Running CMock

CMock is a Ruby script and class. You can therefore use it directly from the command line, or include it in your own scripts or rakefiles.

Mocking from the Command Line

After unpacking CMock, you will find CMock.rb in the 'lib' directory. This is the file that you want to run. It takes a list of header files to be mocked, as well as an optional yaml file for a more detailed configuration (see config options below).

For example, this will create three mocks using the configuration specified in MyConfig.yml:

```
ruby cmock.rb -oMyConfig.yml super.h duper.h awesome.h
```

And this will create two mocks using the default configuration:

```
ruby cmock.rb ../mocking/stuff/is/fun.h ../try/it/yourself.h
```

Mocking From Scripts or Rake

CMock can be used directly from your own scripts or from a rakefile. Start by including cmock.rb, then create an instance of CMock. When you create your instance, you may initialize it in one of three ways.

You may specify nothing, allowing it to run with default settings:

```
cmock = CMock.new
```

You may specify a YAML file containing the configuration options you desire:

```
cmock = CMock.new('../MyConfig.yml')
```

You may specify the options explicitly:

```
cmock = CMock.new('plugins' => ['cexception','ignore'],'mock_path' => 'my/mocks/')
```

Config Options:

The following configuration options can be specified in the yaml file or directly when instantiating.

Option	**Purpose**
plugins	An array of which plugins to enable. 'expect' is always active. 'cexception' and 'ignore' are also currently available.
mock_path	The directory where you would like the mock files generated to be placed.
includes	An array of additional include files which should be added to the mocks. Useful for global types and definitions used in your project.
tab	What does tab mean in your project? By default it's a pair of spaces.
expect_call_count_type	Used internally by CMock... but maybe you don't like int's for some reason?
ignore_bool_type	Used internally by Ignore plugin... but maybe you have a better bool type?
cexception_include	Tell cexception plugin where to find Exception.h... only need to define if it's not in your build path already.
cexception_throw_type	Tell cexception what type you are “throwing” around in your application. It assumes an int

Generated Mock Module Summary

In addition to the mocks themselves, CMock will generate the following functions for use in your tests. The expect functions are always generated. While the ignore and cexception functions are only generated if those plugins are enabled:

Expect:

Original Function	**Generated Mock Function**
void func(void)	void func_Expect(void)
void func(params)	void func_Expect(expected_params)
retval func(void)	void func_ExpectAndReturn(retval_to_return)
retval func(params)	void func_ExpectAndReturn(expected_params, retval_to_return)

CException:

Original Function	**Generated Mock Function**
void func(void)	void func_ExpectAndThrow(value_to_throw)
void func(params)	void func_ExpectAndThrow(expected_params, value_to_throw)
retval func(void)	void func_ExpectAndThrow(value_to_throw)
retval func(params)	void func_ExpectAndThrow(expected_params, value_to_throw)

Ignore:

Original Function	**Generated Mock Function**
void func(void)	void func_Ignore(void)
void func(params)	void func_Ignore(void)
retval func(void)	void func_IgnoreAndReturn(retval_to_return)
retval func(params)	void func_IgnoreAndReturn(retval_to_return)