

# Creating Custom Dynamic Classes

---



**Jason Roberts**

.NET Developer

@robertsjason

dontcodetired.com



# Overview



Why custom dynamic classes?

The IDynamicMetaObjectProvider interface

The DynamicObject base class

Number of virtual methods

Custom dynamic HtmlElement class

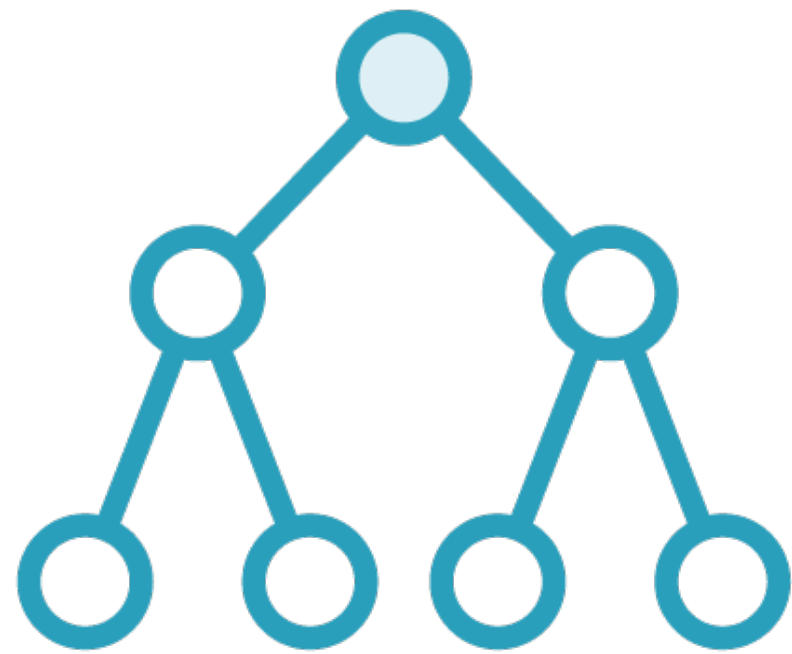
```
dynamic image = new HtmlElement("img");
```

```
image.src = "car.png";
```

```
string html = image.ToString();
```



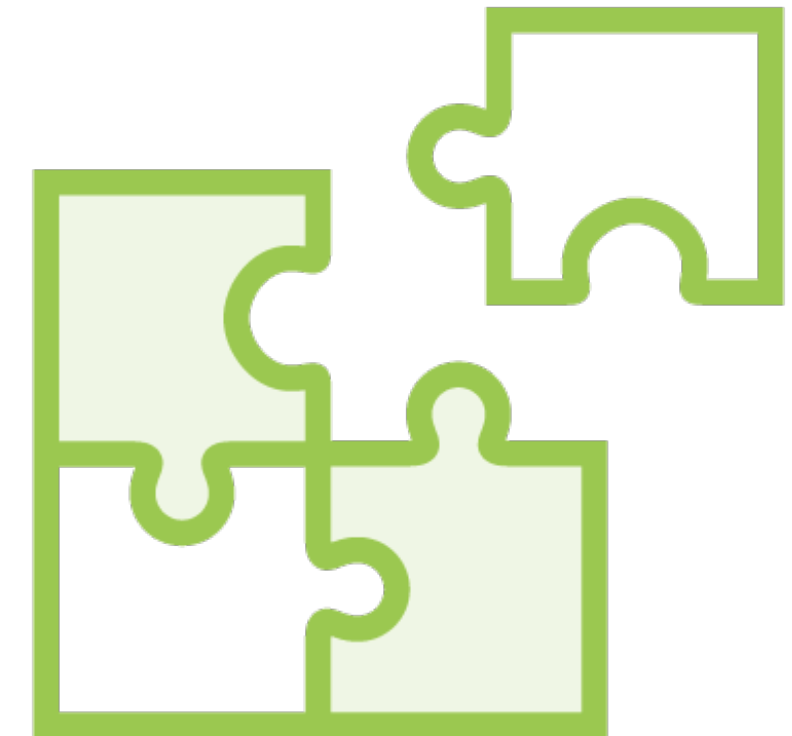
# Why Custom Dynamic Classes?



**Non static structures**  
**Highly fluid / untyped**  
**Unknown to compiler**  
**Known during runtime**  
**`ViewBag.Title = "xyz";`**



**Improved readability**  
**(non-dynamic) clutter**  
**Clearer intent**



**Interoperability**  
**IronPython**  
**IronRuby**



# The IDynamicMetaObjectProvider Interface

---



# The IDynamicMetaObjectProvider Interface

```
public sealed class ExpandoObject :  
    IDynamicMetaObjectProvider,  
    IDictionary<string, object>,  
    INotifyPropertyChanged
```

```
private sealed partial class DapperRow :  
    System.Dynamic.IDynamicMetaObjectProvider
```



# The IDynamicMetaObjectProvider Interface

```
public sealed class ExpandoObject :  
    IDynamicMetaObjectProvider,  
    IDictionary<string, object>,  
    INotifyPropertyChanged
```

```
private sealed partial class DataRow :  
    System.Dynamic.IDynamicMetaObjectProvider
```



# The DynamicObject Base Class

[SerializableAttribute]

```
public class DynamicObject : IDynamicMetaObjectProvider
```



# DynamicObject

The DynamicObject class enables you to define which operations can be performed on dynamic objects and how to perform those operations. For example, you can define what happens when you try to get or set an object property, call a method, or perform standard mathematical operations such as addition and multiplication. [MSDN]





# DynamicObject Virtual Methods

<code>public virtual bool TryInvokeMember(...)</code>	Calling a method
<code>public virtual bool TryGetMember(...)</code>	Getting property/field value
<code>public virtual bool TrySetMember(...)</code>	Setting property/field value
<code>public virtual bool TryGetIndex(...)</code>	Getting value by index
<code>public virtual bool TrySetIndex(...)</code>	Setting value by index
<code>public virtual bool TryUnaryOperation(...)</code>	Unary operators, e.g. !
<code>public virtual bool TryBinaryOperation(...)</code>	Binary operators, e.g. +
<code>public virtual bool TryConvert(...)</code>	Converting (casting) to other types
<code>public virtual bool TryInvoke(...)</code>	Invoking the object



# TryGetMember

```
private readonly Dictionary<string, string> _attributes =  
    new Dictionary<string, string>();  
  
public override bool TryGetMember(GetMemberBinder binder,  
    out object result)  
{  
    string attribute = binder.Name;  
  
    result = _attributes[attribute];  
  
    return true;  
}
```



# Summary



Why custom dynamic classes?

The IDynamicMetaObjectProvider interface

The DynamicObject base class

Overrode DynamicObject methods

TrySetMember() & TryGetMember()

GetDynamicMemberNames()

IDictionary<string, object>

TryInvokeMember()

TryInvoke()



Up Next:

Interoperating with Dynamic Languages

---

