

Changing Behavior	
<b>Changing returned values</b>	<pre>Isolate.WhenCalled(() =&gt; SomeClass.StaticMethod())     .WillReturn(10);  SomeClass myInstance = new SomeClass(); Isolate.WhenCalled(() =&gt; myInstance.ReturnFive()) .WillReturn(10);</pre>
<b>Throwing exceptions</b>	<pre>Isolate.WhenCalled(() =&gt; myInstance.ReturnFive()) .WillThrow(new Exception());</pre>
<b>Ignoring methods</b>	<pre>Isolate.WhenCalled(() =&gt; myInstance.VoidMethod()) .IgnoreCall();</pre>
<b>Changing Linq queries return values</b>	<pre>Isolate.WhenCalled(() =&gt; from c in customerList select c).WillReturn(     new List&lt;Customer&gt;     {         new Customer{ Id = 1, Name="Dave" },         new Customer{ Id = 2, Name="John" },         new Customer{ Id = 3, Name="Abe" }     });</pre>
<b>Custom return values</b>	<pre>Isolate.WhenCalled(()=&gt; myInstance.GetAgeOf ("AnyName")).DoInstead((callContext)=&gt; {     string name = callContext.Parameters[0] as string;     if(name == "John Smith") return 30;     return -1; });</pre>
<b>Conditional behavior</b>	<pre>Isolate.WhenCalled(()=&gt;myInstance.GetAgeOf ("John Smith"))     .WithExactArguments().WillReturn(30);</pre>
<b>Collections</b>	<pre>Isolate.WhenCalled(() =&gt; SomeClass.Products)     .WillReturnCollectionValuesOf(new[]     {         new ProductItem {Product = prodA, Quantity = 20},         new ProductItem {Product = prodB, Quantity = 40},         new ProductItem {Product = prodA, Quantity = 30}     });</pre>
<b>Replacing Ref and Out values</b>	<pre>int refParamToReplace = 6; string outParamToReplace = "FakeName"; Isolate.WhenCalled(() =&gt; myInstance.RefOutMethod(ref refParamToReplace,     out outParamToReplace)).WillReturn(10);</pre>
<b>Non-public Members</b>	<pre>Isolate.NonPublic.WhenCalled(myInstance, "PrivateMethod").WillReturn("Name"); Isolate.NonPublic.Property.WhenGetCalled(myInstance, "PrivateProp").WillReturn(5); Isolate.NonPublic.Property.WhenSetCalled(myInstance, "PrivateProp").IgnoreCall();</pre>

Creating Objects	
<b>Fake object creation</b>	<pre>var myInstance = Isolate.Fake.Instance&lt;SomeClass&gt;(Members.ReturnRecursiveFakes); // Default, returns fake objects. Members.CallOriginal // Methods not faked, constructor called. Members.ReturnNulls // Methods return null values Members.MustSpecifyReturnValues // Methods fail if called without setting // behavior</pre>
<b>Interfaces</b>	<pre>var myInstance = Isolate.Fake.Instance&lt;ISomeInterface&gt;(); // Also for abstract classes</pre>
<b>Faking static methods</b>	<pre>Isolate.Fake.StaticMethods &lt;SomeClass&gt;(); // Same arguments as Isolate.Fake.Instance.</pre>
<b>Future objects</b>	<pre>var myInstance = Isolate.Fake.Instance&lt;SomeClass&gt;(); Isolate.Swap.NextInstance&lt;SomeClass&gt;().With(myInstance);</pre>

Call Verification	
<b>Public methods</b>	<pre>Isolate.Verify.WasCalledWithAnyArguments (     ()=&gt;myInstance.ReturnFive()); Isolate.Verify.WasCalledWithExactArguments (     ()=&gt;myInstance.GetCustomerByName ("John")); Isolate.Verify.WasNotCalled ( ()=&gt;myInstance.ReturnFive());</pre>
<b>Non public</b>	<pre>Isolate.Verify.NonPublic.WasCalled(myInstance, "PrivateMethod"); Isolate.Verify.NonPublic.Property     .WasCalledGet(myInstance, "PrivateProp"); Isolate.Verify.NonPublic.Property     .WasCalledSet(myInstance, "PrivateProp"); Isolate.Verify.NonPublic     .WasCalled(myInstance, "GetCustomerByName").WithArguments("John");</pre>