Advanced Tutorial, Learning Boo Kung-Fu

Now that you've got the basics down (you did read my first tutorial and studied up using the Language Guide right?) its time to witness the sleek, rumbling power of Boo Kung-Fu, but first, the requisite backstory:

Just a few days ago I was wrestling with a problem in parsing XML – my primary issue was the fact that I had many attributes represented in an XML attribute that had to be converted to many different kinds of types. For example, consider this XML element:

```
<Ninja color="#FF00FF" name="John Kho Kawn" style="Crazy Martial Arts" strength="71"
speed = "74" stamina = "65" />
```

The color attribute has to be converted to a type of System.Drawing.Color, but name and style are strings; strength, speed, and stamina, however, needed to be converted to a set of integers. Sounds easy right? Man, I wish. I realized as the XML became more complicated and trickier to parse with some attributes being present, and some not, that I was creating a huge mess of unmaintainable code. There had to be a simpler and more elegant way that was easily extended.

And there was: Boo.

My inital implementation of attributes-to-data structures was over 70 lines ⚠ long, and I hadn't even scratched the surface yet. Using my ultimate Boo fighting technique however, I produced something similar to the following code (this is just an example, sans error checking and junk). It took me about 25 minutes to write it up while doing the article, so it ain't half bad, says me.

```
import System.Reflection
import System
import System.Drawing
import System.Xml from System.Xml

class Ninja:
 public Name as string
 public Style as string
 public Speed as int
 public Strength as int
 public Stamina as int
 public Color as System.Drawing.Color
 def ToString():
  return "You see ${Name}, a ninja of the ${Style} with the stats
${Strength}/${Stamina}/${Speed}. He is ${Color}."
def ParseAttributesToValues(xmlNode as XmlNode, attributesToPropertiesConverter as
Hash):
 attributesToValue = { }
 //This will store an attribute -> equivilent data structure.
 for key as string in attributesToPropertiesConverter.Keys:
  //If this attribute exists in this xml node...
  if xmlNode.Attributes[key] is not null:
   //Get the converter for this attribute, then add it to the dictionary.
   //Use the same key, because that's clever.
   converter = attributesToPropertiesConverter[key] as callable

   attributesToValue.Add(key, converter(xmlNode.Attributes[key].Value)) if converter
is not null
   attributesToValue.Add(key, xmlNode.Attributes[key].Value) if converter is null
 //Return a dictoinary containing the data-structures associated with each attribute.
 return attributesToValue
def BindAttributeToProperty(propertyName, propertyValue, myObject):
 property = myObject.GetType().GetField(propertyName)
 property.SetValue(myObject, propertyValue)
```

```
def ToColor(str as string):
 //Color has built-in converter. ;)
 return
System.ComponentModel.TypeDescriptor.GetConverter(System.Drawing.Color).ConvertFrom(st
r)
def ToInt(str as string):
 //Also built in. +D
 return int.Parse(str)

xml = """
<Ninja color="#FF00FF" name="John Kho Kawn" style="Crazy Martial Arts" strength="71"
speed = "74" stamina = "65" />
"""
document = XmlDocument()
document.LoadXml(xml)

ninjaNode = document.SelectSingleNode("//Ninja")
attributesToProperties = {"color":"Color", "name":"Name", "strength":"Strength",
"style":"Style", "speed":"Speed", "stamina":"Stamina"}
attributesToPropertiesConverter = {"color": ToColor, "strength":ToInt,"speed":ToInt,
"stamina":ToInt, "name":null, "style":null}
//Grab the data structures produced by parsing the attributes.
attributesToValues = ParseAttributesToValues(ninjaNode,
attributesToPropertiesConverter)
ninja = Ninja()
for key in attributesToValues.Keys:
```

```
 BindAttributeToProperty(attributesToProperties[key], attributesToValues[key], ninja)
print ninja
```

It was pretty easy to follow along, but I'll guide you through it anyway to explain some of the cool stuff.

```
def ParseAttributesToValues(xmlNode as XmlNode, attributesToPropertiesConverter as
Hash):
```

ParseAttributes takes two parameters, an XmlNode that is hopefully rife with attributes, and a special dictionary that holds the attribute name to be converted as a key, and a pointer to the method that is to do the conversion, as the value. For example,

```
attribute = { "strength":ToInt }
```

would map the "strength" property to the ToInt() conversion method. Then, in the body of ParseAttributes, we extract the ToInt method when we find an instance of the "strength" attribute, then we feed the value of "strength" into ToInt(), which returns an integer representation of the strength attribute - then, glory be, we take that return integer and stuff it into another dictionary and, when we've all finished, we return this dictionary. For future reference, ToInt() invokes a method, while ToInt passes a pointer to the ToInt() method, which is called a "callable" in Boo. If you need to extract a method pointer from an arraylist or something that only returns objects, make sure to cast it to a "callable" first before trying to use it.

```
...
                      converter = attributesToPropertiesConverter[key] as callable

   attributesToValue.Add(key, converter(xmlNode.Attributes[key].Value)) if converter
is not null
   attributesToValue.Add(key, xmlNode.Attributes[key].Value) if converter is null
 //Return a dictoinary containing the data-structures associated with each attribute.
 return attributesToValue}}
```

The return dictionary holds the mapping from attribute -> data-structure, which probably looks like this internally:

```
{ "strength": 71 }
```

You'll note the special conditional - if I want to pick up a attribute's value but not convert it I simply pass 'null' as that attributes method pointer (The 'converter' variable) in the dictionary, and it is passed by string into the dictionary to be returned. If you were feeling really twisted, you could use,

```
attributesToValue.Add(key, (converter(xmlNode.Attributes[key].Value),
xmlNode.Attributes[key].Value)[if converter is null] )
```

instead and saved yourself one very precious line of code at the cost of your co-workers smacking you in the back of the head when they browse your source code.

Returning to our regularily schedule program, this is what it looks like when we pass in a null instead of a "ToInt" or a "ToColor":

```
{ "style" : null }
```

comes out to be, in the returned dictionary,

```
{ "style": "Crazy Martial Arts" }
```

Now, its time we just pick and choose keys from the dictionary and assign them to their relevant fields in a Ninja object, right? No, sorry – I'm lazy, and since I've gone this far, I might as well go all the way, baby!

Witness this method:

```
BindAttribute(propertyName, propertyValue, myObject):
```

This method is deceptively simple yet incredibly useful. It uses a feature of the .NET framework called "reflection" to dynamically assign values to fields possessed by "myObject." I won't go over it in any great detail; it is suffice to say that it works its magic in incredible and mysterious ways, so that the end result is the same as

```
myObject.propertyName = propertyValue
```

minus the clean syntax.

So, there you have it - a clean, easy, extensible way to convert attributes to data structures and assign these data structures to their proper classes, a real world solution to a real world problem I was struggling with – yeah, that's right, something **useful** 😊

Another way using Duck Typing

Note that there was another way to solve this design problem using a most excellent feature of Boo called Duck Typing and its interface, "IQuackFu," but that's another story for another day, my children.

See XML Object for an example.

Another way using XML Serialization

Nice example. It looks like something that .NET XML serialization may be good for (although it for some reason chokes on the color type, so I worked around that).

See XML Serialization for this version.