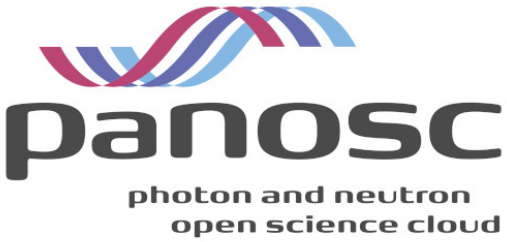

Milestone MS5.3: VINYL Software Release

Mads Bertelsen, Stella d'Ambrumenil, Juncheng E, Aljosa Hafner, Gergely Norbert Nagy, Shervin Nourbakhsh, Mousumi Upadhyay Kahaly, Carsten Fortmann-Grote

May 18, 2022

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823852.

1 Introduction

The design and implementation of a user interface (UI) requires special attention in any software product. A simulation software package must expose interfaces for the configuration of the simulation task, provision of input data, execution of the task including task and data distribution on parallel compute architectures, data retrieval and documentation. Traditionally, each simulation package implements its own specialized user interface making it hard for users to learn the usage of multiple different simulation tools. Furthermore, simulation pipelines that concatenate various simulation steps have to provide wrappers for each simulation in the pipeline. Our python3 package *libpyvinyl* addresses these issues by providing a base software layer that supports the aforementioned user interface functionalities in a generic way and ontop of which simulation codes and simulation frameworks[1] can build their specialized implementations. We strive to abstract away as much as possible the details of user interface functionalities while maintaining enough flexibility to support implementation of truly new functionality.

Simulation packages built on *libpyvinyl* will be able to skip basic tasks on parameters and software structure and go straight to implementing interesting scattering physics. Furthermore such packages will adhere to a standard scheme, facilitating collaboration across different simulation codes that share *libpyvinyl* as a foundation. The most crucial benefit is however that users will find the interfaces of packages built on *libpyvinyl* similar and thus what they learn using one package will be transferable to the others. These are the benefits of standardisation and harmonisation. The SimEx and McStasScript packages that simulate X-ray and Neutron instruments, respectively have already adopted *libpyvinyl*.

In the following, this report documents the components of *libpyvinyl*, their relation to each other as well as the installation and utilization of the package. *libpyvinyl* is registered on zenodo at https://dx.doi.org/10.5281/zenodo.6558164. At the time of writing, the latest release is version v1.1.1.

2 Libpyvinyl and its components

Since *libpyvinyl* needs to serve as a flexible foundation, it is important that the layout is general. The overall structure is briefly outlined here before the major components are described in their separate sections.

The basic unit in a simulation is referred to as a *Calculator* as it represents the execution of a simulation algorithm. Specification of the required input parameters and input data as well as the output data is the developer's responsibility. The *Data* class represents input and output data while the *Parameters* class represents simulation configuration parameters. These software entities already support a number of use cases including the concatenation of simulation steps where output data from one *Calculator* becomes the input of the next *Calculator*. Such a simulation pipeline is represented by the *Instrument* class.

2.1 Calculators

The calculator is a very general class that can take input and can provide output. All code that affect the physics of the simulation is contained within such calculators, it could for example be a calculator for a piece of optics or a sample. The input of a calculator can be data in the form of one or more data objects 2.2 and a input for a number of parameters 2.3. The output is also organised as data objects. When a new calculator is developed, the dependence on the external data and parameters is specified. Figure 1 gives a schematic representation of these base classes and some specialized classes.

[1] By 'framework', we understand a software product that supports configuration, execution and data retrieval for one or several simulation codes but that does not necessarily implement a simulation algorithm itself, rather, it drives simulations based on the user's

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823852.

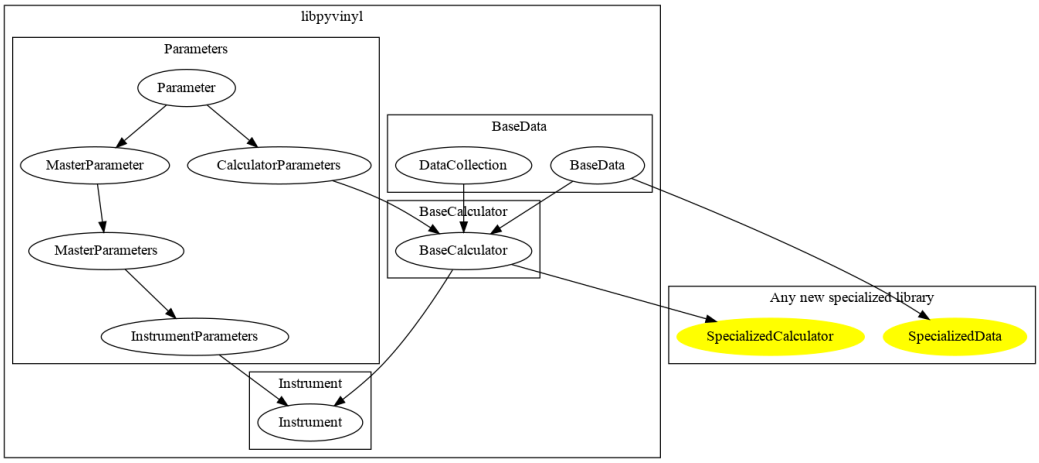

Figure 1: Schematic representation of the *libpyvinyl* base classes, derived classes and their role in describing complex simulation pipelines. Highlighted classes are implemented by specialized simulation software developers. These inherit the generic functionality and entity relations from *libpyvinyl*'s base and derived classes.

2.2 Data

The *libpyvinyl* package provides baseclasses for both individual datasets, the *Data* class (e.g. to represent a single simulated diffraction image) as well as collections of datasets (a series of diffraction images from sequential sample exposures). These objects mainly facilitate the data transfer between *Calculators*.

2.3 Parameters

The individual parameters are usually defined by a calculator class using the *libpyvinyl* base parameter class. This class includes the ability to contain arrays, units and permissible value ranges. The *Parameters* class is a major part of the safety net that prevents a user from faulty configuration of the interfaced simulation code which would cause runtime errors or lead to wrong or misleading results.

2.4 Instruments

Instruments are conceptually a series of *Calculators* but can also contain master parameters that configure the simulation globally. One could imagine a source *Calculator*, some optics *Calculators*, a sample *Calculator* and then a detector *Calculator* along with a *Parameter* describing the simulated energy. As a *MasterParameter*, information about the simulated energy can be provided to multiple *Calculators* that would need that information to avoid the user having to specify redundant information. The chain of *Calculators* can then be executed by calling the *Instrument*'s member functions.

3 Software availability and Installation

The source code for *libpyvinyl* is hosted on github. Tagged releases are registered the python package index *pypi* as documented in Sec. 3.2.

input. An example would be the Oasys project, a framework for x-ray optics simulations.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823852.

Each commit to the code base triggers a build–test–deploy cycle. Besides building the library components, also the documentation is compiled and published. Table 1 provides the URLs to the aforementioned resources.

Software repository	https://github.com/PaNOSC-ViNYL/libpyvinyl
Releases registry	https://github.com/PaNOSC-ViNYL/libpyvinyl/releases
Pypi	https://pypi.org/libpyvinyl
CI/CD status	https://github.com/PaNOSC-ViNYL/libpyvinyl/actions
Documentation	https://libpyvinyl.readthedocs.io/

Table 1: Resource URLs for *libpyvinyl*

3.1 Dependencies and requirements

libpyvinyl is continuously tested on python versions 3.6 until 3.10. Required python packages are listed in the repository's *requirements* files and will be installed automatically if the following instructions are applied.

3.2 Installation from the python package index pypi

Installation via *pypi* is the most straightforward method. On a Unix based system, the shell command

```
$> pip install libpyvinyl
```

will download, configure, and install the library into the users current python runtime environment.

3.3 Installation from source

Installing the library directly from the sources allows to install the latest, potentially unstable, release. This may be an option for developers seeking to contribute to *libpyvinyl* or to test new features that are not available in the released versions yet.

To get the sources from github and then install into the current runtime environment:

```
$> git clone https://github.com/PaNOSC-ViNYL/libpyvinyl
$> pip install [-e] .
```

Activating the option `-e` will link the installed code to the downloaded sources such that changes in the source code would immediately be reflected in the runtime environment, i.e. no re-installation is required.

4 Instructions for developers

Extensive usage instructions for developers of downstream simulation codes are provided in the online documentation. Here we only summarize the main points:

Identify needed components Not all components provided by *libpyvinyl* may be needed to implement the targeted functionality. E.g. a standalone simulation code does not necessarily have to implement the *Instrument* class functionalities.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823852.

Inherit base classes In their simulation code, developers would then define their own classes to represent data, code execution, parameters etc. Fig. 1 hints at which classes to inherit from for the various components.

Implement specialized functionality In many cases, only a very limited number of methods need to be implemented and members declared. Consult the source code of each parent class as well as the examples provided in the documentation.

Test We strongly recommend to add unit and regression tests for your derived classes. Examples for how to setup a test class are provided in the `tests/` directory of the source code repository. In particular, `tests/integration/plusminus` contains tests for a derived *Calculator* which is also documented as an example in the documentation.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823852.