Rhino Mocks 3.3 Quick Reference

Record/Replay Syntaxes

Classic

```
Expect.Call(...);
mocks.ReplayAll();
...
mocks.VerifyAll();
```

using()

```
using (mocks.Record())
{
        Expect.Call(...);
}
using (mocks.Playback())
{
        ...
}
```

Fluent

```
With.Mocks(mocks).Expecting(delegate
{
        Expect.Call(...);
})
.Verify(delegate
{
        ...
});
```

Types of Mocks

	Single-type mock	Multi-interface mock
Strict replay	`CreateMock<T>()`	`CreateMultiMock<T>(params Type[])`
Loose replay	`DynamicMock<T>()`	`DynamicMultiMock<T>(params Type[])`
Loose replay, auto PropertyBehavior	`Stub<T>()`	None
Partial implementation (classes only)	`PartialMock<T>()`	`PartialMultiMock<T>(params Type[])`

Expectations and Setup

	Single-threaded playback	Multi-threaded playback
Strict replay	`Expect.Call(…)`	`Expect.On(…).Call(…)`
Strict replay, void return	`Expect.Call(delegate{…;})` Or, …; `LastCall`	`Expect.On(…).Call(delegate{…;})` Or, …; `LastCall.On(…)`
Loose replay (auto Repeat.Any)	`SetupResult.For(…)`	`SetupResult.On(…).For(…)`

Methods

```
Expect.Call(list.Contains(42)).Return(true);
```

Void Methods

```
Expect.Call(delegate{list.Add(42);});

Or, list.Add(42); // optionally followed by LastCall.
```

Out and Ref Parameters

```
string outParam;
Expect.Call(dict.TryGetValue("key", out outParam)).OutRef("value").Return(true);
```

Property Getters

```
Expect.Call(foo.Name).Return("Bob");
```

Property Setters

```
Expect.Call(foo.Name = "Bob");
```

Automatic Properties

```
Expect.Call(foo.Name).PropertyBehavior();
```

Event Subscription

```
Expect.Call(delegate{view.Load += null;}).Constraints(Is.NotNull());
```

Event Was Raised

```
Expect.Call(delegate{subscriber.Handler(source, EventArgs.Empty);});
```

Raising a Mock Event

```
IEventRaiser load =
	Expect.Call(delegate{view.Load+=null;}).IgnoreArguments().GetEventRaiser();
// switch to playback mode
load.Raise();
```

Throwing a Mock Exception

```
Expect.Call(delegate{file.Flush();}).Throw(new IOException("message"));
```

Delegates

```
Predicate<int> predicate = mocks.CreateMock<Predicate<int>>();
Expect.Call(predicate(42)).Return(true);
```

Custom Behavior

```
Expect.Call(delegate{list.Add(0);}).IgnoreArguments()
	.Do((Action<int>)delegate(int item) {
		if (item < 0) throw new ArgumentOutOfRangeException();
	});
```

Assert Messages

```
Expect.Call(delegate{file.Flush();}).Message("File must be flushed before closing");
```

Ordered and Unordered Expectations

Note: the default recorder state is unordered.

```
using(mocks.Ordered())
{
	Expect.Call(databaseManager.BeginTransaction());
	using(mocks.Unordered())
	{
		Expect.Call(accountOne.Withdraw(1000));
		Expect.Call(accountTwo.Deposit(1000));
	}
}
```

Call Repetition

```
Repeat.Once() // the default for Expect.Call()
Repeat.Any()  // the default for SetupResult.For()
Repeat.Never()
Repeat.AtLeastOnce()
Repeat.Twice()
Repeat.Times(3)
Repeat.Times(4, int.MaxValue)
```

Argument Constraints

```
IgnoreArguments() // removes all argument constraints
```

Constraints Example:

```
Expect.Call(file.Read(null, 0, 0))
.Constraints(Property.Value("Length", 4096), Is.Equal(0), Is.GreaterThan(0) &&
Is.LessThan(4096));
```

This expects a call to file.Read where the buffer length is 4096, the offset is 0, and the count is between 0 and 4096.

```
Is.Anything() // very useful for constraining only some arguments
Is.Equal(3)
Is.NotEqual(42)
Is.Same(obj)
Is.NotSame(obj)
Is.Null()
Is.NotNull()
Is.GreaterThan(3)
Is.GreaterThanOrEqual(3)
Is.LessThan(3)
Is.LessThanOrEqual(3)
Is.Matching<IList<int>>(delegate(IList<int> list) {
       return list.Count == 10 && list.IndexOf(3) > 2;
})
Is.TypeOf<IDisposable>()

Property.Value("Length", 0)
Property.IsNull("InnerException")
Property.IsNotNull("InnerException")
Property.ValueConstraint("Items", Property.Value("Count", 0))

List.Count(Is.Equal(0));
List.Element(2, Text.StartsWith("Three"));
List.Equal(new int[] {1, 2, 3}); // object[] {1, 2, 3} would be accepted
List.IsIn("Two"); // a parameter of string[] {"One", "Two"} would be accepted
List.OneOf(new int[] {2, 3}); // a parameter of 3 would be accepted

Text.StartsWith("Hello");
Text.EndsWith("World");
Text.Contains("or");
Text.Like(@"^[2-9]\d{2}-\d{3}-\d{4}$");
```