

## Re: Using "custom attributes"

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.aspnet/2006-06/msg03373.html>

---

- *From:* "Ravi" <[sibravi@xxxxxxxxxx](mailto:sibravi@xxxxxxxxxx)>
  - *Date:* 23 Jun 2006 07:49:46 -0700
- 

Hi

I am analyzing the AspectDNG, I would like to weave my custom attribute, please provide me an example for the same.

Thanks in advance  
Ravi

Tor Bådshaug wrote:

First of all, do you know reflection well? Reflection is a mechanism that allows runtime inspection of your classes, their members and methods etc.

Reflection typically comes in handy when you need to handle unknown classes and assemblies (i.e. classes and assemblies over which you have no control) in a generic way.

A few samples where reflection is useful.

<http://msdn.microsoft.com/msdnmag/issues/02/08/NETReflection/default.aspx>  
<http://msdn.microsoft.com/msdnmag/issues/04/07/NETMatters/default.aspx>

Personally, I like to view attributes as an extension to the reflection concept. While reflection traditionally only deals with the implicit description of your code (classes, members, methods etc), attributes gives you the possibility to supplement the implicit descriptions of your code with explicit information.

Attributes only make sense in conjunction with one or more features. A particular set of attributes can be viewed as "the configuration of your class (assembly, method, etc...) for a feature or set of features". I am sure you have noticed that the .NET framework uses attributes rather extensively for features such as controlling design-time behavior, marshaling, serialization and security. Don't you think using declarative security with attributes is a convenient way to take advantage of .NET framework security features? You surely wouldn't want to swap that with manual coding, would you?

## Re: Using "custom attributes"

Another more precise example is the `Obsolete` attribute which you may use to signal that a method will be removed in a future version of your code. Then there is a feature in the compiler that will look out for this obsolete attribute and will provide a warning that you are using an obsolete method. There is even another attribute that allows you to define that the compiler should treat such a case as an error (broken build).

So I think understanding how and why the standard framework features are utilizing .NET standard attributes, and why it is useful, is a good starting point for understanding where custom attributes might help you.

To give some concrete examples I would, like with reflection, turn to generic tool and frameworks :

1) NUnit uses attributes, such as

```
[TestFixture]
public class YourClassContainingTests
```

to "mark" your test classes such that the NUnit runtime can figure out what tests to run. Notice that the same could have been achieved with reflection (such as JUnit for Java does), but attributes is simpler, clearer and less code intrusive (because an attribute-based approach can rely on explicit information about code, while a pure reflection-based model must define convention (test class must be a subclass of `TestCase` and test methods must start with "test" etc..)).

2) AspectDNG AOP Framework

You may not be familiar with AOP, but think of it as follows:

- You have your "code", to which you want to add features (In AOP, features are called aspects). A typical aspect could be logging.
- You code the features according to your AOP framework (independently of your "code")
- You apply the features to the "code" by specifying which aspect should be applied to what "code", either at runtime (dynamics AOP) or during an extra build step (static AOP).

AspectDNG uses static AOP, where the magic happens during the extra build steps to "weave" aspects into the "code". If a logging aspect is applied to some "code", the product of the extra build step is code with logging. With static AOP, there is nothing AOP'ish about the code at runtime.

So what has this AspectDNG thing to do with custom attributes?

Well, the thing is AspectDNG implements this extra build step through a tool that uses custom attributes. It accepts a "code" assembly and an aspect assembly as input. The aspects in the aspect assembly are marked with custom attributes to specify where the aspects should apply. The tool inspects these attributes and emits the "final code". For our example with a logging aspect, the "final code" emitted by the tool would be logging code inserted alongside your "code".

Take a look at <http://www.dotnetguru.org/sarl/aspectdng/> for more details

Re: Using "custom attributes"

about aspectdng as well as its custom attributes.

This became a lot more lengthy than I planned, but I hope it helps! :-)

Tor Bådshaug  
tor.badshaug [//at\\] bekk.no.