# Cylc: A Workflow Engine for Cycling Systems

**Hilary J Oliver**[1], **Matthew Shin**[2], **and Oliver Sanders**[3]

**1** National Institute of Water and Atmospheric Research (NIWA), New Zealand **2** Met Office, UK **3** Met Office, UK

**DOI:** 10.21105/joss.00716

**Software**

- Review
- Repository
- Archive

**Submitted:** 01 May 2018
**Published:** 04 May 2018

## Summary

Cylc is a workflow engine for orchestrating complex *suites* of inter-dependent distributed cycling (repeating) tasks, as well as ordinary non-cycling workflows. It has been widely adopted for weather, climate, and related forecasting applications in research and production HPC environments, and it is now part of the official software infrastructure for the Unified Model atmospheric model. Cylc is written in Python and developed primarily by NIWA (NZ) and Met Office (UK). It has strong support for large production systems, but ease of use for individuals with smaller workflow automation requirements remains a key priority, and despite its core user base it is not in any way specialized to environmental forecasting.

In cycling workflows tasks repeat on sequences that may represent forecast cycles, chunks of a simulation that is too long for a single run, steps in some multi-program iterative process (e.g. for optimizing model parameters), or datasets to be processed as they are generated or received, and so forth. Cycling in Cylc is controlled by ISO 8601 date-time recurrence expressions (e.g. for environmental forecasting), or integer recurrence expressions ( e.g. for iterative processes). Dependence across cycles creates ongoing, potentially never-ending, workflows (rather than simply a succession of disconnected single workflows). Cylc is unique in its ability to manage these workflows without imposing a global cycle loop, so that one cycle does not have to complete before the next can start. Instead, Cylc's novel meta-scheduling algorithm runs tasks from many cycles at once, to the full extent allowed by individual dependencies. So, for example, on restarting after extended downtime, a workflow that processes real-time data can clear its backlog and catch up very quickly by interleaving cycles.

As a distributed system, Cylc scales sideways: each workflow is managed by its own lightweight ad-hoc server program. Existing scripts or programs can be used without modification in Cylc tasks: they are automatically wrapped in code to trap errors and report run status via authenticated HTTPS messages or polling. Cylc workflows are defined with a graph notation that efficiently expresses dependence between tasks; and task runtime properties (what to run, and where and how to run the job) are defined in an inheritance hierarchy for efficient sharing of common settings. Tasks can depend on the wall clock and arbitrary external events, as well as on the status of other tasks (*submitted*, *started*, *succeeded*, *failed*, etc.). Dependence between workflows is also supported: for coupled systems you can choose between a large suite that controls all tasks, and many smaller suites that depend on each other.

Other features of Cylc include a comprehensive command line interface and GUI; restart from workflow state snapshots with automatic determination of the fate of orphaned jobs; support for common HPC resource managers and batch systems (PBS, Slurm, etc.) as well as background jobs; edit run (manually modify a job script at the last minute); configurable automatic retries; flexible event handling; simulation and dummy modes;

support for the Jinja2 template processor in workflow definitions; internal queues to limit job submission; and conditional triggering. Major items on the development roadmap include a web GUI and port to Python 3.

The Cylc source code is available on GitHub http://cylc.github.io/cylc/ and is archived to Zenodo with the linked DOI: (Oliver, Shin, and et.al. 2018)

## References

Oliver, Hilary, Matt Shin, and Oliver Sanders et.al. 2018. "Cylc/Cylc: Cylc-7.6.1." Zenodo. https://doi.org/10.5281/zenodo.1208732.