# The SCons build system – "...*make* as simple as possible, but not simpler"

Marek Kochańczyk[1]

[1] *Interfaculty Studies of Natural Science,*
*Jagiellonian University, ul. Reymonta 4, 30-059 Kraków, Poland*
kochanczyk@bioinformatics.org

**Abstract.** Any advanced software package that is distributed in the form of source code requires an adequately powerful build system. As there are many tools of this kind available, the choice of the most suitable one should be done owing to its management capabilities for developers and the ease of use and portability for potential users. The user's perspective is usually very narrow, hence this short report is an effort to provide comparative analysis of a new rising star – the *SCons* build system – in relation to the old and ubiquitous *GNU Autotools* from the developer's point of view. *This is by no means a guide or another fancy evidence that the author managed to follow mentally the introductory manual.*

## 1. Landscape

Out of plethora of more or less exotic build systems, usually basing on derivations of the common *make* tool, there are some that attempt to enable specification of dependencies in source code and build rules within a fresh mindset. Some more eminent Open Source solutions in this category are:

- *Apache Ant* – aimed to be general but widely considered and therefore applied as Java-specific,
- *Boost.Build* and *Bjam* – designed with generality in mind, found their niche mainly in projects written in C++,
- *Cons* and *SCons* – introducing new concepts in a very terse way, far from being complete yet of generality proved in manifold applications. Because of its growing popularity and some author's experience with this tool, *SCons* became the hero of this report.

## 2. What is SCons

The fact that during the years many clones of *make* had appeared (just to enumerate some: *tmake*, *cmake*, *nmake*, *imake*, *qmake* – all of them not backward-compatibile) leans towards impression that the paradigm was robust yet the canonical implementation was missing some essential features.

The *SCons* is a software build tool that is intended to be a replacement for *make* and related tools. Written in Python, that impacted strongly

Written 4 May 2007.

overall attitude to circumscription of building rules, it draws from more than 20 years of *make* experience and recapitulates many architectural solutions introduced in its direct Perlish successful predecessor, *Cons*.

## 3. *SCons* vs *GNU Autotools*

### *3.1 Language of configuration files. ⇒ Portability.*

In order to use *Autotools* successfully one should know:

- the m4 language – a living fossil amongst domain-specific languages,
- syntax of Makefiles – lacking if-the-else construction and any scoping,
- eventually, the Bourne shell scripting – not available on every system.

To get rid of obvious operating system dependences manifestating in explicit commands and references to the file system, *SCons* introduces configuration files as Python scripts, that are written in a declarative manner with the access to a complete general-purpose programming language just at hand.

### *3.2 Organization of configuration files. ⇒ Scalability.*

For the purpose of resolving source code dependencies, in the big picture *Autotools* seem to introduce inevitably even more dependences, just for auto-management. Configuration files in *SCons* are organized hierarchically.

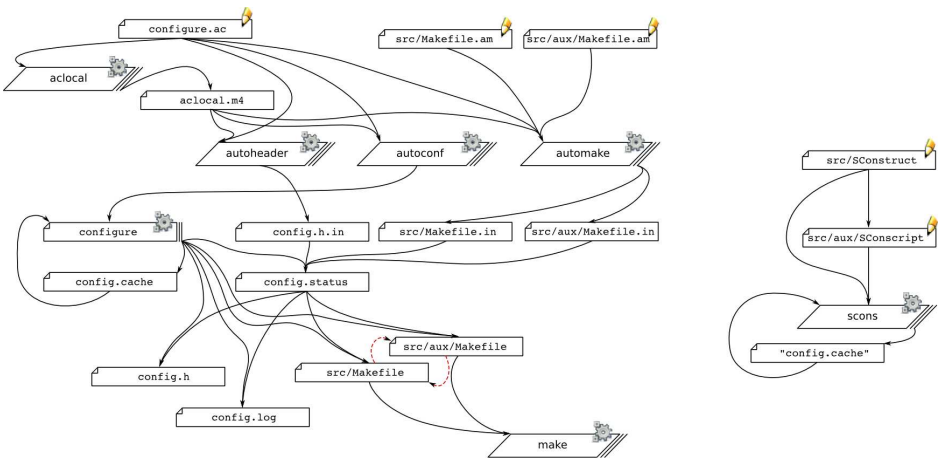


**Figure 1.** Comparison of build system workflows for a simple project with a nested source directory. Visibly, dependencies among *Autotools* cannot be even depicted in a planar graph. Prepared by the author.

Cyclic recursive dependencies that could be encountered in Makefiles (especially in generated ones) are not allowed in *SCons*.

### *3.3 Dependency extraction. ⇒ Performance.*

*Autotools* extract dependencies relying on built-in dependency analysis of GCC, so only a very limited fraction of languages is supported. Unlikely, *SCons* uses more naive and slower yet quite universal and easily-configurable regular expression scanning. Additionally, in order to rebuild targets of which source code was modified since last build, MD5 signatures are harnessed by default instead of timestamps (that increases safety in case of use of a shared file system).

*SCons* works with global view of all dependencies, so only a single target ordering and build pass is performed. Moreover, the cache can store intermediate compilation products of virtually any type, not only for C/C++ builds.

### *3.4 Built-in support and extensibility.*

While *Autotools* are C/C++-oriented, *SCons* supports equivalently most popular (imperative) programming languages and interface/stub generators. Moreover, it is able to fetch files through version control systems, is aware of automated documentation generation (including LaTeX) and basic unit testing.

Other languages or file types can be supported via user-defined Builders (class inheritance).

### *3.5 Parallel builds.*

*SCons* is capable of parallel builds and can perform those more smoothly that *make* because, when possible, does not take into account the way directories are nested.

## 4. Author's use cases

### *4.1 Generated source files.*

When some source or header files are to be generated during the build process (eg. files containing parametrization that is preferred to be hardcoded), *SCons* is able to discover appropriate dependencies on-the-fly.

### *4.2 Discarding symbols from object files.*

As the configuration file is an imperative python code, it is not straightforward as in Makefiles to arbitrarily call a system command on a file. Unfortunately, quite advanced knowledge of the *SCons* is required to extend an existing Builder class with a simple `strip` call.

*4.3 Debug levels specified in the command line.*

If the source code is meddled with macrodefs dependent on the debug level, in *SCons* it is straightforward to specify the level on the command line (eg. issuing `scons debug=3`), then parse it with a simple Python code and finally append appropriate compiler flag. In the same way, compilers with their whole building environments can be selected.

## 5. Who uses *SCons*

Unlikely the old *Cons*, its Pythonic incarnation seems to be quite vivid: recently, the SCons Foundation was awarded two student projects for Google Summer of Code 2007. Among projects using *SCons* are VMware, Blender and Quake III.

Author of this paper has been using *SCons* for 3 years now, often starting new projects with both build systems in parallel and ending up with the *SCons* because of its flexibility and expressiveness.

## 6. Conclusions

The *SCons* constuction tool is a serious modern competitor of the classic *GNU Autotools*. Despite the lack of the version 1.0, it is feature-rich. Awareness of variety of common tasks, ease of use and a gentle learning curve would probably cause an upsurge of its popularity. Probably, is worth of consideration as an aid in building and maintenance of your next project.

## Disclaimer

Although the border in this report is blurred, it should be underlined that "a make replacement" usually does not stand for "another complete build system".

## References

[1] S. Knight, *The SCons project website,*
`http://www.scons.org`.
[2] A. McCall, *Stop the autoconf insanity! Why we need a new build system,*
`http://freshmeat.net/articles/view/889`.
[3] P. Miller, *Recursive Make Considered Harmful,*
`http://aegis.sourceforge.net/auug97.pdf`.
[4] A. Neagu, *Make alternatives,*
`http://freshmeat.net/articles/view/1715/`.
[5] G. V. Vaughan, B. Elliston, T. Tromey, and I. L. Taylor, *"The Goat Book",*
`http://sourceware.org/autobook/`.