

## Re: Application.Run() problem

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2006-10/msg02886.html>

---

- *From:* "Peter Duniho" <[NpOeStPeAdM@xxxxxxxxxxxxxxxxxxxx](mailto:NpOeStPeAdM@xxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Mon, 16 Oct 2006 10:55:15 -0700
- 

"Joel" <[joelagnel@xxxxxxxxxx](mailto:joelagnel@xxxxxxxxxx)> wrote in message  
<news:1160999165.328201.103510@xx>

Thank you for your response and I apologise for posting the follow-up.

Apology accepted. :)

I feel much better about the first question I asked but I still am very much confused about the paint event handler.

Me too...but I think in a different way than you are. :)

[...]

I want to know, which event holds the delegate list, is it the base class or the derived class?

It most obviously should be the derived class, but in the Control-Form case, it seems to be the base class (Control) whose paint event holds the list of the registered paint event handlers of the derived class.

It seems that you're not understanding how inheritance works. When a class derived from Control inherits an event from that class, there is no "derived event". The inheriting class simply gets to use the base class's event as if it were its own.

In only specific situations will an inherited member of a class be different from the base class's member. In each of those situations, some additional work must be done to explicitly cause the inherited member to actually supercede the base member. At the minimum, a new instance of that member must be defined in the derived class. And typically, this is done only for override (virtual) methods, which ensures that the base class uses the same new instance of the member that the derived class is using.

## Re: Application.Run() problem

This extra work doesn't exist for the events you're asking about. The derived classes don't define new events that supercede the base class's instances. So when a derived class inherits those events, they are the exact same events known to the base class. It works in exactly the same way that a method in the base class is inherited in the derived class without a new instance of the method being created. When the derived class calls the inherited method, the code that's run is the code that was originally defined for the base class, not some new code that was created as a consequence of the inheritance. Likewise, the inherited event is not some new event that was created through inheritance, but rather just the original event the base class had.

So, if you are using the paint event to register a paint handler, you use the same paint event defined in the base Control class, even though you may be dealing with an inherited class. There's no additional event in the derived class unless one has been explicitly defined, and one hasn't been explicitly defined because it's not needed.

Pete

.