Language Mechanics

Estimated time for completion: 30 minutes

Overview:

In this lab you will investigate how dynamic typing can simplify parsing hierarchical data.

Goals:

- Build a dynamic type for parsing XML

Lab Notes:

N/A

Part 1: Creating a Dynamic XML Parser

One of the powerful features of dynamic languages is the ability to wire up parsing code at runtime for data being processed. The dynamic support in C# 4.0 brings this power to C# where we can simulate methods dynamically. In this part of the lab you will build an XML parser that supports dynamic parsing of an XML file

1. For this part of the lab you will be using the `DynamicParsing` project in the `LanguageMechanics` solution. It contains the data to be parsed in the form of an **order.xml** file. Take a look at the file and familiarize yourself with its contents
2. Add a new class to the project calling it `DynamicXmlParser`
3. This class needs to support dynamic behavior so derive it from `DynamicObject`
4. Add a member variable to the class of type `XElement` called `element`
5. Add a `public` constructor to the class that takes a file name as a `string`. Inside the constructor call `XElement.Load` on the file to initialize the `element` member variable
6. Add a `private` constructor that takes an `XElement` as a parameter and use this to initialize the `element` member variable. We will use this version internally during parsing

```
class DynamicXmlParser : DynamicObject
{
  XElement element;
  public DynamicXmlParser(string file)
  {
    element = XElement.Load(file);
  }

  private DynamicXmlParser(XElement el)
```

```
  {
    element = el;
  }
}
```

7. The idea of dynamic parsing is we want to allow the consumer to walk through the XML data using properties. We will dynamically create those properties as requested. In fact all we really have to do it to return the appropriate object as if that property had actually existed. To perform this dynamic property wire up we need to `override` the `TryGetMember` `virtual` method of `DynamicObject`
8. `TryGetMember` returns a `bool`. We need to return `true` if we support the property and `false` if we do not. In effect this means that if the sub-element they have asks for exists we return `true`, else we return `false`. So the first job it to try to get the requested sub-element. The name of the requested element is in the `binder` parameter's `Name` property. Use the `Element` method of the `element` member variable to attempt to get the sub-element
9. If the sub-element does not exist then the result will be `null`. In this case we need to initialize the `result` to `null` (it must be initialized as it is an out param) and return `false`
10. If the element does exist we need to create a new instance of the `DynamicXmlParser` wrapped around this sub-element and set the `result` to it. This allows the consumer to recursively use this propery syntax as they walk through the document. Don't forget to return `true` as we do support the property

```
public override bool TryGetMember(GetMemberBinder binder, out object result)
{
  XElement sub = element.Element(binder.Name);
  if (sub == null)
  {
    result = null;
    return false;
  }
  else
  {
    result = new DynamicXmlParser(sub);
    return true;
  }
}
```

11. Finally we need to provide a way for the consumer to actually get at the data in the document – the data being the text nodes and the attributes. We will override `ToString` for the text node and use an indexer for the attributes

Error! Reference source not found.

a. Override `ToString` and return the `Value` property of the `element` member variable
b. Add an indexer that returns a `string` and takes a `string` index. Pass the index to the `Attribute` method on the `element` member variable and return its `Value` property

```
public override string ToString()
{
  return element.Value;
}

public string this[string attr]
{
  get { return element.Attribute(attr).Value; }
}
```

12. We have now finished the parser so we can use it in `Main`. Create an instance of the `DynamicXmlParser` class passing the file path to the **order.xml** file. Assign this object to a `dynamic` local variable. This will enable the dynamic behavior as we use the variable
13. Use the parser to get the ordered attribute and some of the text nodes from the document

```
dynamic parser = new DynamicXmlParser(@"..\..\order.xml");
Console.WriteLine(parser["orderId"]);
Console.WriteLine(parser.customer.name);
Console.WriteLine(parser.orderItem.product);
Console.WriteLine(parser.orderItem.supplier);
```

14. Compile and test your code

Solutions

after\LanguageMechanics.sln