The Journal of Open Source Software

WATTS: Workflow and template toolkit for simulation

Paul K. Romano [1¶], **Nicolas E. Stauff** [1], **Zhiee Jhia Ooi** [1], **Yinbin Miao** [1], **Amanda Lund** [1], **and Ling Zou** [1]

1 Argonne National Laboratory, USA ¶ Corresponding author

DOI: 10.21105/joss.04735

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @sskutnik
- @munkm
- @yadudoc

Submitted: 22 August 2022
Published: 10 November 2022

Summary

Modeling and simulation in many science and engineering domains often involves the execution and/or iteration of a sequence of applications, with data transfer between applications typically required. These applications often do not have a formal application programming interface (API). Instead, executing an application requires first writing a text-based input file, the format of which is typically defined in a user's manual. While text-based input files are suitable for simple one-off calculations, they can become cumbersome if a user wants to execute the applications multiple times and systematically vary input parameters, especially when a complex workflow is involved. In this case, they must resort to either manually making changes in the input file or developing their own script that modifies the input file and executes the application. Depending on the format of the input file, writing such a script can be a non-trivial and error-prone task.

watts (Workflow and Template Toolkit for Simulation) is a Python package that consists of a set of classes that can manage the execution of one or more applications. Most importantly, it provides an ability to use placeholder values in text-based input files that are filled in programmatically from Python, thereby giving users of scientific applications a means of performing parameter and sensitivity studies, optimization, and other scientific workflows using common third-party Python packages. When running multiple applications in sequence, this capability also provides a means of using the outputs of one application as the inputs (parameters) in a subsequent application. watts relies on the Jinja (Jinja Developers, 2022) templating engine for handling templated variables and expressions in input files. In a Jinja template, an identifier surrounded by a pair of {{ and }} braces denotes a variable; the variable can then be specified using the Parameters class from watts. When an application is executed via watts, it will first *render* the template using the specified parameters.

One of the challenges of managing scientific computing workflows that involve multiple applications is dealing with differing unit systems. Some applications may use the SI system of units whereas others may use some variant of the CGS system or even imperial units. When a single parameter is used by multiple applications, it begs the question of what units should be used when specifying the parameter. watts solves this problem by optionally storing physical quantities using the Quantity class from Astropy (Astropy Collaboration et al., 2018, 2013), which enables the user to specify a value along with its associated units. Each application that is linked to watts has a unit system specified so that when a templated input file for that application is rendered, any parameters stored as Quantity instances are first converted to the appropriate units. For example, if an application uses SI units and a parameter is stored in inches, it will first be converted to meters.

In addition to the templating capabilities provided by watts, there are a number of other useful capabilities for scientific simulation workflows. Each time an application is executed through watts, an isolated execution environment is used so that input and output files are not overwritten from multiple invocations. Additionally, watts keeps a local database of application input and output files along with the parameters that are associated with them for

later retrieval. Plugin classes, which are discussed further below, encapsulate the execution logic for particular applications and provide extra postprocessing capabilities for interpreting application results.

Statement of need

The motivation for the development of watts originated from research and development activities in nuclear science and engineering (NSE), which rely on a wide array of modeling and simulation applications covering areas such as reactor physics, thermal hydraulics, fuel performance, and more. Many of these applications have been developed over decades, and although some—particularly those written in C++ and Python—have a formal API by which external software can interface with, most legacy software packages in NSE typically rely on simple text-based input files and do not have an API. Thus, watts is meant to aid scientists and engineers in working with these applications, enabling integration with other off-the-shelf and open source software packages, and providing a means of data transfer between applications.

It is helpful to place watts within the context of other open source workflow systems. Many workflow systems (Lampa et al., 2019; Mölder et al., 2021; Peterson et al., 2022; Uhrin et al., 2021) provide capabilities to define workflows involving multiple applications, either through a dedicated workflow specification language or via high-level logic in a programming language. Although watts allows multiple applications to be executed within a Python script, it does not provide a mechanism for defining these workflows through a formal specification. Instead, watts is primarily intended to enable the execution of applications with templated input files that can be rendered programmatically. Other workflow systems (Babuji et al., 2019; Lampa et al., 2019; Salim et al., 2019) are focused on enabling the execution of a workflow on heterogeneous and/or distributed computing resources, often involving high-performance computing clusters. This is also outside of the scope of what watts provides.

There have been prior efforts to develop software that enables parameterization of input files. In particular, the Funz package (Richet & Chabalier, 2021) allows input files to be templated in a similar manner to watts. However, it differs in several key respects. First, Funz appears to have a broader scope in terms of how applications are executed; it allows simulations to be performed from a command-line interface, Excel, R, Python, bash, Java, and others. watts, on the other hand, solely focuses on enabling Python-based parameterized workflows. Another key difference is that Funz defines its own syntax for template parameters and expressions. In contrast, watts relies on the Jinja templating engine and its associated syntax. We believe this is advantageous for a number of reasons. Relying on Jinja significantly simplifies the implementation in watts by delegating all the logic associated with template rendering. It is also beneficial to users because learning Jinja and its associated syntax gives them a transferrable skill that is useful in any other context where Jinja is used (e.g., web development). Finally, Funz does not provide any functionality for handling unit conversions whereas watts does.

watts provides a set of "plugin" classes for specific simulation applications. These plugin classes define how the application is executed (location of executable and command-line arguments, if any), what input files are necessary, the system of units to use, and what output files are produced and collected at the end of a simulation. At present, the collection of plugin classes consists of common applications used in NSE, including MOOSE (Permann et al., 2020) and MOOSE-based applications, SAS (Fanning et al., 2016), OpenMC (Romano et al., 2015; Romano & others, 2022), MCNP (Werner et al., 2018), Serpent (Leppänen et al., 2015), RELAP5 (Fletcher & Schultz, 1992), Dakota (Adams et al., 2020), and PyARC (Stauff, 2020). However, the core capabilities of watts are not specific to the NSE field and could be applied to any science or engineering application.

At Argonne National Laboratory, watts is currently being used in a variety of research projects focused on nuclear reactor design that rely on the aforementioned set of applications. Ongoing work at Argonne also seeks to tie traditional nuclear reactor design tools with techno-economic and energy market modeling applications.

Acknowledgments

This material is based upon work supported by Laboratory Directed Research and Development (LDRD) funding from Argonne National Laboratory, provided by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-06CH1135.

References

Adams, B., Bohnhoff, W., Dalbey, K., Ebeida, M., Eddy, J., Eldred, M., Hooper, R., Hough, P., Hu, K., Jakeman, J., Khalil, M., Maupin, K., Monschke, J. A., Ridgway, E., Rushdi, A., Seidl, D., Stephens, J., & Winokur, J. (2020). *Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 6.13 user's manual.* (No. SAND2020-12495). Sandia National Laboratories. https://doi.org/10.2172/1817318

Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., Lim, P. L., Crawford, S. M., Conseil, S., Shupe, D. L., Craig, M. W., Dencheva, N., Ginsburg, A., Vand erPlas, J. T., Bradley, L. D., Pérez-Suárez, D., de Val-Borro, M., Aldcroft, T. L., Cruz, K. L., Robitaille, T. P., Tollerud, E. J., … Astropy Contributors. (2018). The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package. *The Astronomical Journal*, *156*(3), 123. https://doi.org/10.3847/1538-3881/aabc4f

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A. M., Kerzendorf, W. E., Conley, A., Crighton, N., Barbary, K., Muna, D., Ferguson, H., Grollier, F., Parikh, M. M., Nair, P. H., … Streicher, O. (2013). Astropy: A community Python package for astronomy. *Astronomy and Astrophysics*, *558*, A33. https://doi.org/10.1051/0004-6361/201322068

Babuji, Y., Woodard, A., Li, Z., Katz, D. S., Clifford, B., Kumar, R., Lacinski, L., Chard, R., Wozniak, J. M., Foster, I., Wilde, M., & Chard, K. (2019). Parsl: Pervasive parallel programming in python. *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, 25–36. https://doi.org/10.1145/3307681.3325400

Fanning, T. H., Thomas, J. W., Sumner, T., & Brunett, A. (2016). Recent developments for the SAS4A/SASSYS-1 safety analysis code. *16th International Topical Meeting on Nuclear Reactor Thermal Hydraulics*.

Fletcher, C. D., & Schultz, R. R. (1992). *RELAP5/MOD3 code manual* (NUREG/CR-5535-Vol.5). Nuclear Regulatory Commission.

Jinja Developers. (2022). Jinja. In *GitHub repository*. GitHub. https://github.com/pallets/jinja

Lampa, S., Dahlö, M., Alvarsson, J., & Spjuth, O. (2019). SciPipe: A workflow library for agile development of complex and dynamic bioinformatics pipelines. *GigaScience*, *8*(5). https://doi.org/10.1093/gigascience/giz044

Leppänen, J., Pusa, M., Viitanen, T., Valtavirta, V., & Kaltiaisenaho, T. (2015). The Serpent Monte Carlo code: Status, development, and applications in 2013. *Annals of Nuclear Energy*, *82*, 142–150. https://doi.org/10.1016/j.anucene.2014.08.024

Mölder, F., Jablonski, K., Letcher, B., Hall, M., Tomkins-Tinch, C., Sochat, V., Forster, J., Lee, S., Twardziok, S., Kanitz, A., Wilm, A., Holtgrewe, M., Rahmann, S., Nahnsen, S.,

& Köster, J. (2021). Sustainable data analysis with snakemake. *F1000Research*, *10*(33). https://doi.org/10.12688/f1000research.29032.2

Permann, C. J., Gaston, D. R., Andrš, D., Carlsen, R. W., Kong, F., Lindsay, A. D., Miller, J. M., Peterson, J. W., Slaughter, A. E., Stogner, R. H., & Martineau, R. C. (2020). MOOSE: Enabling massively parallel multiphysics simulation. *SoftwareX*, *11*, 100430. https://doi.org/10.1016/j.softx.2020.100430

Peterson, J. L., Bay, B., Koning, J., Robinson, P., Semler, J., White, J., Anirudh, R., Athey, K., Bremer, P.-T., Di Natale, F., Fox, D., Gaffney, J. A., Jacobs, S. A., Kailkhura, B., Kustowski, B., Langer, S., Spears, B., Thiagarajan, J., Van Essen, B., & Yeom, J.-S. (2022). Enabling machine learning-ready HPC ensembles with merlin. *Future Generation Computer Systems*, *131*, 255–268. https://doi.org/10.1016/j.future.2022.01.024

Richet, Y., & Chabalier, N. (2021). *Funz/funz-client* (Version 1.14) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.5761067

Romano, P. K., Horelik, N. E., Herman, B. R., Nelson, A. G., & Forget, B. (2015). OpenMC: A state-of-the-art Monte Carlo code for research and development. *Annals of Nuclear Energy*, *82*, 90–97. https://doi.org/10.1016/j.anucene.2014.07.048

Romano, P. K., & others. (2022). *openmc-dev/openmc: OpenMC* (Version 0.13.0) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.591748

Salim, M. A., Uram, T. D., Childers, J. T., Balaprakash, P., Vishwanath, V., & Papka, M. E. (2019). *Balsam: Automated scheduling and execution of dynamic, data-intensive HPC workflows*. arXiv. https://doi.org/10.48550/arxiv.1909.08704

Stauff, N. (2020). *Integration of PyARC/workbench with new fast reactor modeling and simulation capabilities* (ANL/NEAMS-20/2). Argonne National Laboratory.

Uhrin, M., Huber, S. P., Yu, J., Marzari, N., & Pizzi, G. (2021). Workflows in AiiDA: Engineering a high-throughput, event-based engine for robust and modular computational workflows. *Computational Materials Science*, *187*, 110086. https://doi.org/10.1016/j.commatsci.2020.110086

Werner, C. J., S., B. J., Solomon, C. J., Jr., Brown, F. B., Mckinney, G. W., Rising, M. E., Dixon, D. A., Martz, R. L., Hughes, H. G., III, Cox, L. J., Zukaitis, A. J., Armstrong, J. C., Forster, R. A., III, & Casswell, L. (2018). *MCNP version 6.2 release notes* (LA-UR-18-20808). Los Alamos National Laboratory. https://doi.org/10.2172/1419730