Typemock™
The Unit Testing Company

Typemock Isolator Cheat Sheet

Arrange-Act-Assert C# API

Changing Behavior

Changing returned values

```
Isolate.WhenCalled(() => SomeClass.StaticMethod())
        .WillReturn(10);

SomeClass myInstance = new SomeClass();
Isolate.WhenCalled(() => myInstance.ReturnFive()).WillReturn(10);
```

Throwing exceptions

```
Isolate.WhenCalled(() => myInstance.ReturnFive()).WillThrow(new Exception());
```

Ignoring methods

```
Isolate.WhenCalled(() => myInstance.VoidMethod()).IgnoreCall();
```

Changing Linq queries return values

```
Isolate.WhenCalled(() => from c in customerList select c).WillReturn(
    new List<Customer>
    {
        new Customer{ Id = 1, Name="Dave" },
        new Customer{ Id = 2, Name="John" },
        new Customer{ Id = 3, Name="Abe" }
     });
```

Custom return values

```
Isolate.WhenCalled(()=> myInstance.GetAgeOf("AnyName")).DoInstead((callContext)=>
    {
        string name = callContext.Parameters[0] as string;
        if(name == "John Smith") return 30;
        return -1;
    });
```

Conditional behavior

```
Isolate.WhenCalled(()=>myInstance.GetAgeOf ("John Smith"))
    .WithExactArguments().WillReturn(30);
```

Collections

```
 Isolate.WhenCalled(() => SomeClass.Products)
    .WillReturnCollectionValuesOf(new[]
    {
        new ProductItem {Product = prodA, Quantity = 20},
        new ProductItem {Product = prodB, Quantity = 40},
        new ProductItem {Product = prodA, Quantity = 30}
    });
```

Replacing Ref and Out values

```
int refParamToReplace = 6;
string outParamToReplace = "FakeName";
Isolate.WhenCalled(() => myInstance.RefOutMethod(ref refParamToReplace,
    out outParamToReplace)).WillReturn(10);
```

Non-public Members

```
Isolate.NonPublic.WhenCalled(myInstance, "PrivateMethod").WillReturn("Name");
Isolate.NonPublic.Property.WhenGetCalled(myInstance,"PrivateProp").WillReturn(5);
Isolate.NonPublic.Property.WhenSetCalled(myInstance,"PrivateProp").IgnoreCall();
```

Creating Objects

Fake object creation

```
var myFake = Isolate.Fake.Instance<SomeClass>(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
```

Interfaces

```
var myFake = Isolate.Fake.Instance<ISomeInterface>();
// Also for abstract classes
```

Faking static methods

```
Isolate.Fake.StaticMethods <SomeClass>();
// Same arguments as Isolate.Fake.Instance.
```

Future objects

```
var myFake =
    Isolate.Swap.NextInstance<SomeClass>().WithRecursiveFake();
```

Faking dependencies

```
var myInstance = Isolate.Fake.Dependencies<SomeClass>();
var fakeDependency = Isolate.GetFake<ISomeInterface>(myInstance);
```

Call Verification

Public methods

```
Isolate.Verify.WasCalledWithAnyArguments(
    ()=>myInstance.ReturnFive());
Isolate.Verify.WasCalledWithExactArguments(
    ()=>myInstance.GetCustomerByName("John"));
Isolate.Verify.WasNotCalled(()=>myInstance.ReturnFive());
```

Non public Methods

```
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");
```