Worch Overview

Worch = **W**af + **Orch**estration:

- **software suite builder** used to build large suites of software composed of many packages from all different sources.
- **configuration manager** using a simple declarative language in order to precisely and concisely assert all build parameters.
- **workflow manager** using Waf to run interdependent tasks in parallel
- **software build features** "batteries included" for exercising many common package-level build methods
- **bootstrap aggregation** packaged using Python's `setuputils` with support for developing domain-specific extensions to easily create the build environment.
- **policy-free** leaving issues such as installation layout, target package formats, suite content, build parameters up to the end user.

Waf

Waf is a Python-based dependency-driven task scheduler (ie, `make`-replacement).
Some Waf concepts relevant to Worch:

- **task** one unit of processing, may have declared input and output files.
- **group** a logical set of *tasks* that must complete atomically and each of which are processed serially.
- **feature** a named set of *tasks*.
- **tool** a Python module providing a *feature*

Within a *group* all *tasks* are processed in a parallel to the extent allowed by dependencies and number of CPUs (or as governed by the familiar *-jN* flag.)

Worch adds/extends:

- **package** a set of *features* applied to a source package.
- **group** a Waf *group* of a set of *package*'s *tasks*.

Waf invocation

Basic running of waf:

```
$ emacs wscript                    # Waf's "Makefile"
$ waf [--prefix=/path/to/install] configure
$ waf
$ waf install
```

Worch extends Waf to accept additional command line arguments:

```
$ waf --orch-config=mysuite.cfg [...] configure
```

And to provide some additional Waf commands:

```
$ waf dot  # produce graph of of tasks dependencies
$ waf dump # dump parsed Worch configuration back out
```

Configuration Overview

```
[package root]
version = 5.34.14
features = tarball, cmake, makemake, modulesfile
environment = group:buildtools, package:cmake, package:python, package:gccxml
depends = prepare:python_install, prepare:gccxml_install
source_url = ftp://root.cern.ch/{package}/{source_archive_file}
unpacked_target = CMakeLists.txt
build_target = bin/root.exe
install_target = bin/root.exe
export_ROOTSYS = set:{install_dir}
buildenv_VERBOSE = set:1
userenv_PATH = prepend:{install_dir}/bin
```

- This snippet is from the `g4root` example, part of the Worch distribution.
- Each package gets a configuration section of a given name (here "`root`"). It need not match the actual package name.

Configuration Overview

```
[package root]
version = 5.34.14
features = tarball, cmake, makemake, modulesfile
environment = group:buildtools, package:cmake, package:python, package:gccxml
depends = prepare:python_install, prepare:gccxml_install
source_url = ftp://root.cern.ch/{package}/{source_archive_file}
unpacked_target = CMakeLists.txt
build_target = bin/root.exe
install_target = bin/root.exe
export_ROOTSYS = set:{install_dir}
buildenv_VERBOSE = set:1
userenv_PATH = prepend:{install_dir}/bin
```

Set `version` string as a variable for later reference.

Configuration Overview

```
[package root]
version = 5.34.14
features = tarball, cmake, makemake, modulesfile
environment = group:buildtools, package:cmake, package:python, package:gccxml
depends = prepare:python_install, prepare:gccxml_install
source_url = ftp://root.cern.ch/{package}/{source_archive_file}
unpacked_target = CMakeLists.txt
build_target = bin/root.exe
install_target = bin/root.exe
export_ROOTSYS = set:{install_dir}
buildenv_VERBOSE = set:1
userenv_PATH = prepend:{install_dir}/bin
```

Assert which Waf features to apply, different features expect different parameters. Here:

- **tarball** download source as a tar archive from the `sorce_url`.
- **cmake** prepare the source assuming it uses CMake.
- **makemake** run `make` and `make install`.
- **modulesfile** produce configuration for Environment Modules.

Configuration Overview

```
[package root]
version = 5.34.14
features = tarball, cmake, makemake, modulesfile
environment = group:buildtools, package:cmake, package:python, package:gccxml
depends = prepare:python_install, prepare:gccxml_install
source_url = ftp://root.cern.ch/{package}/{source_archive_file}
unpacked_target = CMakeLists.txt
build_target = bin/root.exe
install_target = bin/root.exe
export_ROOTSYS = set:{install_dir}
buildenv_VERBOSE = set:1
userenv_PATH = prepend:{install_dir}/bin
```

- Allow build-time environment provided others packages.
- Implicitly sets these packages as dependencies.
- Can depend on an individual *package* or a Waf *group* of packages.

Configuration Overview

```
[package root]
version = 5.34.14
features = tarball, cmake, makemake, modulesfile
environment = group:buildtools, package:cmake, package:python, package:gccxml
depends = prepare:python_install, prepare:gccxml_install
source_url = ftp://root.cern.ch/{package}/{source_archive_file}
unpacked_target = CMakeLists.txt
build_target = bin/root.exe
install_target = bin/root.exe
export_ROOTSYS = set:{install_dir}
buildenv_VERBOSE = set:1
userenv_PATH = prepend:{install_dir}/bin
```

Explicitly depend on certain steps in other packages to complete before running the stated step on this package:

<mystep>:<other-package>_<other-step>.

Configuration Overview

```
[package root]
version = 5.34.14
features = tarball, cmake, makemake, modulesfile
environment = group:buildtools, package:cmake, package:python, package:gccxml
depends = prepare:python_install, prepare:gccxml_install
source_url = ftp://root.cern.ch/{package}/{source_archive_file}
unpacked_target = CMakeLists.txt
build_target = bin/root.exe
install_target = bin/root.exe
export_ROOTSYS = set:{install_dir}
buildenv_VERBOSE = set:1
userenv_PATH = prepend:{install_dir}/bin
```

- Set a parameter used by a particular *feature* (in this case `tarball`).
- Example of referencing other variables defined on the package using a simple templating feature of the Worch configuration langauge.
- Can also reference parameters from other sections by prefixing their package name: `<package>_<parameter-name>`.

Configuration Overview

```
[package root]
version = 5.34.14
features = tarball, cmake, makemake, modulesfile
environment = group:buildtools, package:cmake, package:python, package:gccxml
depends = prepare:python_install, prepare:gccxml_install
source_url = ftp://root.cern.ch/{package}/{source_archive_file}
unpacked_target = CMakeLists.txt
build_target = bin/root.exe
install_target = bin/root.exe
export_ROOTSYS = set:{install_dir}
buildenv_VERBOSE = set:1
userenv_PATH = prepend:{install_dir}/bin
```

- Worch implicitly uses per-*task* "touched" output files to provide a standard way of expressing dependencies.
- Additional task output files can be declared to assure the task completed successfully.

Configuration Overview

```
[package root]
version = 5.34.14
features = tarball, cmake, makemake, modulesfile
environment = group:buildtools, package:cmake, package:python, package:gccxml
depends = prepare:python_install, prepare:gccxml_install
source_url = ftp://root.cern.ch/{package}/{source_archive_file}
unpacked_target = CMakeLists.txt
build_target = bin/root.exe
install_target = bin/root.exe
export_ROOTSYS = set:{install_dir}
buildenv_VERBOSE = set:1
userenv_PATH = prepend:{install_dir}/bin
```

Three ways to influence different environments

- buildenv during the building of this package
- export used by any dependencies through setting an `environment` parameter.
- userenv provided for any *features* implementing end-user environment management systems (eg, EM, UPS).

Other Configuration Section Types

`start`	the starting point for the interpreter, references groups, include files, Waf tools to load
`defaults`	provide global defaults
`group`	name a group, reference a list of packages, provides group-level default parameters (may override defaults)
`package`	package-level parameters (may override defaults and group)
`keytype`	special section defining the hierarchical nature of the structure of groups and packages (could be extended).

Distribution of Worch and Extensions

Preferred installation method:

```
$ virtualenv venv
$ source venv/bin/activate
$ pip install worch
# or:
$ pip install my-worch-extension
$ waf --orch-config=mysuite.cfg configure build install
```

- Exploit Python packaging ecosystem (`setuptools`, PyPI, `pip`, `virtualenv`).
- Provide a copy of `waf`
- Worch defines conventions for installation locations of **configuration file sets**, any **patches** and **Python modules** implementing Waf *tools*, *features*, and *tasks*.
- Experiments can extend Worch by providing their own Python packages.
- Trivial build environment setup.

Defining *features* - just to give the flavor

```
import orch.features
orch.features.register_defaults(  # parameter defaults
    "featurename",                # overiden by Worch
    some_param = "myinputfile",   # configuration file
)

from waflib.TaskGen import feature
@feature("featurename")
def feature_featurename(tgen):    # access to Worch config
    "Some docstring"
    tgen.step("stepname",  # feature task as a shell command
              rule="cp ${SRC} ${TGT}", # waf interprets
              source = tgen.worch.some_param + ".in",
              target = tgen.worch.some_param + ".copy")
    tgen.step("otherstep", # this one takes a function
              rule=some_function,
              source = tgen.worch.some_param + ".copy",
              target = tgen.worch.some_output)
```

Summary

- Worch provides declarative, concise and comprehensive configuration management which drive interdependent tasks for automating the building of complex software suites.
- It is general purpose and policy free.
  - Does not dictate form of build products
  - Smooths putty over the varied upstream package-level build method.
  - Simultaneous support for single-rooted or version-tree installation areas.
  - Simultaneous support for Environment Modules, UPS or add your favorite packaging.
- Comes with many batteries included.
  - CMake, Autoconf, make, tarballs, git/svn/cvs/hg repos.
- Extensible and in a way that preserves easy distribution.