Typemock™
The Unit Testing Company

Isolator++ Cheat Sheet

Change behavior	

Changing return values	`WHEN_CALLED(SomeClass::StaticMethod()).Return(10); // static`
	
	`SomeClass myClass;`
	`WHEN_CALLED(myClass.Method()).Return(10); // instance`
	
	`FAKE_GLOBAL(fopen);`
	`WHEN_CALLED(fopen(_, _)).Return(NULL); // global C style`

Deep chaining	`WHEN_CALLED(SomeClass::StaticMethod()->OtherMethod()->Inside()).Return(10);`
Ignoring Methods	`WHEN_CALLED(SomeClass::StaticMethod()).Ignore();`
Throwing Exceptions	`exception problem("Something has gone wrong!");` `WHEN_CALLED(SomeClass::StaticMethod()).Throw(&problem);`

Custom Return Value	`static int CustomValueWithData(int a)`
	`{`
	`  if (a == 5) return 30;`
	`  return 10000;`
	`}`
	`WHEN_CALLED(SomeClass::StaticMethod(_)).DoStaticOrGlobalInstead(CustomValue WithData,<user-data>).`

Conditional behavior	`WHEN_CALLED(SomeClass::StaticMethod(EQ("US")).Return(10);`

Out args	`SYSTEMTIME fakeTime;`
	`WHEN_CALLED(GetSystemTime(RET(&fakeTime))).Ignore();`
	
	`WHEN_CALLED(GetSystemTime(RET_IF(EQ(…), &fakeTime))).Ignore(); // conditional`

Non public methods	`PRIVATE_WHEN_CALLED (_, MyClass::staticPrivateMethod).Ignore(); // static`
	
	`SomeClass myClass;`
	`PRIVATE_WHEN_CALLED(myClass,privateMethod).Return(10); // instance`

Conditions		
-	All arguments are ok	`WHEN_CALLED(fake->Foo(_)).Return(1);`
Any value - ANY_VAL	All arguments are ok (value types)	`WHEN_CALLED(fake->Foo (ANY_VAL(Type))).Return(1);`
Any ref - ANY_REF	All arguments are ok (byref)	`WHEN_CALLED(fake->Foo(ANY_REF(Type))).Return(1);`
Equal - EQ	Arg must equal (using == operator)	`WHEN_CALLED(fake->Foo(EQ(100))).Return(1);`
Not equal - NE	Arg must not equal	`WHEN_CALLED(fake->Foo(NE(3))).Return(1);`
Less than - LT	Arg is smaller than	`WHEN_CALLED(fake->Foo(LT(5))).Return(1);`
Less or equal - LE	Arg is smaller or equals	`WHEN_CALLED(fake->Foo(LE(4))).Return(1);`
Greater than - GT	Arg is greater than	`WHEN_CALLED(fake->Foo(GT(10.2))).Return(1);`
Greater or equal - GE	Arg is greater or euqals	`WHEN_CALLED(fake->Foo(GE(1))).Return(1);`
Lambda function - IS	Arg must pass lambda function	`WHEN_CALLED(fake->Foo(IS(<char*>([] (char* s){return !strcmp(s, "typemock"); }))).Return(1);`
Lambda function by ref – IS_REF	By Ref Arg must pass lambda function	`WHEN_CALLED(fake->Foo(IS_REF(<const char*>([] (const char* s){return !strcmp(s, "typemock");}))).Return(1);`
Assign out value – RET_IF	Assign out value, if condition is true	`WHEN_CALLED(fake->Foo(RET_IF(EQ(&value), &out)).Return(1);`
Assign value with condition – BY_REF	Use in condition macros to assign value directly	`WHEN_CALLED(fake->Foo(RET_IF(EQ(BY_REF("typemock")), &out)).Return(1);`

Creating objects	
Create a fake object	`SomeClass * fakeClass = FAKE<SomeClass>();`
Pure Virtual	`IInterface* fake = FAKE<IInterface>();`
Future Objects	`SomeClass* classHandle = FAKE_ALL<SomeClass>();`

Acting on objects	

Call private method	`bool ret = false;`
	`ISOLATOR_INVOKE_FUNCTION(ret, _, SomeClass::StaticMethod, arg1, arg2..);`
	
	`ISOLATOR_INVOKE_FUNCTION(ret, myClass , MyMethod, arg1, arg2...);`

Set variable	`ISOLATOR_SET_VARIABLE(_,SomeClass::m_static, 10);`
	
	`SomeClass* myClass = new SomeClass ();`
	`ISOLATOR_SET_VARIABLE(myClass, m_id, 10);`
	
	`ISOLATOR_SET_VARIABLE(_,staticVariable, 10);`

Get variable	`int memberValue;`
	`ISOLATOR_GET_VARIABLE(_,SomeClass::m_static, memberValue);`
	
	`SomeClass* myClass = new SomeClass ();`
	`ISOLATOR_GET_VARIABLE(myClass, m_id, memberValue);`
	
	`ISOLATOR_GET_VARIABLE(_,staticVariable, memberValue);`

Call verification	

Public methods	`ASSERT_WAS_CALLED(SomeClass::StaticMethod());`
	
	`ASSERT_WAS_CALLED(myClass.Method());`

Private methods	`PRIVATE_ASSERT_WAS_CALLED((_, MyClass::staticPrivateMethod);`
	
	`PRIVATE_ASSERT_WAS_CALLED(myClass,Method);`

Conditional	`ASSERT_WAS_CALLED(SomeClass::StaticMethod(EQ("US"));`
	
	`PRIVATE_ASSERT_WAS_CALLED(myClass, Method, EQ("US"));`

For more examples go to our documentation here.