ACCU 2016

A Quick Cuppa

with scons

Managing C++ Build Complexity Using Cuppa: A SCons-based Build System — Jamie Allsop

What is cuppa?

An **Scons**-based build system focused on **C++ development**

Where can I “get” it?

pypi/**cuppa** or

github/ja11sop/**cuppa**

How do I install it?

The same as other Python packages: `$ sudo pip install cuppa`

Goals - Simple, Easy, Smart

- **minimise** the need for adding **logic** into sconscript files
- **declarative** sconscripts that are clearer and simpler than the equivalent makefile, without the need to learn a new scripting language like `make`, `cmake` or `bjam`
- provide a **clear structure for extending** the facilities offered by cuppa
- provide a **clear vocabulary** for building projects (**dependencies**, **methods**, **profiles**, **runners**, **variants**, **abi**, **architecture**, **toolchains** etc.)
- **codify best practices** so users just need to call appropriate methods knowing cuppa will do the right thing with their intent
- a framework that **allows experts to provide facilities for others to use**

Scons - One Minute Primer

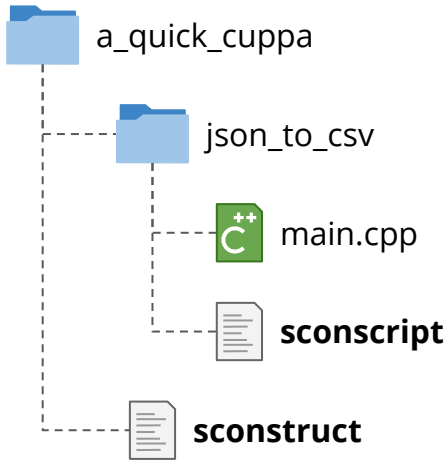
```
a_quick_cuppa$ scons
```

Executes the **sconstruct** file in the current directory which may call one or more **sconscript** files

```
a_quick_cuppa/json_to_csv$ scons -D
```

Descend from the current folder until it finds an **sconstruct** file and then executes it (which may call one or more **sconscript** files)

See Scons User Guide or Scons Man Page for more information



Tutorial

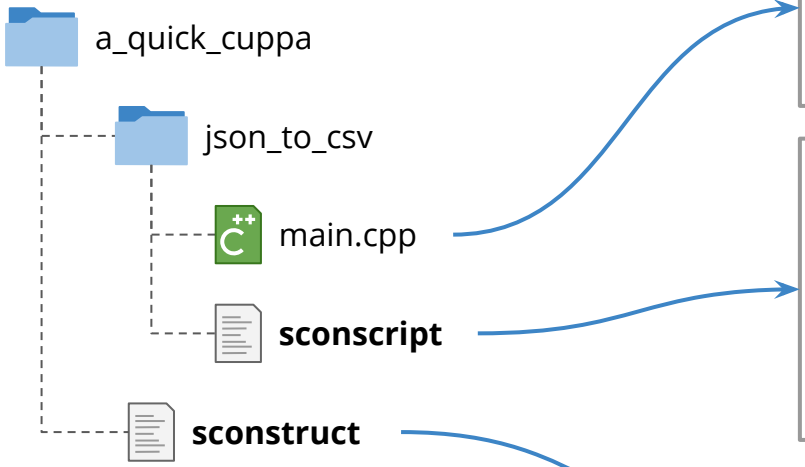
Read a StockTwits dump file of **JSON** messages and extract fields of interest to CSV (a two column CSV as it happens...)

Use C++, use both GCC and Clang, build debug and release variants

BONUS: Add version and timing information

github/ja11sop/**a_quick_cuppa**

Create the basic structure



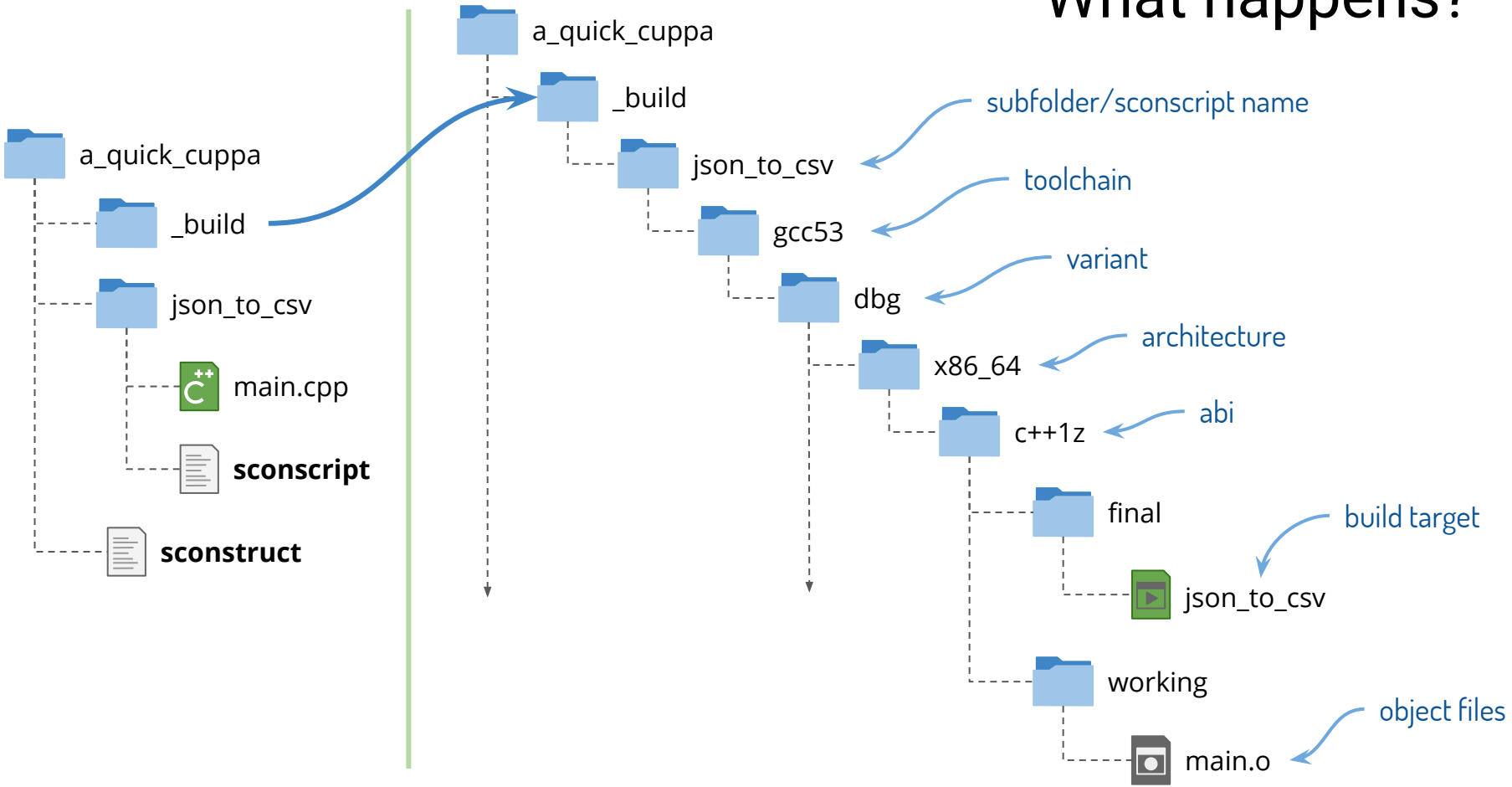

```
int main( int argc, char* argv[] )
{
    return 0;
}
```

```
Import('env')

Sources = [
    'main.cpp',
]

env.Build( 'json_to_csv', Sources )
```

```
import cuppa
cuppa.run()
```

Build with the default toolchain for the default variants: `a_quick_cuppa$ scons`

What happens?



Let's use boost::program_options for CL args

main.cpp

```cpp
// Boost Library Includes
#include <boost/program_options.hpp>
#include <boost/optional.hpp>

// Standard Library Includes
#include <iostream>

/// Helper functions validated_options() and options()

int main( int argc, char* argv[] )
{
    auto Options = validated_options( argc, argv );
    if( !Options )
    {
        std::cout << options() << std::endl;
        return 1;
    }

    return 0;
}
```

sconstruct

```
import cuppa
cuppa.run(
    default_options = {
        'boost-version': 'latest',
    },
    default_dependencies = ['boost'],
)
```

sconscript

```
Import('env')

Sources = [ 'main.cpp' ]

env.AppendUnique( STATICLIBS = [
    env.BoostStaticLibs( [
        'program_options',
    ] )
] )

env.Build( 'json_to_csv', Sources )
```

Helper functions for completeness 1

main.cpp

```
auto options()
{
    namespace po = boost::program_options;
    static auto Options
        = []()
            {
               auto Options = po::options_description( "Allowed Options" );

               Options.add_options()
                    ( "help", "Display this help message" )
                    ( "input", "The StockTwits JSON dump we wish to read" )
                    ( "output,o", po::value<std::string>(), "The name of the output file");

                return Options;
            }();
    return Options;
}
```

Helper functions for completeness 2

main.cpp

```
auto validated_options( int argc, char* argv[] )
{
    auto OptionsMap = read_options( argc, argv );

    using optional_options_t = boost::optional<decltype(OptionsMap)>;

    if( OptionsMap.count( "help" ) )
    {
        return optional_options_t();
    }
    if( !OptionsMap.count( "input" ) )
    {
        std::cout << "Option \"input\" required but not provided" << std::endl;
        return optional_options_t();
    }
    return optional_options_t( std::move(OptionsMap) );
}
```

Helper functions for completeness 3

main.cpp

```
auto positional_options()
{
    auto PositionalOptions = boost::program_options::positional_options_description();
    PositionalOptions.add( "input", -1 );
    return PositionalOptions;
}

auto read_options( int argc, char* argv[] )
{
    boost::program_options::variables_map VariablesMap;
    store
        (   boost::program_options::command_line_parser( argc, argv )
            .options( options() )
            .positional( positional_options() )
            .run(),
            VariablesMap   );

    return VariablesMap;
}
```

Extract the options

main.cpp

```cpp
int main( int argc, char* argv[] )
{
    auto Options = validated_options( argc, argv );
    if( !Options )
    {
        std::cout << options() << std::endl;
        return 1;
    }

    auto& OptionMap = *Options;

    auto InputPath = OptionMap["input"].as<std::string>();

    auto OutputPath = InputPath + ".csv";
    if( OptionMap.count( "output") )
    {
        OutputPath = OptionMap["output"].as<std::string>();
    }

    return 0;
}
```

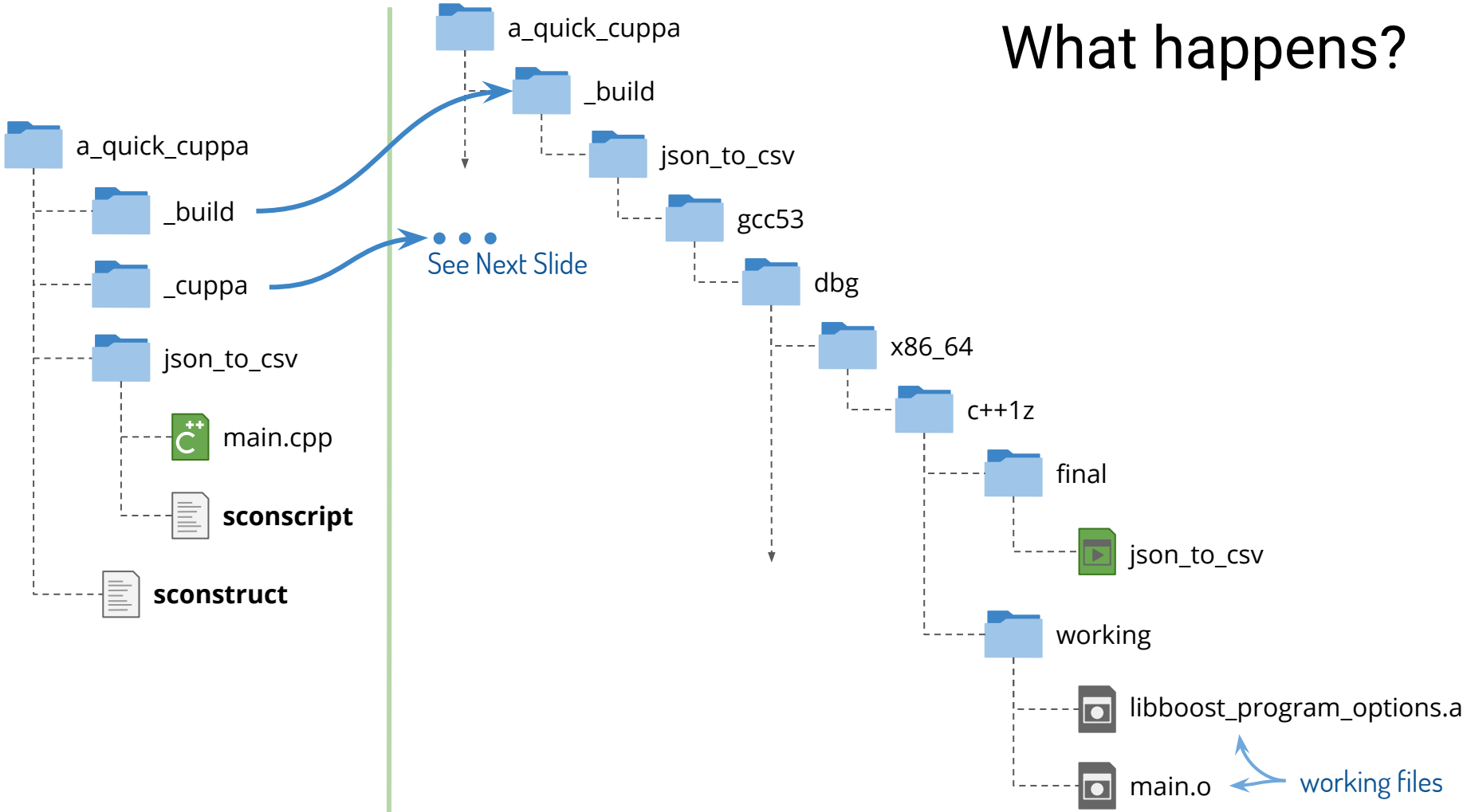
It Builds!

```
$ scons
scons: Reading SConscript files ...
cuppa: version 0.9.6
cuppa: core: [info] using sconstruct file [sconstruct]
cuppa: configure: [info] No settings to load, skipping configure
cuppa: core: [info] available toolchains are ['gcc53', 'gcc5', 'gcc', 'clang38', 'clang37',
'clang36', 'clang']
cuppa: core: [info] Using sub-sconscripts ['./json_to_csv/sconscript']
scons: done reading SConscript files.
scons: Building targets …
Progress( SconstructBegin )
Progress( Begin sconscript: [./json_to_csv/sconscript] )
Progress( Starting variant: [_build/json_to_csv/gcc53/rel/x86_64/c++1z/working] )
< RELEASE VARIANT BUILD OUTPUT HERE - SEE NEXT SLIDE! >
Progress( Finished variant: [_build/json_to_csv/gcc53/rel/x86_64/c++1z/working] )
Progress( Starting variant: [_build/json_to_csv/gcc53/dbg/x86_64/c++1z/working] )
< DEBUG VARIANT BUILD OUTPUT HERE >
Progress( Finished variant: [_build/json_to_csv/gcc53/dbg/x86_64/c++1z/working] )
Progress( End sconscript: [./json_to_csv/sconscript] )
Progress( SconstructEnd )
scons: done building targets
```

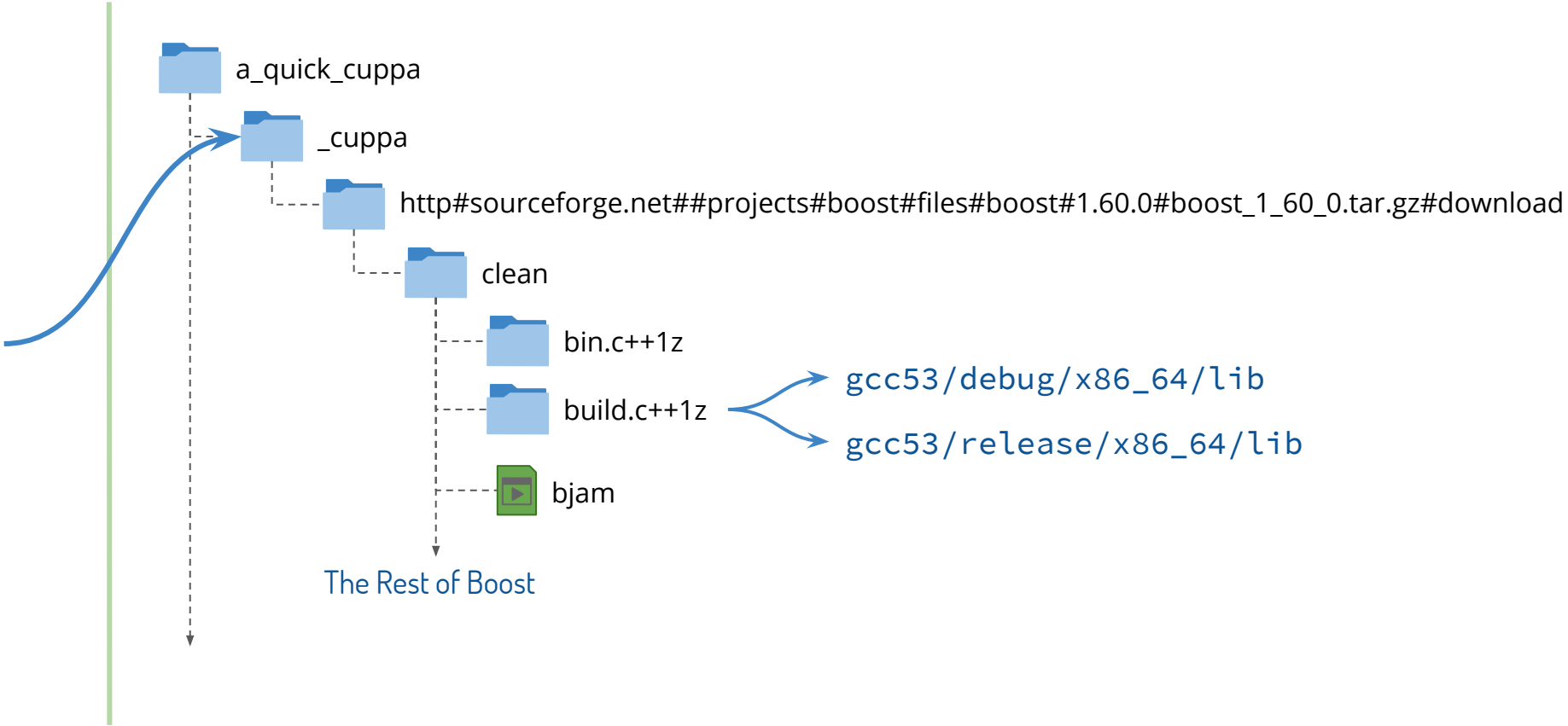
Example Release Variant Output

```
Progress( Starting variant: [_build/json_to_csv/gcc53/rel/x86_64/c++1z/working] )

WriteToolsetConfigJam(["_cuppa/http#sourceforge.net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.
gz#download/clean/gcc53._jam"], [])
cuppa: adding toolset config [using gcc :  : g++ ;] to dummy toolset config [_cuppa/http#sourceforge.
net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.gz#download/clean/gcc53._jam]

BoostLibraryAction(["_cuppa/http#sourceforge.net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.
gz#download/clean/build.c++1z/gcc53/release/x86_64/lib/libboost_program_options.a"], [])

./bjam -j 1 --with-program_options toolset=gcc variant=release    cxxflags="-std=c++1z" define="
BOOST_DATE_TIME_POSIX_TIME_STD_CONFIG" link=static --build-dir=./bin.c++1z stage
--stagedir=./build.c++1z/gcc53/release/x86_64

Install file: "_cuppa/http#sourceforge.net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.
gz#download/clean/build.c++1z/gcc53/release/x86_64/lib/libboost_program_options.a" as
"_build/json_to_csv/gcc53/rel/x86_64/working/libboost_program_options.a"

g++-5 -o _build/json_to_csv/gcc53/rel/x86_64/working/main.o -c -std=c++1z -fexceptions -Wall -g -O3 -rdynamic -
DNDEBUG -DBOOST_PARAMETER_MAX_ARITY=20 -DBOOST_DATE_TIME_POSIX_TIME
_STD_CONFIG -isystem_cuppa/http#sourceforge.net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.
gz#download/clean -I. -I_build/json_to_csv/gcc53/rel/x86_64/working -Ijson_to_csv -
I_build/json_to_csv/gcc53/rel/x86_64/working/json_to_csv -Ijson_to_csv/json_to_csv json_to_csv/main.cpp

g++-5 -o _build/json_to_csv/gcc53/rel/x86_64/final/json_to_csv -Wl,-rpath=.
_build/json_to_csv/gcc53/rel/x86_64/working/main.o -Xlinker -Bstatic
_build/json_to_csv/gcc53/rel/x86_64/working/libboost_program_options.a -Xlinker -Bdynamic -lpthread -lrt

Progress( Finished variant: [_build/json_to_csv/gcc53/rel/x86_64/c++1z/working] )
```

What happens?



_cuppa: the dependency store



Now let's read the file and parse some JSON

Let's choose an existing JSON library to help us. We'll pick RapidJSON since it appears to work, but there are many to choose from.

We can get RapidJSON from Github here:

https://github.com/miloyip/rapidjson.git

RapidJSON is a header-only library so let's update our **sconstruct** file to make this available for our use. In cuppa we can specify the location of a dependency along with the protocol to get it, including branch and revision information if supported:

git+https://github.com/miloyip/rapidjson.git

`cuppa.location_dependency(` *name*, *include=None*, *sys_include=None*)

Cuppa provides a type factory function for creating dependencies for location-based libraries. This type factory creates a dependency factory for the name specified and publishes three options:

name`-location`

Specify where the source for the dependency can be found, locally or remotely by specifying a protocol.

name`-include`

A sub-folder where header files will be found (`-I`).

name`-sys-include`

A sub-folder where "system" header files will be found (`-isystem`).

Values can be specified in the **sconstruct** or command-line, for example: `--`*name*`-include=include`

The **sconstruct** again - **no** changes to the **sconscript**

sconstruct

```
import cuppa

cuppa.run(
    dependencies = {
        'rapidjson': cuppa.location_dependency( 'rapidjson', sys_include='include' ),
    },
    default_options = {
        'rapidjson-location': 'git+https://github.com/miloyip/rapidjson.git',
        'boost-version': 'latest',
    },
    default_dependencies = [
        'rapidjson',
        'boost',
    ],
)
```

Store a dependency factory **rapidjson** as “rapidjson”

Where are the header files?

Where can we find “rapidjson” ?

By default make this available to all sconscript files

Let’s write some code to read the JSON

main.cpp

```
// RapidJSON Includes
#include "rapidjson/document.h"

/// OTHER CODE REMOVED FOR CLARITY

int main( int argc, char* argv[] )
{
    auto Options = validated_options( argc, argv );
    if( !Options )
    {
        std::cout << options() << std::endl;
        return 1;
    }

    auto& OptionMap = *Options;
    auto InputPath = OptionMap["input"].as<std::string>();
    auto OutputPath = InputPath + ".csv";
    if( OptionMap.count( "output") )
    {
        OutputPath = OptionMap["output"].as<std::string>();
    }
    return generate_csv( InputPath, OutputPath );
}
```

Open our files for reading and writing...

main.cpp

```
int generate_csv( const std::string& InputPath, const std::string& OutputPath )
{
    auto JsonFile = std::ifstream( InputPath );
    if( !JsonFile )
    {
        std::cout << "Error: Could not open input file [" << InputPath << "]" << std::endl;
        return -1;
    }

    auto CsvFile = std::ofstream( OutputPath );
    if( !CsvFile )
    {
        std::cout << "Error: Could not open output file [" << OutputPath << "]" << std::endl;
        return -1;
    }

    std::cout << "Writing CSV file to: " << OutputPath << " ..." << std::flush;
    process_json_file( JsonFile, CsvFile );
    std::cout << " Done." <<std::endl;

    return 0;
}
```

Finally do the work

main.cpp

```
template<class InputStreamT, class OutputStreamT>
void process_json_file( InputStreamT& JsonFile, OutputStreamT& CsvFile )
{
    for( std::string Line; std::getline( JsonFile, Line ); )
    {
        rapidjson::Document Json;
        Json.Parse( Line.c_str() );

        auto Body = Json["body"].GetString();
        auto Sentiment = Json["entities"].FindMember("sentiment");

        auto SentimentValue = "";
        if( !Sentiment->value.IsNull() )
        {
            SentimentValue = Sentiment->value["basic"].GetString();
        }
        CsvFile << "\"" << Body << "\",\"" << SentimentValue << "\"\n";
    }
}
```

It Builds!

```
Progress( Starting variant: [_build/json_to_csv/gcc53/rel/x86_64/c++1z/working] )
g++-5 -o _build/json_to_csv/gcc53/rel/x86_64/working/main.o -c -std=c++1z -fexceptions -Wall -g -O3 -
rdynamic -DNDEBUG -DBOOST_PARAMETER_MAX_ARITY=20 -DBOOST_DATE_TIME_POSIX_TIME_STD_CONFIG -
isystem_cuppa/http#sourceforge.net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.
gz#download/clean -I. -I_build/json_to_csv/gcc53/rel/x86_64/working -Ijson_to_csv -
I_build/json_to_csv/gcc53/rel/x86_64/working/json_to_csv -Ijson_to_csv/json_to_csv -
I_cuppa/git#https#github.com##miloyip#rapidjson.git/include json_to_csv/main.cpp
g++-5 -o _build/json_to_csv/gcc53/rel/x86_64/final/json_to_csv -Wl,-rpath=.
_build/json_to_csv/gcc53/rel/x86_64/working/main.o -Xlinker -Bstatic
_build/json_to_csv/gcc53/rel/x86_64/working/libboost_program_options.a -Xlinker -Bdynamic -lpthread -
lrt
Progress( Finished variant: [_build/json_to_csv/gcc53/rel/x86_64/c++1z/working] )
Progress( Starting variant: [_build/json_to_csv/gcc53/dbg/x86_64/c++1z/working] )
g++-5 -o _build/json_to_csv/gcc53/dbg/x86_64/working/main.o -c -std=c++1z -fexceptions -Wall -g -
rdynamic -DBOOST_PARAMETER_MAX_ARITY=20 -DBOOST_DATE_TIME_POSIX_TIME_STD_CONFIG -
isystem_cuppa/http#sourceforge.net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.
gz#download/clean -I. -I_build/json_to_csv/gcc53/dbg/x86_64/working -Ijson_to_csv -
I_build/json_to_csv/gcc53/dbg/x86_64/working/json_to_csv -Ijson_to_csv/json_to_csv -
I_cuppa/git#https#github.com##miloyip#rapidjson.git/include json_to_csv/main.cpp
g++-5 -o _build/json_to_csv/gcc53/dbg/x86_64/final/json_to_csv -Wl,-rpath=.
_build/json_to_csv/gcc53/dbg/x86_64/working/main.o -Xlinker -Bstatic
_build/json_to_csv/gcc53/dbg/x86_64/working/libboost_program_options.a -Xlinker -Bdynamic -lpthread -
lrt
Progress( Finished variant: [_build/json_to_csv/gcc53/dbg/x86_64/c++1z/working] )
```

Optional: Update the **sconscript** to copy test data

As **a_quick_cuppa** contains a sample data file under the `data` folder we can copy this to a centrally accessible location such as under a folder `json_to_csv` under the user's home directory.

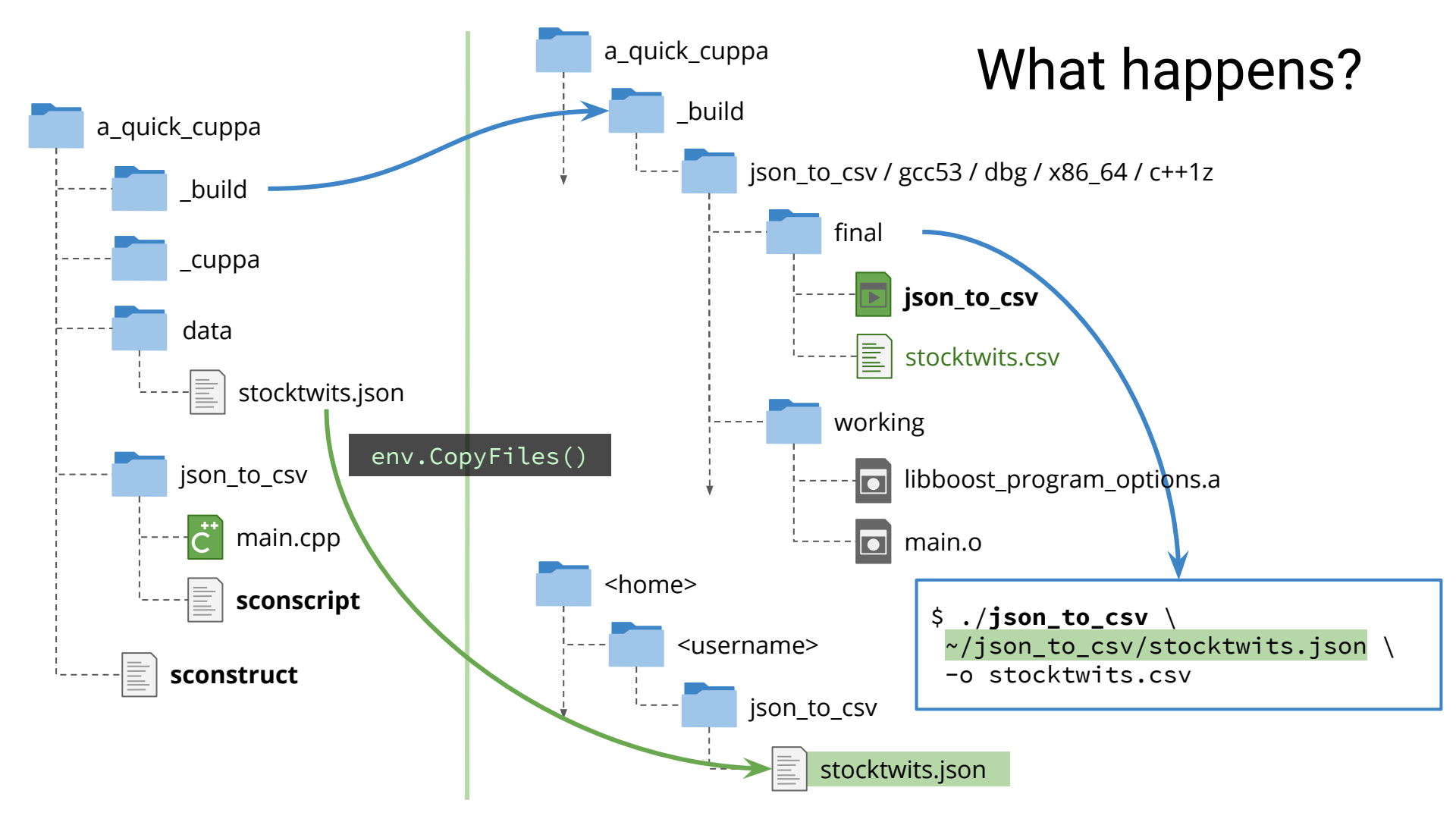
We can use the `CopyFiles(` *destination_folder*, *source_files* `)` method for this.

We use the Python **`os.path.expanduser()`** method to get the home directory.

sconscript

```
Import('env')

Version = "00.01.00"
Sources = [ 'main.cpp' ]

env.AppendUnique( STATICLIBS = [
    env.BoostStaticLibs( [
        'program_options'
    ] )
] )

import os.path
env.CopyFiles(
    os.path.join( os.path.expanduser('~'), 'json_to_csv' ),
    '#data/stocktwits.json'
)

env.Build( 'json_to_csv', Sources )
```

What happens?



It works!

```
$ ./json_to_csv ~/json_to_csv/stocktwits.json -o stocktwits.csv
Writing CSV file to: stocktwits.csv ... Done.
$
```

There's more, for example...

Build for GCC **and** Clang too:

```
$ scons --toolchains=clang38,gcc53
```

Make Clang the default:

```
$ scons --save-conf --toolchains=clang38
```

Build debug only:

```
$ scons --dbg
```

And **many more**, see the **cuppa documentation**.

Much More...

Override the C++ version used (default is most recent available):

```
$ scons --stdcpp=c++11
```

Build a variant for coverage as well as release and debug:

```
$ scons --cov --rel --dbg
```

Run all tests (and calculate coverage for `--cov` variants):

```
$ scons --cov --rel --dbg --test
```

Specify all clang toolchains using a wildcard, plus GCC 5.3, release mode only:

```
$ scons --toolchains=clang*,gcc53 --rel
```

Build in parallel, making best use of available concurrency:

```
$ scons --parallel
```

Get a list of what you can do:

```
$ scons --help
```

... and note that dependencies can publish their own options as well, prefixed by the name.

Questions?

BONUS 1
Add Version Information

Update **sconscript** (no changes to sconstruct)

`env.CreateVersion`(*version_file*, *sources*, *namespaces*, *version*, *location=env['base_path']*)

Overview: Creates a version cpp and hpp file that can be included in other files in your project while ensuring dependencies between files are correctly handled and that the cpp file is built correctly.

Effects: Creates a *version_file* and associated header file for the target file that depends on *sources*. The version file will have a class `identity` nested inside *namespaces*`::build` with an interface as described in the documentation. The *version* provided is used to populate the result of `product_version()` method and the *location* is used to specify what directories should be read to determine revision information based on the source control method used.

sconscript

```
Import('env')

Version = "00.01.00"
Sources = [ 'main.cpp' ]

env.AppendUnique( STATICLIBS = [
    env.BoostStaticLibs( [ 'program_options' ] )
] )

Objects = env.Compile( Sources )

VersionFile = env.CreateVersion(
        "version.cpp",
        Objects,
        ["json_to_csv"],
        Version
)

import os.path
env.CopyFiles(
    os.path.join( os.path.expanduser('~'), 'data' ),
    '#data/stocktwits.json'
)

env.Build( 'json_to_csv', Objects + VersionFile )
```

Update main.cpp to print out the version

main.cpp

```cpp
// Local Includes
#include "json_to_csv/version.hpp"

int main( int argc, char* argv[] )
{
    std::cout << "json_to_csv: ver." << json_to_csv::build::identity::product_version() << std::endl;
    auto Options = validated_options( argc, argv );
    if( !Options )
    {
        std::cout << options() << std::endl;
        return 1;
    }
    else if( Options->count( "version" ) )
    {
        std::cout << json_to_csv::build::identity::report() << std::endl;
        return 0;
    }
    auto& OptionMap = *Options;
    auto InputPath = OptionMap["input"].as<std::string>();
    auto OutputPath = InputPath + ".csv";
    if( OptionMap.count( "output") )
    {
        OutputPath = OptionMap["output"].as<std::string>();
    }
    return generate_csv( InputPath, OutputPath );
}
```

Updates to helper functions for completeness 1

main.cpp

```
auto options()
{
    namespace po = boost::program_options;
    static auto Options
        = []()
            {
               auto Options = po::options_description( "Allowed Options" );

               Options.add_options()
                    ( "help", "Display this help message" )
                    ( "version", "Display detailed version information" )
                    ( "input", "The StockTwits JSON dump we wish to read" )
                    ( "output,o", po::value<std::string>(), "The name of the output file");

                return Options;
            }();
    return Options;
}
```

Updates to helper functions for completeness 2

main.cpp

```
auto validated_options( int argc, char* argv[] )
{
    auto OptionsMap = read_options( argc, argv );

    using optional_options_t = boost::optional<decltype(OptionsMap)>;

    if( OptionsMap.count( "help" ) )
    {
        return optional_options_t();
    }
    else if( OptionsMap.count( "version" ) )
    {
        return optional_options_t( OptionsMap );
    }
    if( !OptionsMap.count( "input" ) )
    {
        std::cout << "Option \"input\" required but not provided" << std::endl;
        return optional_options_t();
    }
    return optional_options_t( std::move(OptionsMap) );
}
```

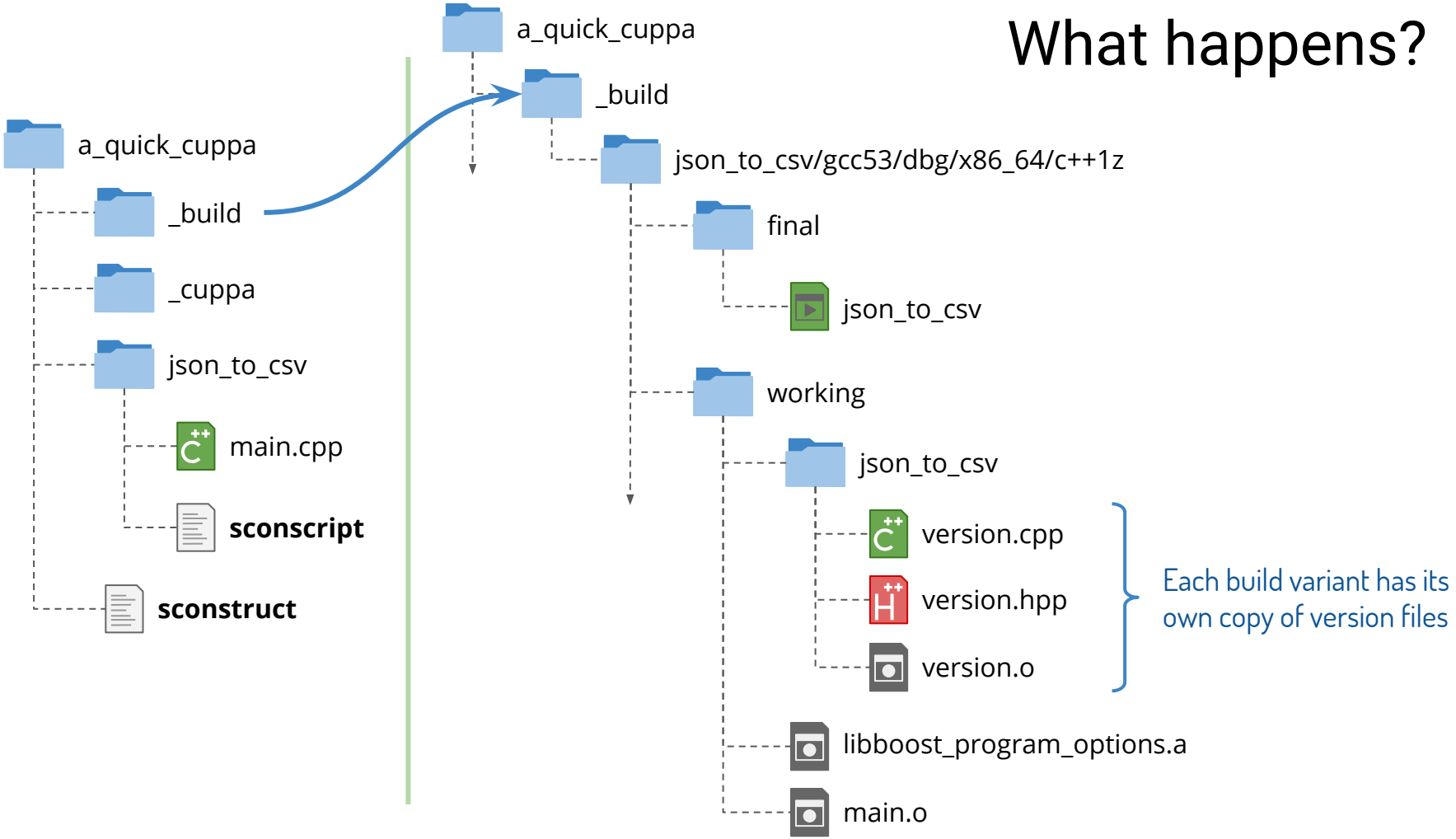
It Builds! (release output shown)

```
Progress( Starting variant: [_build/json_to_csv/gcc53/rel/x86_64/c++1z/working] )

g++-5 -o _build/json_to_csv/gcc53/rel/x86_64/c++1z/working/main.o -c -std=c++1z -fexceptions -Wall -g -O3 -rdynamic -
DNDEBUG -DBOOST_PARAMETER_MAX_ARITY=20 -DBOOST_DATE_TIME_POSIX_TIME_STD_CONFIG -isystem_cuppa/http#sourceforge.
net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.gz#download/clean -I. -
I_build/json_to_csv/gcc53/rel/x86_64/c++1z/working -Ijson_to_csv -I_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/
json_to_csv -Ijson_to_csv/json_to_csv -I_cuppa/git#https#github.com##miloyip#rapidjson.git/include json_to_csv/main.cpp

CreateVersionFileCpp(["_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/json_to_csv/version.cpp"],
["_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/main.o", "_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/
json_to_csv/version.hpp", "_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/json_to_csv/version.txt"])

g++-5 -o _build/json_to_csv/gcc53/rel/x86_64/c++1z/working/json_to_csv/version.o -c -std=c++1z -fexceptions -Wall -g -O3
-rdynamic -DNDEBUG -DBOOST_PARAMETER_MAX_ARITY=20 -DBOOST_DATE_TIME_POSIX_TIME_STD_CONFIG -
isystem_cuppa/http#sourceforge.net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.gz#download/clean -I. -
I_build/json_to_csv/gcc53/rel/x86_64/c++1z/working -Ijson_to_csv -I_build/json_to_csv/gcc53/rel/x86_64
/c++1z/working/json_to_csv -Ijson_to_csv/json_to_csv -I_cuppa/git#https#github.com##miloyip#rapidjson.git/include
_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/json_to_csv/version.cpp

g++-5 -o _build/json_to_csv/gcc53/rel/x86_64/c++1z/final/json_to_csv -Wl,-rpath=.
_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/main.o
_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/json_to_csv/version.o -Xlinker -Bstatic
_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/libboost_program_options.a -Xlinker -Bdynamic -lpthread -lrt

Progress( Finished variant: [_build/json_to_csv/gcc53/rel/x86_64/c++1z/working] )
```

It works!

```
$ ./json_to_csv --version
json_to_csv: ver.00.01.00
Product:
 |- Version    = 00.01.00
 |- Repository = https://github.com/ja11sop/a_quick_cuppa.git
 |- Branch     = master
 +- Revision   = 2b605f0
Build:
 |- Variant    = dbg
 |- Time       = 2016-04-24 19:05:46.391255
 |- User       = ja11sop
 +- Host       = penguin
Dependencies:
boost
 |- Version    = 1_60
 |- Repository = http://sourceforge.net
 |- Branch     = /projects/boost/files/boost/1.60.0/boost_1_60_0.tar.gz/download
 +- Revision   = not under version control
rapidjson
 |- Version    = master rev. fdf665d
 |- Repository = https://github.com/miloyip/rapidjson.git
 |- Branch     = master
 +- Revision   = fdf665d
```

What happens?



BONUS 2: Add Time Information

Let's use boost::timer for to measure execution time

Let's use **Boost.Timer** to measure the execution time of the process and print it out when the process completes.

Boost.Timer needs to be built as a library, like Boost. Program_options. However, unlike Boost. Program_options, **Boost.Timer also depends on** other Boost libraries, notably **Boost.Chrono** and **Boost. System**.

Cuppa automatically ensures the correct libraries are built as required.

Simply adding `'timer'` to the list of Boost libraries that you wish to link against is sufficient.

sconscript

```
Import('env')
Version = "00.01.00"
Sources = [ 'main.cpp' ]

env.AppendUnique( STATICLIBS = [
    env.BoostStaticLibs( [
        'program_options', 'timer'
    ] )
] )

Objects = env.Compile( Sources )

VersionFile = env.CreateVersion(
        "version.cpp",
        Objects,
        ["json_to_csv"],
        Version
)
import os.path
env.CopyFiles(
    os.path.join( os.path.expanduser('~'), 'data' ),
    '#data/stocktwits.json'
)
env.Build( 'json_to_csv', Objects + VersionFile )
```

Update main.cpp to print out the execution time

main.cpp

```
// Boost Includes
#include <boost/timer/timer.hpp>

int main( int argc, char* argv[] )
{
    boost::timer::auto_cpu_timer Timer;
    std::cout << "json_to_csv: ver." << json_to_csv::build::identity::product_version() << std::endl;
    auto Options = validated_options( argc, argv );
    if( !Options )
    {
        std::cout << options() << std::endl;
        return 1;
    }
    else if( Options->count( "version" ) )
    {
        std::cout << json_to_csv::build::identity::report() << std::endl;
        return 0;
    }
    auto& OptionMap = *Options;
    auto InputPath = OptionMap["input"].as<std::string>();
    auto OutputPath = InputPath + ".csv";
    if( OptionMap.count( "output") )
    {
        OutputPath = OptionMap["output"].as<std::string>();
    }
    return generate_csv( InputPath, OutputPath );
}
```

It Builds! (sample release output shown)

```
Progress( Starting variant: [_build/json_to_csv/gcc53/rel/x86_64/c++1z/working] )
BoostLibraryAction(["_cuppa/http#sourceforge.net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.
gz#download/clean/build.c++1z/gcc53/release/x86_64/lib/libboost_timer.a", "_cuppa/http#sourceforge.
net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.gz#download/clean/build.
c++1z/gcc53/release/x86_64/lib/libboost_chrono.a", "_cuppa/http#sourceforge.net##projects#boost#files#boost#1.60.0
#boost_1_60_0.tar.gz#download/clean/build.c++1z/gcc53/release/x86_64/lib/libboost_system.a", "_cuppa/http#sourceforge.
net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.gz#download/clean/build.
c++1z/gcc53/release/x86_64/lib/libboost_program_options.a"], [])

./bjam -j 1 --with-program_options --with-system --with-timer --with-chrono toolset=gcc-5 variant=release    cxxflags="-
std=c++1z" define="BOOST_DATE_TIME_POSIX_TIME_STD_CONFIG" link=static --build-dir=./bin.c++1
z stage --stagedir=./build.c++1z/gcc53/release/x86_64

Install file: "_cuppa/http#sourceforge.net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.gz#download/clean/build.
c++1z/gcc53/release/x86_64/lib/libboost_timer.a" as "_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/libboost_timer.a"
Install file: "_cuppa/http#sourceforge.net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.gz#download/clean/build.
c++1z/gcc53/release/x86_64/lib/libboost_chrono.a" as "_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/libboost_chrono.
a"
Install file: "_cuppa/http#sourceforge.net##projects#boost#files#boost#1.60.0#boost_1_60_0.tar.gz#download/clean/build.
c++1z/gcc53/release/x86_64/lib/libboost_system.a" as "_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/libboost_system.
a"

g++-5 -o _build/json_to_csv/gcc53/rel/x86_64/c++1z/final/json_to_csv -Wl,-rpath=.
_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/main.o
_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/json_to_csv/version.o -Xlinker -Bstatic
_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/libboost_timer.a
_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/libboost_chrono.a
_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/libboost_system.a
_build/json_to_csv/gcc53/rel/x86_64/c++1z/working/libboost_program_options.a -Xlinker -Bdynamic -lpthread -lrt
Progress( Finished variant: [_build/json_to_csv/gcc53/rel/x86_64/c++1z/working] )
```

Dependent libraries Chrono and System also built

It works!

```
$ ./json_to_csv ~/Documents/stocktwits_messages.json -o stocktwits.csv
json_to_csv: ver.00.01.00
Writing CSV file to: stocktwits.csv ... Done.
15.989862s wall, 8.760000s user + 0.910000s system = 9.670000s CPU (60.5%)
```

Get Involved!

pypi/**cuppa** or

github/ja11sop/**cuppa**