Ceedling: Working With IDE’s

Before attempting to get Ceedling working with an IDE, we strongly encourage you to get everything configured properly to be used as a command line tool. Ceedling is primarily a command line tool. It works with most compilers and simulators that are command-line accessible. Because IDE’s bring with them their own complications to an already complex toolchain, making sure everything works on your command line first will go a long way towards easing your transition.

Once running properly, then the necessary tweaks can be made to fit it into your workflow. It often doesn’t take much in order for it to fit snugly enough that it becomes natural. The following things will go a long ways towards making unit testing feel natural inside your IDE:

1. Trigger tests with hotkeys and/or buttons and view results.
2. Finding errors quickly.
3. Making your test and source modules feel related.
4. Handy code completion for faster development.

Let’s talk about each of these quick:

Triggers

Most modern IDE’s have the ability to launch command line tools based on a hotkey or custom button. We want to use these to trigger tests.

Ideally, we can trigger Ceedling to run ALL the tests in the suite OR we can trigger Ceedling to run the test associated with the module we are currently editing. It is the latter that is most interesting. We’ve found it most useful to have the IDE trigger a test when looking at a test file, the related source file, or even the related header file. For example, if the user is currently editing Cookies.c and triggers a single test, we would expect the tests for TestCookies.c to build and run with Ceedling.

There are other features that might be useful to trigger from the IDE as well. For example, we may want to have hotkeys or buttons to support a clobber, module generation, and so on.

Locating Errors

Many IDE’s have the ability to parse command line output and to make hyperlinks to related lines of code. This will get us the second item on our list. If we can format Ceedling’s output in a format that the IDE recognizes, it will naturally create links to the test file and line when there are errors. Much like compiler and linker errors, this greatly simplifies the developer’s job of hunting down failures.

Making Tests and Code Related
This one is subtle, but incredibly useful.

We already mentioned one instance above. It's convenient to have our system understand that a request to test a source file, really means that we want Ceedling to run the TEST file related to that source module. This is so useful, in fact, that it comes baked into Ceedling by default. All of these will act exactly the same when executed:

```
Ceedling test:TestPie.c
Ceedling test:Pie.c
Ceedling test:Pie.h
Ceedling test:Pie
```

Similarly, when editing a test file, it's very convenient to have a hotkey which will open the corresponding source module. When editing a source module, it's very convenient to have a hotkey which will open the corresponding test module. The ability to quickly toggle between the two helps to keep our test-implement-refactor loop as quick as possible.

Code Completion
The last major boost to our productivity doesn't depend on Ceedling at all. Many IDE's provide the option of creating boilerplate code and injecting it into our source with hotkeys. Instead of writing the same empty test functions over and over again, and then fleshing them out, we can instead hit a few keys and the IDE will insert a test template. Similarly, it's convenient to have a shorthand for test assertions and ignores… anything to allow us to generate tests more quickly.

Sublime Text

If you're a lover of Sublime Text, or are framework agnostic and are looking for advice, you're in luck! There is a Ceedling plugin for Sublime Text 3 (or 2, if you're still using that). As is the norm for Ceedling, the Ceedling plugin for Sublime Text provides numerous hotkeys. Like most Ceedling plugins, these hotkeys are completely reconfigurable to whatever key combinations you desire.

Triggers

The plugin provides numerous options for triggering tests:

Trigger	Default Keymap
Test Current File	F7
Test All Changed Files	Ctrl+F7
Test All	Ctrl+Alt+F7
Clobber All	Ctrl+Alt+Shift+x

Locating Errors

When Ceedling runs tests, the plugin collects the output in Sublime Text's Console area. All results are dumped here in whatever format Ceedling is configured for. If the developer sticks with standard pretty print (the default), then there is a working hotkey for jumping from a failure and/or ignore line and jumping to a file.

Feature	Default Keymap
Jump to Failure / Ignore	Ctrl+Shift+b o

Making Tests and Code Related

The plugin gives a number of options for switching between files automatically. There are some which open a specific related file, like the test for a given file. There are also a couple options for cycling between the related files. Make use of whichever feel most convenient to you.

File To Open	Default Keymap
Related Header File	Ctrl+Alt+h
Related Source File	Ctrl+Alt+s
Related Test File	Ctrl+Alt+t
Cycle between Source, Test, & Header	Ctrl+period
Toggle between Source & Test	Ctrl+Alt+period

Code Completion

The Ceedling plugin for Sublime Text provides a hotkey for running the module creator (thereby generating source-header-test triads or entire patterns of triads). It also comes stocked with some useful completions for commonly used idioms, many of which will walk you through completion with prompts.

Template	Shorthand
_Expect(params);	.e
_ExpectAndReturn(params);	.er
TEST_ASSERT_EQUAL(exp, act);	ae
TEST_ASSERT_EQUAL_MESSAGE(exp, act, msg);	aem
TEST_ASSERT_EQUAL_HEX(exp, act);	ah
TEST_ASSERT_EQUAL_HEX_MESSAGE(exp, act, msg);	ahm
TEST_ASSERT_EQUAL_HEX8(exp, act);	ah8
TEST_ASSERT_EQUAL_HEX8_MESSAGE(exp, act, msg);	ah8m
TEST_ASSERT_EQUAL_HEX16(exp, act);	ah16
TEST_ASSERT_EQUAL_HEX16_MESSAGE(exp, act, msg);	ah16m
TEST_ASSERT_TRUE(exp);	at
TEST_ASSERT_TRUE_MESSAGE(exp, msg);	atm
TEST_ASSERT_FALSE(exp);	af
TEST_ASSERT_FALSE_MESSAGE(exp, msg);	afm
TEST_ASSERT_EQUAL_MEMORY(exp, act, len);	am
TEST_ASSERT_EQUAL_MEMORY_MESSAGE(exp, act, ...);	amm
TEST_IGNORE();	ig
TEST_IGNORE_MESSAGE(msg);	igm
void test_FUNCTION_should_BEHAVIOR(void) { … }	test
void test_FUNCTION_should_BEHAVIOR(void) { TEST_FAIL… }	testf
void test_FUNCTION_should_BEHAVIOR(void) { TEST_IGN… }	testi

Bonus!

There are a number of bonus features that the Ceedling plugin provides. Here are a few of them:

Function	Default Keymap
Open Ceedling Project File	F1
Build Release	F5

Eclipse

As you are probably aware, Eclipse is the basis of many embedded IDE's in the world. It was therefore one of the earliest IDE's to be integrated with Ceedling. Eclipse, however, with its many variations and combinations of plugins, has continued to be a challenging platform to provide a reliable integration. What we describe here works well at the time of this printing, but that sadly does not guarantee that it will continue to work in the future. If you run into any problems, please check out throwtheswitch.org/eclipse for the latest updates on how to keep this integration working. If you learn or invent tips of your own, we'd love to hear them.

Before proceeding through this guide, you will want to make sure that you have both Eclipse CDT (for C and C++ development) and the CDT Unit Testing Framework installed.

When using Eclipse with Sublime, you're going to want to create your projects as C/C++ projects that are Makefile based with Existing Code. We won't be using a makefile, in actuality, but it suggests to Eclipse that we're going to be driving much of the work from the command line.

You'll want to configure your Toolchain for Indexer Settings, you'll want to select a native gcc (no matter if you are cross compiling for your tests or not). This will give it the ability to index your project files automatically in the background. If none are present, you may choose NONE.

Triggers

In order to trigger tests, we're going to use Eclipse's Run As feature. We long-click the Run As button and select Run Configurations… on the navigation tree. We select C/C++ Unit and Add a new configuration. We name this new test to All Tests.

The Main tab of your new configuration allows you to enter the location of your C/C++ application. We're going to be calling Ceedling, like so:

`~/.rvm/gems/ruby-2.0.0-p643/gems/ceedling-0.19.0/bin/ceedling`
(or where ever your copy of Ceedling is)

On this tab you are also going to want to select Disable Auto Build, because there is no need to build anything in particular before calling ceedling.

Next, on the Arguments tab, add arguments clobber on one line and test:all on the next. You will be using the default working directory.

On the C\C++ Testing tab, select Google Tests Runner as the Test Runner. I know... it's sad that there is no support for Unity directly, yet... but luckily Ceedling is already good at pretending to be Google Test... good enough that Eclipse shouldn't notice the difference. We just have to enable it. Open your Ceedling project.yml file. Scroll down until you see the section marked plugins and add the following plugin. You should comment out all other display plugins, like pretty-print:

```
:plugins:
    :enabled:
        #- stdout_pretty_tests_report
        - stdout_gtestlike_tests_report
```

Back in Eclipse, on the Environment tab, Add an environment variable. Name it TERM and assign it a value of xterm.

The rest of the tabs are probably fine with defaults, so you can select Apply to finish this particular Run Configuration.

So that's useful... but what if we want to test just a single module? Let's do that quick.

Right-Click on the All Tests item and select Duplicate. We're going to name this one Test This. On the Arguments tab, update the list to just one item:

```
test:${selected_resource_name}
```

Again press Apply. Press OK.

You can now verify that your new target works. Selecting All Tests from the menu should kick off all your tests. When finished, it should show you the results in a C\C++ Unit View.

Running This Test is slightly more tricky than it should be. You need to make sure your editor window is selected and that you are viewing either the test file, it's related source module, or the

header for that source module. If you then select This Test, it should kick off the test just for that.
This is a feature that would desperately like a hot-key. Unfortunately it appears that Eclipse does not provide a stub for a hot-key for Run As _. The closest thing we found was Run Configuration which seems to be Ctrl+Alt+R by default (Cmd+Alt+R for Mac users). This shows the configuration window. You can then use the arrow keys to select the one you want and then press ENTER.

NOTE: If you are on a Mac and when you run your tests, and it complains about character encoding, this may be due to a quirk in how Eclipse launches from the Finder. Launch it once from the command line, and it should fix itself from there. See this bug for reference.

Locating Errors

When tests are triggered and completed, it should show the results in a C\C++ Unit View. Clicking any particular test will show its details (if any). Double-clicking the details of a failure should automatically bring you to that test!

Making Tests and Code Related

Eclipse doesn’t have built-in support for the sort of functionality we would really like to see here. However, we have a tip from Oscar Edvardsson, a man who is a helpful participant on our forums and github page. He discovered that we can use the MoreUnit plugin to get some added convenience.

Start installing it by opening your Eclipse Marketplace under your Help menu. Search for MoreUnit and install it. Make sure that MoreUnit Light is checked as you do so.

Once installed, open your project properties (right click the project and select Properties) and navigate to MoreUnit->User Languages.

Enable Use project specific settings and fill in the rules. For example, a common Ceedling configuration would be to have `${srcProject}/src` for the source path, `${srcProject}/test` for the test path, and `test_${srcFile}` or `Test${srcFile}` for naming the test file. Select Save.

Now, whenever you have a source or test file open, you should be able to click Ctrl+J to jump to the other. If the file doesn't exist, it will offer to make it for you. Unfortunately, we haven't figured out a way to make it handle your header file as well, but MOST of your flow is between the source and test files, so this is still incredibly helpful.

Code Completion

Eclipse *does* have good support for creating code templates. You'll be creating a list of these on your own… but here are a few good examples for you to use.
We start by opening Preferences → C/C++ → Editor → Templates.

From there, we add a template by clicking New, and then filling out the details. Let's start with a template for adding a test. We select New and then enter test for the Name. We select C\C++ for the Context and check the box to let Eclipse know we want to Automatically Insert our code. We add something helpful to the Description like "add a skeleton for testing a function." Finally, under Pattern, we get to add the actual template. Ours might look something like this:

```
void test_${function_name}_Should_${should}(void)
{
    ${line_selection}${cursor}

    ${function_name}();
}
```

There are a couple of things to note. One is that we made up some custom fields to use. These are going to allow the user to tab between them and enter data. Fields that are named the same thing will be filled out automatically when the first one is entered. The cursor field is special to Eclipse and will be where the cursor leaves off when they finish tab completing. We have chosen to let them start typing inside our brackets.

To use a template, type the start of the template name test then hit your hot key combination for autocomplete (default Ctrl+Space). You can then tab between fields and fill them out.

Let's make one more. This time, we'll make a test which we don't plan to fill out right now... instead it'll be one we plan to ignore.

```
void test_${function_name}_Should_BeCompleted(void)
{
    TEST_IGNORE_MESSAGE("${function_name} Needs Definition.");
}
```

So what happens if we want to create an entire file instead? No problem! For this, we use Eclipse's support for launching external tools. We click the External Tool Confgurations... option, then press the add button to create a new tool. We can name it something like Create Module. On the Main tab, put the usual location for ceedling, just like you did in the Run Configurations... above. The Arguments box receives a single argument that uses this built-in Eclipse feature to launch a prompt to ask for the module name:

```
module:create[${string_prompt:new_module_name:Module}]
```

Now, when you double-click the external tool (or use a hotkey to launch it), you will be prompted for a module name. Type one in (maybe something like Snow) and then press enter. The script is called and the script creates a src/Snow.c, src/Snow.h, and test/TestSnow.c, all in the appropriate folders. Feel free to add different variations for different folders and/or templates.

Final Thoughts

There are many other IDE's out there, and we recognize that chances are good that you will be using a different one. We hope that this guide will at least show you the type of features and hooks you should be searching for in order to make your integration as complete as you like.

Finally, if you are NOT able to get your IDE to work comfortably with Ceedling, it may not mean the end of your ability to use Ceedling as part of your development process. Many developers, including one of the lead mad scientists of this team, are happy tabbing between a command console window and their text editor in order to run tests. Sometimes the command line is all the power you need.

Happy Testing!