

Re: Delegates and Interfaces

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework/2008-01/msg00061.html>

- *From:* Ryan <rchill81@xxxxxxxxx>
 - *Date:* Mon, 7 Jan 2008 11:59:08 -0800 (PST)
-

On Jan 7, 12:59 pm, "Peter Duniho" <NpOeStPe...@xxxxxxxxxxxxxxxxxxxxx> wrote:

On Mon, 07 Jan 2008 06:11:04 -0800, Ryan <rchil...@xxxxxxxxx> wrote:

[...]

I'm having a hard time getting the delegate part of the interface where I can use it in this fashion. I hope this helps.

I have the same problem Jon does, which is that I don't know VB well enough to advise you on the exact syntax.

However, it appears that Wolf has provided a VB sample and I hope that is close enough to your needs. As far as the specific question goes, you can do what you are trying to if you make "Update" a property in the interface, defined using a delegate defined elsewhere (such as the same namespace in which the interface is declared, as I suggested earlier). Depending on the signature of the delegate, there may already be a delegate type defined in .NET that you can use (for example, "MethodInvoker" has no parameters and no return value).

Each class implementing your SignBoardControl interface would have to implement that property, by having a private field to store the actual value and having the property itself with a setter, and possibly a getter (if you want users of the interface to be able to inspect the current value of the property), that simply copies the value assigned to the property to the private field so it can be used later.

By the way, I recommend you follow the .NET naming convention of starting all interface names with a capital "I". This makes it more clear in the code that something is an interface. It's less of a problem in VB, since class declarations already make it clear by using "implements" instead of "inherits", but elsewhere it will be more obvious as well.

In C# it would look something like this:

```
interface ISignBoardControl
```

Re: Delegates and Interfaces

```
{
  MethodInvocation Update { set; }
}

class PlanActual : ISignBoardControl
{
  private MethodInvocation UpdateHandler;

  public MethodInvocation Update { set { UpdateHandler = value; } }

  // presumably somewhere else in the class you have code that calls
  the delegate,
  // using the UpdateHandler field or, if you also provide a getter
  for the property,
  // using the the property itself
}
```

then somewhere else in code you can assign Update1 or Update2 to the Update method as needed.

All of this assumes that you are using the same method for multiple classes that implement the interface. If there's a one-to-one relationship between the update method and the class it's used for, it seems to me it would make more sense to just make the method itself part of the interface, and implement the appropriate method in each class, rather than messing around with the delegate.

Even if you are using the same method for multiple classes, it may be better to create an intermediate class that's inherited by any class sharing the same method. In this context, the delegate is really more useful when at run-time you may have to reassign a different method according to some criteria. If the class will always have exactly one delegate method assigned to the delegate, then that's something that can be done statically at compile time and without the complication of managing the delegate.

Hope that helps.

Pete

Pete,

This got me on the right track. Each time I used delegate, the compiler kept telling I couldn't use it as a type and I didn't know how to properly use it. Thanks a bunch!!!

Ryan

.