Typemock™
The Unit Testing Company

Isolator++ Cheat Sheet

Change Behavior	

Changing return values	**Version 5 API:**
	`a.CallTo(SomeClass::StaticMethod()).WillReturn(10); // static`
	
	`SomeClass myClass;`
	`a.CallTo(myClass.Method()).WillReturn(10); // instance`
	
	`a.Testable.GlobalFunction(fopen);`
	`a.CallTo(fopen(A::Any(), A::Any())).WillReturn(NULL); // global C style`
	
	Version 4 API:
	`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	**Version 5 API:**
	`a.CallTo(SomeClass::StaticMethod()->OtherMethod()->Inside()).WillReturn(10);`
	
	Version 4 API:
	`WHEN_CALLED(SomeClass::StaticMethod()->OtherMethod()->Inside()).Return(10);`

Ignoring Methods	**Version 5 API:**
	`a.CallTo(SomeClass::StaticMethod()).WillBeIgnored();`
	
	Version 4 API:
	`WHEN_CALLED(SomeClass::StaticMethod()).Ignore();`

Throwing Exceptions	**Version 5 API:** exception problem("Something has gone wrong!"); a.CallTo(SomeClass::StaticMethod()).WillThrow(&problem); **Version 4 API:** 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;
	}
	
	Version 5 API:
	a.CallTo(SomeClass
	::StaticMethod(_)).DoStaticOrGlobalInstead
	(CustomValueWithData);
	
	Version 4 API:
	WHEN_CALLED(SomeClass
	::StaticMethod(_)).WillDoInstead (CustomValueWithData);

Conditional behavior	**Version 5 API:** a.CallTo(SomeClass::StaticMethod(A::Eq("US")).WillReturn(10) ; **Version 4 API:** WHEN_CALLED(SomeClass::StaticMethod(EQ("US")).Return(10);

Out args	**Version 5 API:**
	SYSTEMTIME fakeTime;
	a.CallTo(GetSystemTime(A::SetOut(&fakeTime).WhenIn())).WillB
	eIgnored(); // conditional
	
	Version 4 API:
	SYSTEMTIME fakeTime;
	WHEN_CALLED(GetSystemTime(RET(&fakeTime))).Ignore();
	WHEN_CALLED(GetSystemTime(RET_IF(EQ(…),
	&fakeTime))).Ignore(); // conditional

Non public methods	**Version 5 API:**
	`a.CallToPrivate(A::Global(MyClass::staticPrivateMethod), A::Any()).WillBeIgnored(); // static`
	
	`SomeClass myClass;`
	`a.CallToPrivate(A::Member(myClass,privateMethod)).WillReturn(10); // instance`
	
	Version 4 API:
	`PRIVATE_WHEN_CALLED (_, MyClass::staticPrivateMethod).Ignore(); // static`
	
	`SomeClass myClass;`
	`PRIVATE_WHEN_CALLED(myClass,privateMethod).Return(10); // instance`

Conditions		

Not equal – NE	Arg must not equal	**Version 5 API:**
		
		`a.CallTo(fake->Foo(A::Ne(3))).WillReturn(1);`
		
		Version 4 API:
		
		`WHEN_CALLED(fake->Foo(NE(3))).Return(1);`

Less than – LT	Arg is smaller than	**Version 5 API:**
		
		`a.CallTo(fake->Foo(A::Lt(5))).WillReturn(1);`
		
		Version 4 API:
		
		`WHEN_CALLED(fake->Foo(LT(5))).Return(1);`

Less or equal – LE	Arg is smaller or equals	**Version 5 API:**
		
		`a.CallTo(fake->Foo(A::Le(4))).WillReturn(1);`
		
		Version 4 API:
		
		`WHEN_CALLED(fake->Foo(LE(4))).Return(1);`

Lambda function – IS	Arg must pass lambda function	**Version 5 API:**
		
		`a.CallTo(fake->Foo(A::Matches([]`
		
		`(char* s){return !strcmp(s, “typemock”);`
		
		`})).WillReturn(1);`
		
		Version 4 API:
		
		`WHEN_CALLED(fake->Foo(IS(([]`
		
		`(char* s){return !strcmp(s, “typemock”);`
		
		`}))).Return(1);`

Lambda function by ref – IS_REF	By Ref Arg must pass lambda function	**Version 5 API:**
		
		`a.CallTo(fake->Foo(A::MatchesRef([]`
		
		`( const char* s)`
		
		`(char* s){return !strcmp(s,`
		
		`"typemock"); })).WillReturn(1);`
		
		Version 4 API:
		
		`WHEN_CALLED(fake->Foo(IS_REF ( < const char*>([]( const char* s)(char* s) {return !strcmp(s,`
		
		`"typemock"); }))).Return(1);`

Greater than – GT	Arg is greater than	**Version 5 API:**
		
		`a.CallTo(fake->Foo(A::Gt(10.2))) .WillReturn(1);`
		
		Version 4 API:
		
		`WHEN_CALLED(fake->Foo(GT(10.2))) .Return(1);`

Greater or equal – GE	Arg is greater or equals	**Version 5 API:**
		
		`a.CallTo(fake->Foo(A::Ge(1))).WillReturn( 1);`
		
		Version 4 API:
		
		`WHEN_CALLED(fake->Foo(GE(1))).Return(1);`

Equal – EQ	Arg must equal (using == operator)	**Version 5 API:**
		
		`a.CallTo(fake->Foo(A::Eq(100)))`
		`.WillReturn(1);`
		
		Version 4 API:
		
		`WHEN_CALLED(fake->Foo(EQ(100)))`
		`.Return(1);`

Assign value with condition – BY_REF	Use in condition macros to assign value directly	**Version 5 API:**
		
		`a.CallTo(fake->Foo(A::SetOut(A::Eq`
		`(“typemock”),`
		`&out).WhenIn())).WillReturn(1);`
		
		Version 4 API:
		
		`WHEN_CALLED(fake->Foo(RET_IF(EQ(BY_REF`
		`(“typemock”)), &out)).Return(1);`

Assign out value – RET_IF	Assign out value, if condition is true	**Version 5 API:**
		
		`a.CallTo(fake->Foo(A::SetOut(A::Eq(&value`
		`), &out).WhenIn())).WillReturn(1);`
		
		Version 4 API:
		
		`WHEN_CALLED(fake->Foo(RET_IF(EQ(&value),`
		`&out)).Return(1);`

Any value – ANY_VAL	All arguments are ok (value types)	**Version 5 API:**
		
		`a.CallTo(fake->Foo`
		`(A::Any())).WillReturn(1);`
		
		Version 4 API:
		
		`WHEN_CALLED(fake->Foo`
		`(ANY_VAL(Type))).Return(1);`

Any ref – ANY_REF	All arguments are ok (byref)	**Version 5 API:**
		
		`a.CallTo(fake->Foo(A::Any()))`
		`.WillReturn(1);`
		
		Version 4 API:
		
		`WHEN_CALLED(fake->Foo(ANY_REF(Type)))`
		`.Return(1);`

_	All arguments are ok	**Version 5 API:**
		
		`a.CallTo(fake->Foo(A::Any()))`
		`.WillReturn(1);`
		
		Version 4 API:
		
		`WHEN_CALLED(fake->Foo(_)).Return(1);`

Creating objects	

Create a fake object	**Version 5 API:**
	
	`SomeClass * fakeClass =`
	`a.Fake.Instance<SomeClass>();`
	
	Version 4 API:
	
	`SomeClass * fakeClass = FAKE<SomeClass>();`

Pure Virtual	**Version 5 API:**
	
	`IInterface* fake =`
	`a.Fake.Instance<IInterface>();`
	
	Version 4 API:
	
	`SomeClass * fakeClass = FAKE<SomeClass>();`

Future Objects	**Version 5 API:**
	
	`SomeClass* classHandle =`
	`a.Fake.All<SomeClass>();`
	
	Version 4 API:
	
	`SomeClass* classHandle = FAKE_ALL<SomeClass>();`

Acting on objects	

Call private method	**Version 5 API:**
	`bool ret = false;`
	`ret = a.Invoke(A::Global(SomeClass::StaticMethod), arg1, arg2..);`
	
	`ret = a.Invoke(A::Member(myClass , MyMethod), arg1, arg2…);`
	
	Version 4 API:
	`bool ret = false;`
	`ISOLATOR_INVOKE_FUNCTION(ret, _, SomeClass::StaticMethod, arg1, arg2..);`
	
	`ISOLATOR_INVOKE_FUNCTION(ret, myClass , MyMethod, arg1, arg2…);`

Set variable	**Version 5 API:**
	`a.Variable.Set(SomeClass::m_static, 10);`
	
	`SomeClass* myClass = new SomeClass ();`
	`a.Variable.Set(myClass, "m_id", 10);`
	
	`a.Variable.Set(staticVariable, 10);`
	
	Version 4 API:
	`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	**Version 5 API:**
	`int memberValue;`
	`memberValue = a.Variable.Get(SomeClass::m_static);`
	
	`SomeClass* myClass = new SomeClass ();`
	`memberValue = a.Variable.Get(myClass, "m_id:);`
	
	`memberValue = a.Variable.Get(staticVariable);`
	
	Version 4 API:
	`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);`

Calling verification	

Public methods	**Version 5 API:**
	`a.CallTo(SomeClass::StaticMethod()).VerifyWasCalled();`
	
	`ASSERT_WAS_CALLED(myClass.Method());`
	
	Version 4 API:
	`ASSERT_WAS_CALLED(SomeClass::StaticMethod());`
	
	`ASSERT_WAS_CALLED(myClass.Method());`

Private methods	**Version 5 API:**
	`a.CallTo((A::Any(),`
	`MyClass::staticPrivateMethod).VerifyWasCalled();`
	
	Version 4 API:
	`PRIVATE_ASSERT_WAS_CALLED(myClass,Method);`

Conditional	**Version 5 API:**
	`a.CallTo(SomeClass::StaticMethod(A::Eq("US")).VerifyWasCalled();`
	
	`a.CallTo(myClass, Method, A::Eq("US")).VerifyWasCalled();`
	
	Version 4 API:
	`ASSERT_WAS_CALLED(SomeClass::StaticMethod(EQ("US"));`
	
	`PRIVATE_ASSERT_WAS_CALLED(myClass, Method, EQ("US"));`

For more examples, go to our documentation here.