows you to iterate over all your machine's applications. Each application is represented by an ICatalogObject interface.

For each application in the collection, you then get the components collection, iterate through it, and look for the event class. If not found, get the next application collection and scan its components collection. Once you find the event class, set its MultiInterfacePublisherFilterCLSID property to the filter's CLSID. Finally, save the changes on the components collection and release everything.

The good news is that the helper class implements an interface called IFilterInstaller, defined as:

```
interface IFilterInstaller : IUnknown
{
  HRESULT Install([in]CLSID clsidEventClass,
                         [in]CLSID clsidFilter);
  HRESULT Remove ([in]CLSID clsidEventClass);
};
```

IFilterInstaller makes adding a filter a breeze. You just specify the class ID of the event class and the filter, and IFilterInstaller does the rest. It's as simple as this one line:

```
pFilterInstaller->Install(CLSID_MyEventClass,
                          CLSID_MyFilter);
```

Note that you do not need to specify the application name as a parameter, just the event class and the filter CLSID. And because an event class can be associated with only one filter at a time, installing a new filter will override the existing one. Use IFilterInstaller::Remove() to remove any filter associated with a specified event class.

Implement a Publisher-Side Filter

The next step to making the COM+ events system more palatable is simplifying the implementation of the filter class itself; in particular, making a filter that controls which subscriber receives the event.

Implementing a publisher filter has two facets—the event publishing interception and the filtering logic. The interception requires intimate knowledge of the COM+ event system's mechanics. The filter class must follow these "easy," "well-documented" steps:

Implement the interface IMultiInterfacePublisherFilter that affects all interfaces on the event class you are filtering; COM+ calls this interface when the filter is created and every time an event is fired. Cache a pointer to IMultiInterfaceEventControl, which COM+ handed to you. Prepare an initial filtering criteria, and call IMultiInterfaceEventControl::GetSubscriptions() to get an IEventObjectCollection pointer (actually points to subscribers). From the collection, get an enumerator over the event objects—IEnumEventObject—and release the collection. Iterate using IEnumEventObject over the subscriber list and get one subscriber interface at a time (IEventSubscription). Extract the subscriber data from IEventSubscription (such as name, description, and IID). Apply filtering logic and decide whether you want to publish to that subscriber. If you want to fire at a particular subscriber, use a pointer to IFiringControl that COM+ handed to you and call IFiringControl::FireSubscription(), passing in an IEventSubscription pointer to that subscriber. Release the current subscriber and continue to iterate. Finally, release the enumerator.

Luckily, the intercepting plumbing is generic enough that I was able to implement all of it in an ATL COM object called CGenericFilter, which does all the messy interaction with the COM+ event system required of a publisher filter. All you have to do is provide the filtering logic.

As part of this article's files available for download, you will find the Filter project. The application /domain specific filtering logic is in a dedicated CPP file called "Domain Specific.CPP," which contains two simple helper methods you should implement to provide your own filtering logic: CGenericFilter::ShouldFire() and CGenericFilter::GetCriteria() (download Listing 2).

CGenericFilter calls GetCriteria() once per event to determine what subscribers consider for filtering. For example, you can set up initial criteria for publishing only to subscribers that have subscribed to this particular event class. You would do so by providing a criteria in the form of:

```
EventClassID == {66D88CB1-51D4-4396-B5D4-80B9FAC3077}.
```

The filtering criteria is nothing more than an advanced optimization, and the default implementation of CGenericFilter returns "All."

ShouldFire() is the interesting method. CGenericFilter calls it once per subscriber for a particular event, passing in a custom struct of type SubscriptionData as a parameter. This custom struct contains every available bit of information about the subscriber: the name, description, machine name, and so on. You examine the subscriber and return TRUE if you want to publish to this subscriber, FALSE for everything else. The default CGenericFilter implementation provides a simple example of filtering logic. It returns TRUE from ShouldFire() only for subscribers whose description field says "Paid Extra." VCDJ

About the Author

Juval Lowy is a seasoned software architect. He spends his time publishing and conducting training classes and conference talks on object-oriented design and COM/COM+. He was an early adopter of COM, and has unique experience in COM design. He is also author of an up-and-coming book on COM+ and .NET (O'Reilly). E-mail him at idesign@componentware.net.

GO ONLINE

Use these DevX Locator+ codes at www.vcdj.com to go directly to these related resources.
VC0101 Download all the code for this issue of *VCDJ.*
VC0101PC Download the code for this article separately. This article's code includes help files, a test harness, and a sample application containing the generic filter class.
VC0101PC_T Read this article online. DevX Premier Club membership is required.
Want to subscribe to the Premier Club? Go to www.devx.com.

PUBLISHER FILTERING

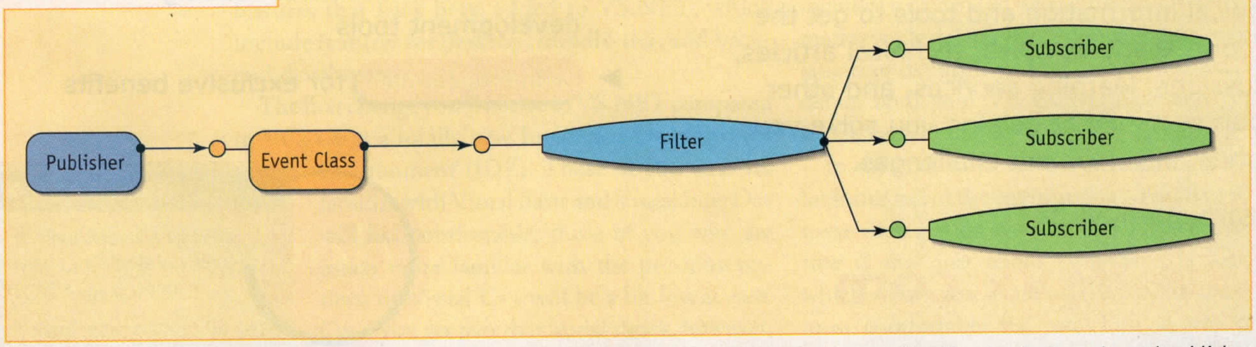

Figure 3 | A publisher filter is an object that intercepts the event firing using the event class, applies filtering logic on it, and publishes accordingly. For example, the filter can decide which subscriber gets the event. The filter affects all subscribers to that event class. This gives the publisher fine-grained control over events delivery.