ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-TUP004

A COMMUNITY EFFORT TOWARD A PARTICLE ACCELERATOR LATTICE STANDARD (PALS)*

C. Mitchell†, J-L. Vay, E. Zoni, A. Huebl, J. Qiang, LBNL, Berkeley, CA, USA
D. Sagan and M. G. Signorelli, Cornell University, Ithaca, NY, USA
C. Mayes, xLight, Inc., Palo Alto, CA, USA
D. Bruhwiler, RadiaSoft LLC, Boulder, CO, USA
D. Kallendorf, GSI Helmholtzzentrum für Schwerionenforschung, Darmstadt, Germany
Yue Hao, FRIB, Michigan State University, East Lansing, MI, USA
D. Winklehner, Physics Dept., Massachusetts Institute of Technology, MA, USA

Abstract

A major obstacle to collaboration on accelerator projects has been the sharing of lattice description files among modeling codes. To address this problem, a standardized lattice description called the Particle Accelerator Lattice Standard (PALS) is being developed. PALS development is a community-wide international effort involving accelerator physicists from multiple institutions. Along with the standard, interface packages written in commonly used languages will be developed.

The importance for developing PALS is due to the increase in scale and complexity of new machines bringing an ever greater need for global collaboration, as well as interfacing with the data-driven activities using artificial intelligence and machine learning.

The proposed Particle Accelerator Lattice Standard aims to promote: (i) portability between applications, (ii) a unified open-access description for scientific data (publishing and archiving), (iii) a unified description for post-processing, visualization and analysis. We present an introduction to the effort, an overview of the standard, a simple example, and discuss plans and future involvements from the community.

INTRODUCTION

Multi-institutional collaboration on accelerator projects that involve beam dynamics modeling, lattice design optimization, and numerical code benchmarking are challenging in part because of differing conventions among beam dynamics codes, variations in file format, and differences in implementation details. Community standardization of user-facing code inputs, outputs, and workflows provides an effective and manageable approach to mitigate these difficulties [1].

There is now an increasing need for standardization due to the scale and complexity of new machines (which typically require the integrated use of multiple simulation codes during the design stage) and the need to connect codes to AI/ML and data science ecosystems through common application programming interfaces (APIs).

The Particle Accelerator Lattice Standard (PALS) aims to define a schema for the sharing of lattice information to describe particle accelerators and storage rings [2]. While past efforts at lattice standardization [3,4] have met with partial or limited success, new factors now include the support of an international collaboration, widespread community adoption of open-source version control software such as `git`, and successful examples of other efforts toward particle accelerator community standards. These efforts include the Open Standard for Particle-Mesh Data (openPMD), a metadata standard for scalable I/O and exchange of particle and mesh-based data [5], and the Particle-In-Cell Modeling Interface (PICMI), defining conventions for the naming and structuring of input files for PIC simulations [6].

OVERVIEW OF PALS

PALS is a standard for describing the physical layout and properties of an accelerator lattice, providing a schema that defines the names of various lattice element kinds, how to organize lattice elements into lines (that beams of particles or photons can move through), and how various branches consisting of lines and elements are interconnected to form an accelerator complex (Fig. 1). The aim of a PALS based lattice is to hold all the information about an entire machine complex, from beam creation to dump lines, enabling a single lattice file to be used as the basis of start-to-end simulations, as well as design, construction and operation.

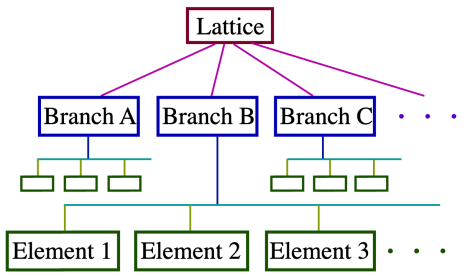

Figure 1: An expanded lattice contains a set of branches (not necessarily in an array) and each branch holds an ordered array of lattice elements.

The PALS standard does *not* describe how particles are to be tracked through a lattice. An effort has been made to separate the physical lattice description from the beam dynamics details, as far as such a separation is possible

* Contributed on behalf of the PALS collaboration.
† ChadMitchell@lbl.gov

ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-TUP004

and useful. As a metadata schema, PALS also does not define any particular syntax to implement a lattice. Options include modern file formats such as YAML, JSON, TOML, XML, etc. There are associated format-specific standards that define syntax for these, but the PALS standard itself is format agnostic.

Coordinates, Elements, and Beamlines

The lattice standard uses the three coordinate systems illustrated in Fig. 2. The floor rectangular (Cartesian) coordinate system is independent of the accelerator. The position of the accelerator itself, as well as external objects may be described using floor coordinates. The branch curvilinear coordinate system follows the bends of the accelerator, and is measured with respect to a reference curve. The branch coordinates are used to define the nominal position of each element. Finally, each element has element body coordinates which, if the element has no alignment shifts (is not "misaligned"), coincide with the branch coordinates.

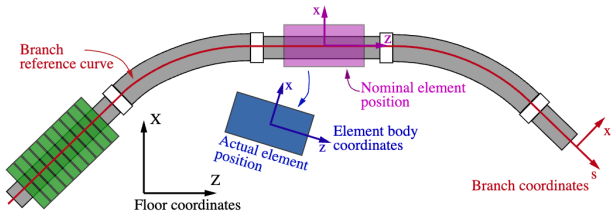

Figure 2: Coordinate systems used to define the PALS standard. Each element may be arbitrarily shifted and rotated with respect to its nominal position to support errors.

Elements are organized into kinds (Drift, SBend, Quadrupole, etc.) The parameter list for each element includes commonly-used parameters such as name, length, kind, and s-position. Additional element parameters are organized into parameter groups with standardized components (e.g., MagneticMultipoleP, containing a sequence of magnetic multipole coefficients). The PALS standard describes general architecture for the construction of complex lattice branches using nested lines, repetition, and inheritance. Finally, multiple branches can be connected using `Fork` elements.

USING PALS

In addition to the lattice standard, the PALS collaboration plans to develop software enabling programs to read, write, and validate PALS-compliant lattice files, as well as modules for lattice manipulation and analysis. The approach is to do translation in three stages as illustrated in Fig. 3. The first step in reading a lattice file is to parse the file. There are standard parser packages available for all languages contemplated (YAML, JSON, etc.) so such parsers do not need to be developed. These packages put the information from the lattice file into a standard internal form.

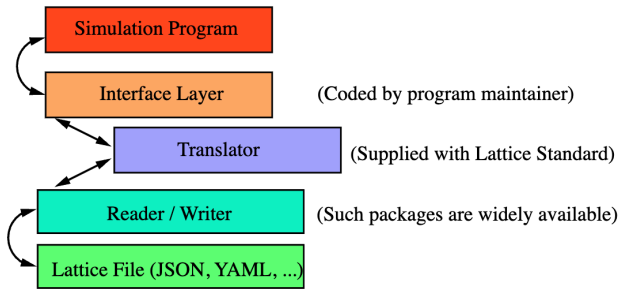

Figure 3: Schematic showing how a simulation program can interact with a PALS compliant lattice file.

The second "translator" translation step involves what is called "lattice expansion", which includes combining beamlines together to form branches and evaluating any arithmetical expressions used to define lattice and element parameter values. Included in this process can be commonly encountered calculations like calculating the positions of lattice elements in the global ("floor") coordinate system. This translation step can be bypassed if the program wants to access the original lattice file information. It is for this translator step that code will be developed by the PALS collaboration.

The final step involves converting the information block constructed in the previous step to a form that can be used by the program. Since this step is program dependent, the code for this will, of necessity, need to be developed by the maintainers of the program in question and will not be part of the code developed by the PALS collaboration.

PALS PYTHON

A Python implementation of the PALS standard is currently under development [7]. In this approach, Pydantic [9] is used to map lattice files in various modern file formats (e.g., YAML, JSON, TOML, XML, etc.) to Python objects. A schema engine like Pydantic offers several advantages, including schema validation and serialization controlled by type annotations and functional validators and serializers to meet custom needs. Not every PALS implementation needs to be as detailed as the reference implementation in Python. Once completed, this implementation can be used to convert between different file formats or to validate a file before reading it with one's favorite YAML/JSON/TOML/XML/... library in a programming language of choice.

Examples

Below is an example of a simple FODO cell expressed in PALS YAML format. Note that the details of the PALS syntax and its representation in YAML are still evolving. This example was modeled using the beam dynamics code ImpactX [11] for a matched 2 GeV electron bunch with an unnormalized emittance of 2 nm, and a visualization of the RMS beam size evolution is shown in Fig. 4. The lattice was read by coupling the code to the `pals-schema` package [10], which supports the parsing of both YAML and JSON input.

ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-TUP004


```yaml
kind: BeamLine
line:
- drift1:
    kind: Drift
    length: 0.25
- quad1:
    MagneticMultipoleP:
      Bn1: 1.0
    kind: Quadrupole
    length: 1.0
- drift2:
    kind: Drift
    length: 0.5
- quad2:
    MagneticMultipoleP:
      Bn1: -1.0
    kind: Quadrupole
    length: 1.0
- drift3:
    kind: Drift
    length: 0.25
```

Listing 1: An example PALS lattice file `fodo.pals.yaml` for a simple FODO cell.

Below is a brief code snippet showing a portion of the ImpactX input file:

```python
# load the accelerator lattice
sim.lattice.load_file("fodo.pals.yaml")
# run simulation
sim.track_particles()
```

Listing 2: Snippet of a Python input script for modeling the lattice from Listing 1 in the code ImpactX [11].

A more complex lattice (demonstrating a short FODO channel) can be found in the `examples` directory of the main PALS repository [8]. This example demonstrates additional features such as: the use of pre-defined elements, inheritance of an element with changes, reusing an element with the same name, and defining a beamline that repeats a line. Some features of this example, such as inheritance, are not yet supported in PALS Python and will be added in the near future.

STATUS AND PLANS

The document describing the PALS schema standard is well-developed [8], with the most commonly-used elements already supported. However, there is a large number of more specialized elements, e.g., plasma-based, radiofrequency quadrupoles (RFQs), cyclotrons, fixed-field alternating-gradient machines (FFAs), and more generally, elements described by (electromagnetic) field maps (e.g., importing openPMD-compliant field data on a mesh). We are actively discussing how best to include these and other as-of-yet not supported elements (such as regions of beam-matter interaction) and we are soliciting input from the community on additional types of desired accelerator lattice elements. The currently existing elements in the standard lend themselves to defining ring-type cyclotrons and many types of FFAs. We are discussing how to include single-structure elements like compact cyclotrons and RFQs beyond simply defining them as a field map or set of field maps.

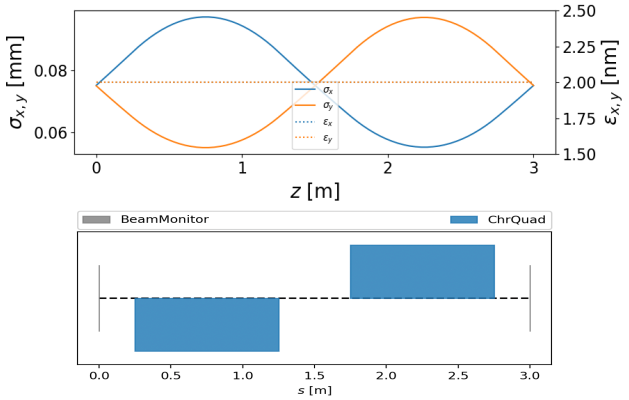

Figure 4: (Upper) Matched RMS beam envelopes obtained by particle tracking in a simple example FODO lattice. (Lower) Survey plot showing quadrupole gradient as a function of lattice position.

One of the key use cases of PALS is to provide a standardized interface for describing the accelerator lattice in virtual accelerator (VA) or digital twin (DT) applications. Simulation codes that adhere to the PALS standard could be seamlessly integrated as physics engines within the VA or DT frameworks, enabling beam dynamics modeling and prediction across various time scales. In addition, PALS will support the definition of dedicated interfaces with control systems through specialized parameter groups—such as ControlP—which include process variable (PV) names, unit conversions, setpoints, and read-back values. This will facilitate online optimization and real-time feedback/feedforward applications.

The PALS Python implementation is currently under development and is expected to increasingly cover the features described in the standard. A first package version is available through the Python Package Index (PyPI) [10].

ACKNOWLEDGEMENTS

Supported by the US National Science Foundation under award No. 2342336 and the US Department of Energy under contracts No. DE-SC0025351, DE-SC0024287, DE-AC02-05CH1123, and DOE SciDAC CAMPA project.

REFERENCES

[1] D. Sagan, M. Bertz *et al*, "Simulations of future particle accelerators: issues and mitigation", *J. Instrum*, vol. 16, no. 10, p. T10002, Oct. 2021.

ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-TUP004

[2] PALS documentation, https://pals-project.readthedocs.io

[3] H. Grote, J. Holt, N. Malitsky, F. Pilat, R. Talman, C. G. Trahern, "SXF (Standard eXchange Format): definition, syntax, examples", Rep. RHIC/AP/155, Brookhaven National Laboratory, Upton, NY, USA, 1998. doi:10.2172/1119545

[4] D. Sagan *et al*, "The Accelerator Markup Language and the Universal Accelerator Parser", in *Proc. EPAC'06*, Edinburgh, UK, Jun. 2006, paper WEPCH150, pp. 2278–2280.

[5] A. Huebl *et al*, "openPMD: A meta data standard for particle and mesh based data", 2015. doi:10.5281/zenodo.33624

[6] PICMI: Standard input format for Particle-In-Cell codes, https://picmi-standard.github.io

[7] PALS Python implementation, https://github.com/campa-consortium/pals-python

[8] PALS main repository, https://github.com/campa-consortium/pals

[9] Pydantic documentation, https://docs.pydantic.dev/latest/

[10] Python implementation of PALS in the Python Package Index, https://pypi.org/project/pals-schema/

[11] ImpactX main repository, https://github.com/BLAST-ImpactX/impactx