f90nml - A Python module for Fortran namelists

Marshall L. Ward[1, 2]

1 NOAA Geophysical Fluid Dynamics Laboratory, Princeton, NJ, USA **2** Australian National University, Canberra, Australia

DOI: 10.21105/joss.01474

Software

- Review
- Repository
- Archive

Submitted: 22 May 2019
Published: 14 June 2019

Summary

`f90nml` is a Python module used for importing, manipulating, and writing Fortran namelist files (ISO, 2018). The primary use case for this module is to read a namelist file via the `Parser` and save its contents into a `Namelist` data structure, which is a case-insensitive subclass of a `dict`, Python's intrinsic associative array. The `Namelist` object can be read and modified as a standard Python `dict`, and its contents can be saved as a formatted namelist file.

Fortran continues to be a dominant programming language in high-performance scientific computing (Müller et al., 2010, p. @ClimateFortranDev:2014) and namelists have been a part of the language for decades. Namelists were an early method of serializing numerical data into a human-readable format, although this has become less practical as data sizes have increased. In more recent times, namelists have been more commonly used for runtime configuration (Griffies, 2012, pp. @WRF:2019, @QUANTUMESPRESSO:2009, @UM:2019). Much of the work associated with managing and documenting the runtime parameters over a large ensemble of runs can in part be reduced to the parsing, modifying, and storing of namelists.

Python has been a dominant programming language in the sciences in recent years (Nunez-Iglesias, 2019), consistent with the overall trend across programming (Robinson, 2017), which has created a growing need for tools in Python which can manage legacy data formats. Given the importance of Fortran in both historical and modern scientific computing, the ability to accurately read and manipulate namelists offers the ability to both archive numerical results from the past and to automate the configuration of future simulations.

An example namelist, such as the one shown below:

```
&config_nml
    input = 'wind.nc'
    steps = 864
    layout = 8, 16
    visc = 1.0e-4
    use_biharmonic = .false.
/
```

would be stored as a `Namelist` which is equivalent to the following `dict`:

```
nml = {
    'config_nml': {
        'input': 'wind.nc',
        'steps': 864,
        'layout': [8, 16],
        'visc': 0.0001,
        'use_biharmonic': False
    }
}
```

The module supports all intrinsic data types, as well as user-defined types and multidimensional arrays. User-defined types are interpreted as a hierarchical tree of `Namelists`. Multidimensional arrays are saved as nested lists of lists, with the most innermost lists corresponding to the first dimensional index in Fortran. This reverses the index order in Python, but corresponds to the usual ordering in memory.

Because a value's data type is assigned by the executable at runtime and is not specified in the namelist, the data type of each value must be inferred by the `Parser`, usually based on the strictest interpretation of the value. Weak typing rules within namelists, such as the optional use of string delimiters or the multiple representations of logical values, can lead to further ambiguity. `f90nml` provides various control flags to manage these cases. A truly ambiguous value will typically be interpreted as a literal string, rather than raise an error.

Another limitation of the namelist format is the use of a arbitrary start index in a Fortran array, which may be assigned at runtime but not specified in the namelist. For this reason, arrays are assumed to begin at the lowest explicit index which is defined in the namelist, and is stored as metadata. For example, if we parse the namelist below:

```
&a_nml
   x(3:4) = 1.0, 1.1
   x(6:7) = 1.2, 1.3
/
```

then it would be saved internally as the following 0-based Python list:

```
nml = {
    'a_nml': {
        'x': [1.0, 1.1, None, 1.2, 1.3]
     }
     '_start_index': {'x': 3}
}
```

If the start index is unspecified, as in the first example, then the index is also unspecified within the `Namelist`, although the list remains 0-based within the Python environment. Additional control flags are also provided to control the start index.

`f90nml` includes a `patch` feature, which allows one to modify the values of an existing namelist while retaining its comments or existing whitespace formatting. There is some limited ability to add or remove values during patching.

`f90nml` also includes the following additional features:

- A command line tool for working in a shell environment
- Lossless conversion between `Namelist` and `dict` types
- Support for legacy Fortran namelist formats
- Conversion between JSON and YAML output
- Configuration of the output formatting rules
- Handling of repeated groups within a single namelist

Development is supported by an extensive test suite with a very high level of code coverage, ensuring compatibility of existing namelists over future releases.

Acknowledgements

Development of `f90nml` has been ongoing for several years, and was created to support research activities of the Australian Centre of Excellence in Climate System Science (ARCCSS) at the Australian National University.

This project is sustained by the feedback from its users, and continues to benefit from contributions from its userbase, for which the author is immensely grateful.

References

Allen, T., Bell, V., Bellouin, N., Bodas-Salcedo, A., Cresswell, P., Dadson, S., Dalvi, M., et al. (2019). *Unified Model documentation paper 000* (p. 91). Met Office.

Giannozzi, P., Baroni, S., Bonini, N., Calandra, M., Car, R., Cavazzoni, C., Ceresoli, D., et al. (2009). QUANTUM ESPRESSO: A modular and open-source software project for quantum simulations of materials. *Journal of Physics: Condensed Matter*, *21*(39), 395–502. doi:10.1088/0953-8984/21/39/395502

Griffies, S. M. (2012). *Elements of the Modular Ocean Model (MOM)* (p. xiii + 632). NOAA/Geophysical Fluid Dynamics Laboratory.

ISO. (2018). *ISO/IEC 1539-1:2018: Information Technology – Programming languages – Fortran – Part 1: Base language* (p. 630). Geneva, Switzerland: International Organization for Standardization.

Méndez, M., Tinetti, F. G., & Overbey, J. L. (2014). Climate models: Challenges for Fortran development tools. In *Proceedings of the 2nd international workshop on software engineering for high performance computing in computational science and engineering*, SE-HPCCSE '14 (pp. 6–12). Piscataway, NJ, USA: IEEE Press. doi:10.1109/SE-HPCCSE.2014.7

Müller, M., Waveren, M. van, Lieberman, R., Whitney, B., Saito, H., Kumaran, K., Baron, J., et al. (2010). SPEC MPI2007 – an application benchmark suite for parallel systems using MPI. *Concurrency and Computation: Practice and Experience*, *22*, 191–205. doi:10.1002/cpe.1535

Nunez-Iglesias, J. (2019, April 12). Counting programming language mentions in astronomy papers. Retrieved from https://github.com/jni/programming-languages-in-astronomy

Robinson, D. (2017, September 6). The incredible growth of Python. Retrieved from https://stackoverflow.blog/2017/09/06/incredible-growth-python/

Skamarock, W. C., Klemp, J. B., Dudhia, J., Gill, D. O., Liu, Z., Berner, J., Wang, W., et al. (2019). *A description of the Advanced Research WRF Version 4.* doi:10.5065/1dfh-6p97