Sticky Bits

A blog looking at developing software for real-time and embedded systems

Setting up Sublime Text to build your project

Posted on April 12, 2018 by Glennan Carnie

For many years embedded development was dominated by complex integrated development environments (IDEs) that hide away all the nasty, messy details of a typical embedded software project.

Recently, with the rapidly accelerating adoption of agile techniques in embedded systems, there has been a move away from integrated development environments towards smaller, simpler, individual tools. Tools like CMake, Rake and SCons are used to manage build configurations. Container facilities like Docker provide lightweight environments for build and test. And developers are free to use their code editor of choice (and let’s face it: the “best editor” is as close to a developer’s heart as the “one-true-brace-style”)

And on. And on.

As the title suggests in this article we’re going at how to integrate the Sublime Text editor with the SCons build tool, to make developing a little more elegant and seamless.

Sublime Text

We like Sublime Text at Feabhas. It’s easy to use, easy to configure and contains some very useful features for developing C/C++ code.

Sublime Text is cross-platform, and can be downloaded from here [www.sublimetext.com]

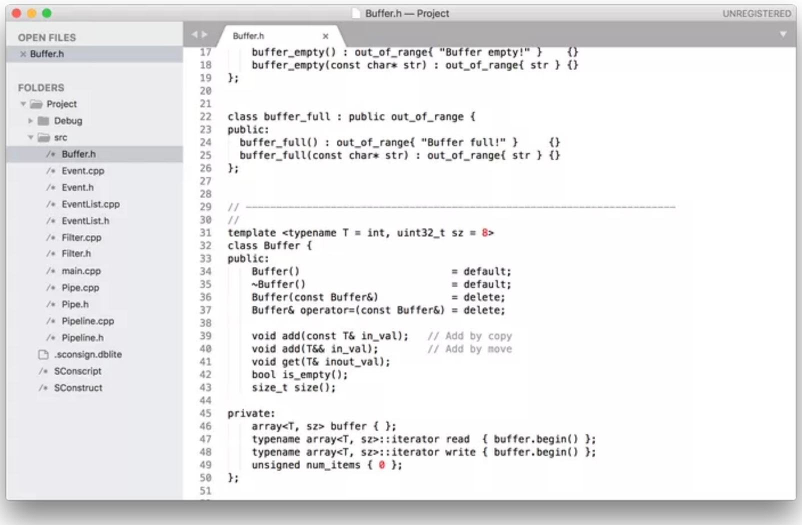
In this short article we’ll look at some basic Sublime Text configuration for incorporating builds (compilation) into the editor.

Before we start

In this example we’ve got an already-building C++ project using SCons (our software construction tool of choice at the moment). The contents of this project are unimportant, but for reference it looks like this:

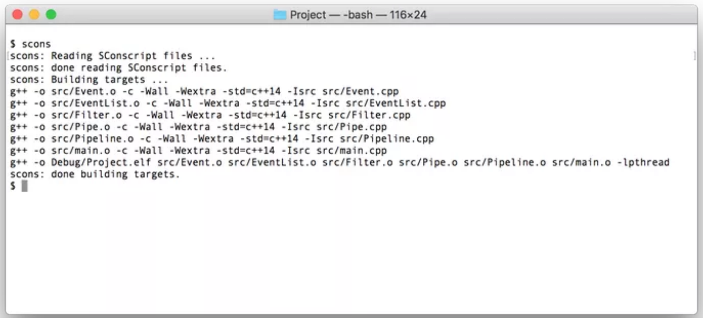
```
.
├── Debug
├── SConscript
├── SConstruct
└── src
    ├── Event.cpp
    ├── Event.h
    ├── EventList.cpp
    ├── EventList.h
    ├── Filter.cpp
    ├── Filter.h
    ├── Pipe.cpp
    ├── Pipe.h
    ├── Pipeline.cpp
    ├── Pipeline.h
    ├── Buffer.h
    └── main.cpp
```

I can launch Sublime Text as an editor, then drag my project root folder onto the tool, which makes all files in the project visible.



```
    buffer_empty() : out_of_range{ "Buffer empty!" }    {}
    buffer_empty(const char* str) : out_of_range{ str } {}
};


class buffer_full : public out_of_range {
public:
  buffer_full() : out_of_range{ "Buffer full!" }     {}
  buffer_full(const char* str) : out_of_range{ str } {}
};


// ------------------------------------------------------------------------
//
template <typename T = int, uint32_t sz = 8>
class Buffer {
public:
    Buffer()                          = default;
    ~Buffer()                         = default;
    Buffer(const Buffer&)             = delete;
    Buffer& operator=(const Buffer&) = delete;

    void add(const T& in_val);   // Add by copy
    void add(T&& in_val);        // Add by move
    void get(T& inout_val);
    bool is_empty();
    size_t size();

private:
    array<T, sz> buffer { };
    typename array<T, sz>::iterator read  { buffer.begin() };
    typename array<T, sz>::iterator write { buffer.begin() };
    unsigned num_items { 0 };
};

```

At the moment, though, I have to revert to the command line to build the project (remembering to save everything before doing so).

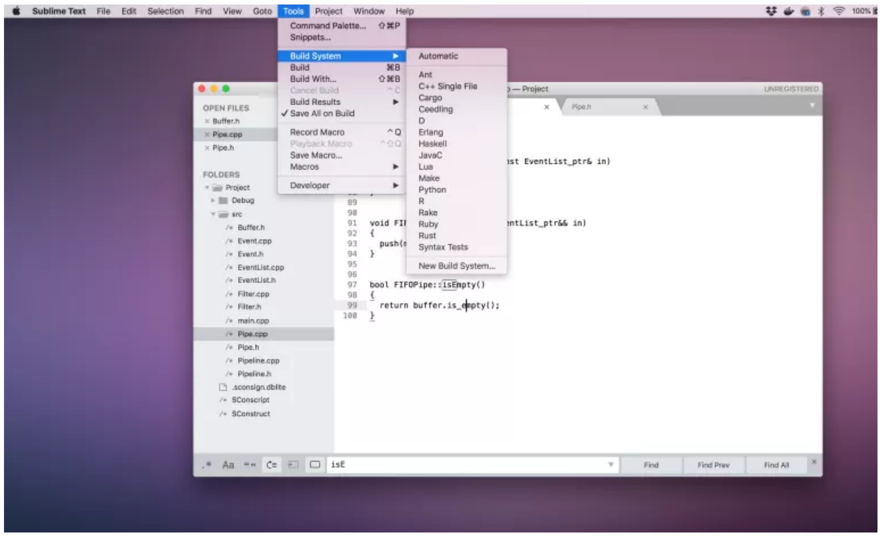


```
$ scons
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
g++ -o src/Event.o -c -Wall -Wextra -std=c++14 -Isrc src/Event.cpp
g++ -o src/EventList.o -c -Wall -Wextra -std=c++14 -Isrc src/EventList.cpp
g++ -o src/Filter.o -c -Wall -Wextra -std=c++14 -Isrc src/Filter.cpp
g++ -o src/Pipe.o -c -Wall -Wextra -std=c++14 -Isrc src/Pipe.cpp
g++ -o src/Pipeline.o -c -Wall -Wextra -std=c++14 -Isrc src/Pipeline.cpp
g++ -o src/main.o -c -Wall -Wextra -std=c++14 -Isrc src/main.cpp
g++ -o Debug/Project.elf src/Event.o src/EventList.o src/Filter.o src/Pipe.o src/Pipeline.o src/main.o -lpthread
scons: done building targets.
$ 
```

It would be more convenient if I could build from directly within Sublime Text. Luckily, Sublime Text has a feature just for this purpose.

Using Sublime Text's Build facility

On the Sublime Text **Tools** menu, select **Build System**.



```
bool FIFOPipe::isEmpty()
{
  return buffer.is_empty();
}
```

You'll notice there are a few already built in; but not one for SCons.

Let's add it.

Select the **New Build System** option

Sublime Text will create a new file called untitled.sublime-build. The contents of this file are

```
{
    "shell_cmd": "make"
}
```

The default, then, is to invoke make on your project. We update this file to invoke SCons instead.

```
{
    "shell_cmd": "scons"
}
```

Save the file as something meaningful like *scons.sublime-build*. (Note, you must retain the *.sublime-build* extension)

This file should be saved in the Sublime Text 'User Packages' folder (the default location).

For Sublime Text3, the locations are the following:

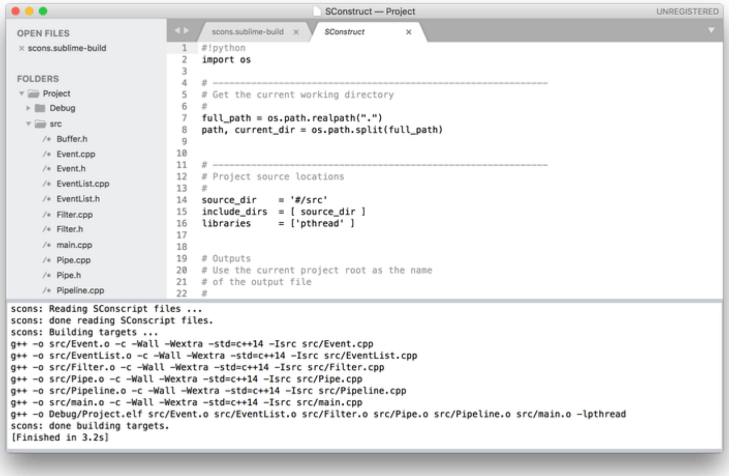
- Windows: %APPDATA%\Sublime Text 3/User
- OS X: ~/Library/Application Support/Sublime Text 3/User
- Linux: ~/.config/sublime-text-3/User

(See here for more details)

Trying our build

First, we have to make our new build script the default build. From the **Tools** menu, select **Build System**, then select your newly-created *scons* build. For convenience, it's also a good idea to select the **Save All on Build** option.

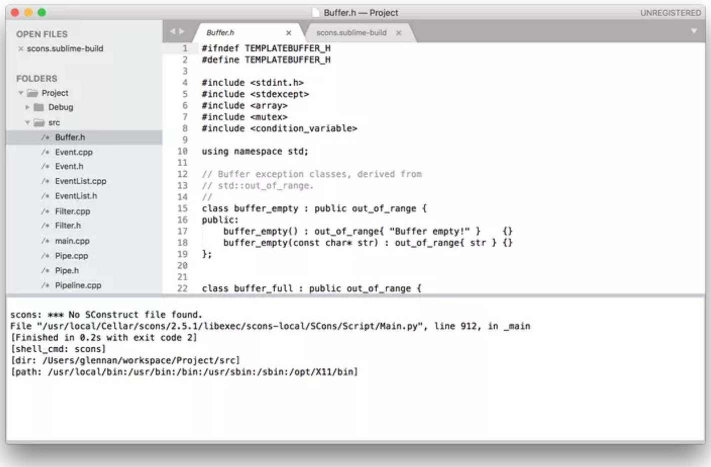
Selecting the *SConstruct* file of our project and selecting Build results in our project building as we expect. Success!



```
#!python
import os

# ------------------------------------------------------------------------
# Get the current working directory
#
full_path = os.path.realpath(".")
path, current_dir = os.path.split(full_path)


# ------------------------------------------------------------------------
# Project source locations
#
source_dir    = '#/src'
include_dirs  = [ source_dir ]
libraries     = ['pthread' ]


# Outputs
# Use the current project root as the name
# of the output file
#
```

```
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
g++ -o src/Event.o -c -Wall -Wextra -std=c++14 -Isrc src/Event.cpp
g++ -o src/EventList.o -c -Wall -Wextra -std=c++14 -Isrc src/EventList.cpp
g++ -o src/Filter.o -c -Wall -Wextra -std=c++14 -Isrc src/Filter.cpp
g++ -o src/Pipe.o -c -Wall -Wextra -std=c++14 -Isrc src/Pipe.cpp
g++ -o src/Pipeline.o -c -Wall -Wextra -std=c++14 -Isrc src/Pipeline.cpp
g++ -o src/main.o -c -Wall -Wextra -std=c++14 -Isrc src/main.cpp
g++ -o Debug/Project.elf src/Event.o src/EventList.o src/Filter.o src/Pipe.o src/Pipeline.o src/main.o -lpthread
scons: done building targets.
[Finished in 3.2s]
```

However, if we select a source file the build script fails, not able to find the *SConstruct* file.



```
#ifndef TEMPLATEBUFFER_H
#define TEMPLATEBUFFER_H

#include <stdint.h>
#include <stdexcept>
#include <array>
#include <mutex>
#include <condition_variable>

using namespace std;

// Buffer exception classes, derived from
// std::out_of_range.
//
class buffer_empty : public out_of_range {
public:
    buffer_empty() : out_of_range{ "Buffer empty!" }    {}
    buffer_empty(const char* str) : out_of_range{ str } {}
};


class buffer_full : public out_of_range {
```

```
scons: *** No SConstruct file found.
File "/usr/local/Cellar/scons/2.5.1/libexec/scons-local/SCons/Script/Main.py", line 912, in _main
[Finished in 0.2s with exit code 2]
[shell_cmd: scons]
[dir: /Users/glennan/workspace/Project/src]
[path: /usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/opt/X11/bin]
```

The problem here is Sublime Text is looking for the *SConstruct* file in whatever the current folder is.

We can resolve this by adding the path the *SConstruct* file in the bulid command file.

```
{
    "working_dir": "/Users/glennan/workspace/Project/",
    "shell_cmd": "scons"
}
```

(*Obviously, your path will be different*)

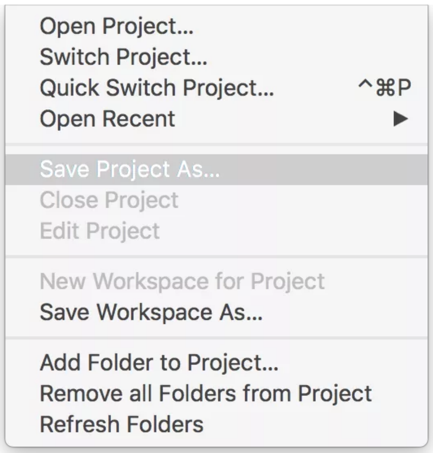
The project will now build no matter which file is currently in focus.

Adding flexibility with Sublime Text projects

The above works fine, but we've hard-coded a build path into Sublime Text for one particular project. This is clearly inflexible if we have multiple code projects to maintain.

A Sublime Text project contains editor configuration for your code. It consists of two files: A *.sublime-project* file, which contains project-specific definitions; and a *.sublime-workspace* file, which contains user specific data, such as the open files and the modifications to each. For this article we're only going to look at the *.sublime-project* file.

We can add a new project to our existing code structure by simply selecting **Project -> Save Project As…**



Select the root of your code structure (the same level as the *SConstruct* file, to make life easier)

```
.
├── Debug
├── Project.sublime-project
├── Project.sublime-workspace
├── SConscript
├── SConstruct
└── src
```

(Note: The *.sublime-workspace* file won't be visible in the project folder structure within Sublime Text; but's it's there)

Now we have a Sublime Text project we can make use of some build system variables. We can edit our build script as follows

```
{
    "working_dir": "${project_path:${folder}}",
    "shell_cmd": "scons"
}
```

This says: set the working directory (for build) to the folder containing the *.sublime-project* file; or else the first folder opened in the current project.

Provided you have a *.sublime-project* file at the same level as your *SConstruct* file this command will invoke SCons with the local *SConstruct* file. We can now have a single build tool that can build multiple projects.

Utilising the Sublime Text project file

At the moment we have added a global SCons build tool that can build any Sublime Text project with its *SConstruct* file in the code-project root directory. We can make this more flexible by adding building information into the Sublime Text project file.

Sublime Text project files are JSON, and support three top level sections:

- **folders,** for the included folders,
- **settings,** for file-setting overrides, and
- **build_systems**, for project specific build systems.

(We're only interested in the build systems settings in this article. Have a look here for more details of project configuration options)

Here's our default project file:

```
{
  "folders":
  [
    {
      "path": "."
    }
  ]
}
```

Let's add in build configurations for this particular project. The project file is edited as follows

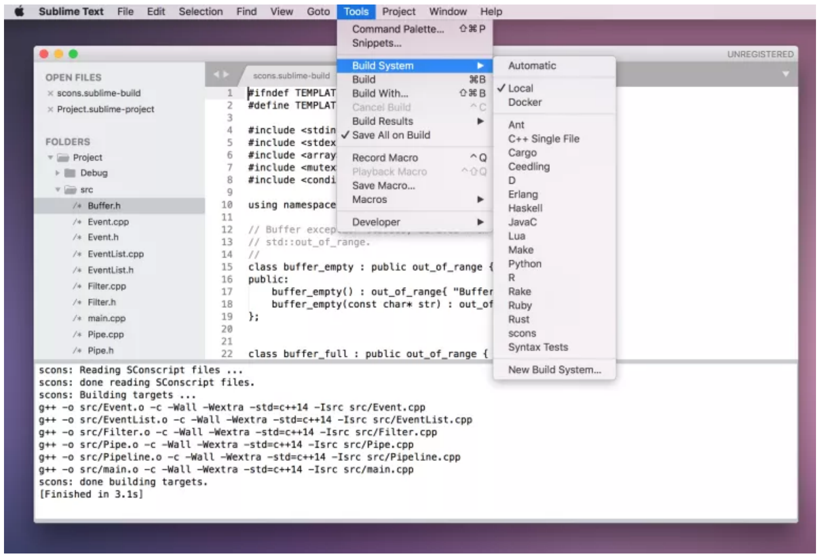
```
{
  "folders":
  [
    {
      "path": "."
    }
  ],

  "build_systems":
  [
    {
      "name": "Local",
      "working_dir": "${project_path:${folder}}",
      "shell_cmd": "scons"
    },

    {
      "name": "Docker",
      "shell_cmd": "docker run --rm -v ${project_path}:/usr/project feabhas/gcc7-scons:1.0"
    }
  ]
}
```

IMPORTANT :

Don't forget the comma between the "folders" section and the "build_systems" section, otherwise you'll get an unhelpful error (or nothing!)

In this example I've added two build configurations: "Local" uses the compiler toolchain specified in the local *SConstruct* file (as previously). The second configuration, "Docker" compiles the project code in a Docker container (for more information on Docker containers have a look at our article here)

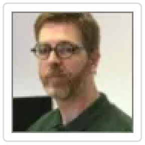
The new build configurations are now added to the **Tools** menu:



```
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
g++ -o src/Event.o -c -Wall -Wextra -std=c++14 -Isrc src/Event.cpp
g++ -o src/EventList.o -c -Wall -Wextra -std=c++14 -Isrc src/EventList.cpp
g++ -o src/Filter.o -c -Wall -Wextra -std=c++14 -Isrc src/Filter.cpp
g++ -o src/Pipe.o -c -Wall -Wextra -std=c++14 -Isrc src/Pipe.cpp
g++ -o src/Pipeline.o -c -Wall -Wextra -std=c++14 -Isrc src/Pipeline.cpp
g++ -o src/main.o -c -Wall -Wextra -std=c++14 -Isrc src/main.cpp
scons: done building targets.
[Finished in 3.1s]
```

Congratulations! You can now build your project (from its *SConstruct* file) directly from Sublime Text.

Happy building!



Glennan Carnie
Technical Consultant at Feabhas Ltd

Glennan is an embedded systems and software engineer with over 20 years experience, mostly in high-integrity systems for the defence and aerospace industry.

He specialises in C++, UML, software modelling, Systems Engineering and process development.

Like (0) Dislike (0)

This entry was posted in Agile, C/C++ Programming, General and tagged agile for embedded, build process, scons, Sublime Text, Toolchain. Bookmark the permalink.

One Response to *Setting up Sublime Text to build your project*

David Aldrich *says:*
April 26, 2018 at 9:36 am

May I recommend using Visual Studio Code with the Cpp extension? I find it works very well. It's free, cross-platform and has a very good developer community.

Like (1) Dislike (0)

Sticky Bits
Proudly powered by WordPress.