BornAgain, software for GISAS and reflectometry: releases 1.17 to 20

Ammar Nejati[1], *Mikhail* Svechnikov[1], and *Joachim* Wuttke[1,*]

[1]Forschungszentrum Jülich GmbH, Jülich Centre for Neutron Science at MLZ, Lichtenbergstraße 1, 85748 Garching, Germany

Abstract. BornAgain is a free and open source cross-platform software for simulating and fitting grazing-incidence small-angle scattering, off-specular scattering, and reflectometry. An authorative description as per release 1.16 of 2019 has been published in J. Appl. Cryst. 53, 262–276 (2020). This report explains the incremental changes from releases 1.17 to 20.

1 Introduction

The research software BornAgain was originally conceived for simulating and fitting neutron and x-ray grazing-incidence small-angle scattering (GISAS). It can also be used with off-specular scattering, reflectometry, and regular small-angle scattering. BornAgain is a cross-platform software with active support for Linux, Windows, and MacOS. BornAgain is a free and open-source project under the GNU Public License. Download links and online documentation can be reached from the project homepage [1]. Source repository and issue tracker are at [2].

The canonical literature reference for BornAgain is the comprehensive paper in J. Appl. Cryst. [3], based on software release 1.16 from 2019. The present report documents the incremental changes from releases 1.17 to 20.

2 Release history, numbering scheme

Releases 1.17 to 1.19 were published in 2020 and 2021 [4]. We then spent two full years on internal consolidation before the next release was published in March 2023. At this point the numbering scheme was changed: The major version number "1" was dropped, and the old minor promoted to major so that 1.19 was followed by version 20.

The change in the numbering scheme expresses our conviction that BornAgain has become too big for any comprehensive re-enginering or partial re-writing that would justify stepping the old major from 1 to 2. Instead, all future evolution of BornAgain will be through incremental changes in agile development cycles that should not take longer than a couple of months.

The full version names were actually ternary (1.19.0), and are now binary (20.0). In either case, the trailing ".0" expresses the patch level that is incremented only in hotfix releases, such as 20.1 from May 2023.

3 Source repository

The source repository [2], including the issue tracker, has been relocated from GitHub to Jugit, our institution's instance of the open-source software GitLab. External contributors can log in through a federated service [5], then self-register. Contributors from institutions that are not participating in the federation can login using their GitHub, Google, or ORCID account [6].

4 Binaries for different Python versions

We provide all necessary configuration scripts and explanations for building BornAgain from source. Most users, however, prefer binary installers, which we provide at [7] for four platforms (Linux and Windows on x64 architecture, Mac on x64 and arm). For Linux, as an alternative to the installer shell script, there exist externally maintained Debian packages [8].

For Python scripting to work, the minor version of the user's Python engine must be the same as was used in building the BornAgain binaries. In the past, version conflicts often prevented users from getting started with BornAgain scripting. Therefore, starting with release 20, we now provide binary installers for several Python minor versions (currently 3.8 to 3.10), and this for each of the four supported target platforms.

5 Python-only packages (wheels for pip)

For those who but intend to use BornAgain through the Python API only, release 20 introduced yet another, even simpler installation method: We packaged a non-GUI variant of BornAgain as a Python *wheel*. A wheel [9] is a zipped archive of Python scripts and object libraries. In our case these libraries comprise the BornAgain modules compiled from the C++ sources, including Swig-generated wrapper code, and all external library BornAgain depends on.

A wheel has a specially formatted file name that encodes software name, software version, Python language

[*]e-mail: j.wuttke@fz-juelich.de

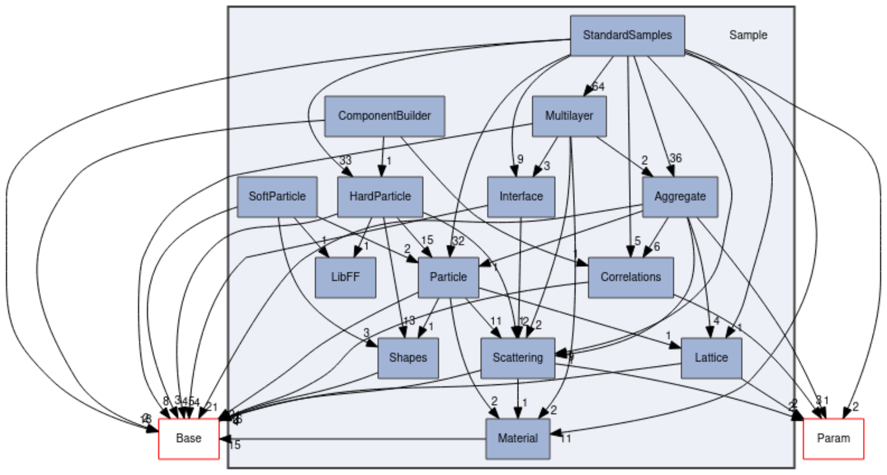


Figure 1. Include relations between code subdirectories concerned with the sample model. Graphical representation generated by Doxygen. All arrows go downwards: there are no cyclic dependences.

version, Python application binary interface version, target operating system and target processor architecture. So we prepared 12 wheels, for the four platforms and the three Python minor versions mentioned above.

We uploaded them to the standard Python package repository, for historical reasons misnomed “index”: the Python Package Index (PyPI). Our wheels can be either downloaded through the “Download files” link at [10], or they can be automatically retrieved and installed using *pip* commands.

6 Internal consolidation

In the years leading to release 1.16, BornAgain has rapidly grown in scope, but also in complexity. The clarity and elegance of the initial architecture was somewhat lost in the course of unforeseen extensions. New developers found it increasingly difficult to implement new features, and most developer time was spent on analyzing how the program was working. This problem was addressed in releases 1.17 to 20 through refactoring aimed at simplifying and unifying code and data structures.

We reorganized the directory tree that holds over 1500 source files, and resolved cyclic dependence between directories [11, Sect. 5.4.1] so that include relations now form a directed graph (Fig. 1). We renamed classes, variables and functions that had evolved away from their original meaning. To facilitate code analysis, functions of same name in unrelated classes were given distinct names. In a majority of cases, C++ templates could be replaced by simpler constructs.

We removed unused code and reverted premature generalizations. To reduce the depth of call chains, numerous short functions were inlined or merged. Classes only meant to hold algorithms were dissolved, and the algorithms moved into free functions or under other classes. New abstractions helped to merge duplicate code.

Three-dimensional vector classes were moved from BornAgain into a separate library, *libheinz* [12], which in the future shall also be used by other projects at Heinz Maier-Leibnitz Zentrum. Form-factor computations [13] were also moved to a separate library, *libformfactor* [14], so that time-consuming numerical tests could be taken out of the BornAgain test suite. Regarding test coverage [11, Sect. 5.8], we removed trivial unit tests, and added functional tests, especially in form of Python scripts which have dual use as part of the example script collection.

As a result, core code could be reduced by 18 % from 38 kLoC (kilo lines of code) in release 1.16 to 31 kLoC (including 2 kLoC moved to libheinz and libformfactor) in release 20, and the GUI code by 13 %, from 60 kLoC to 52 kLoC, all while improving and extending functionality.

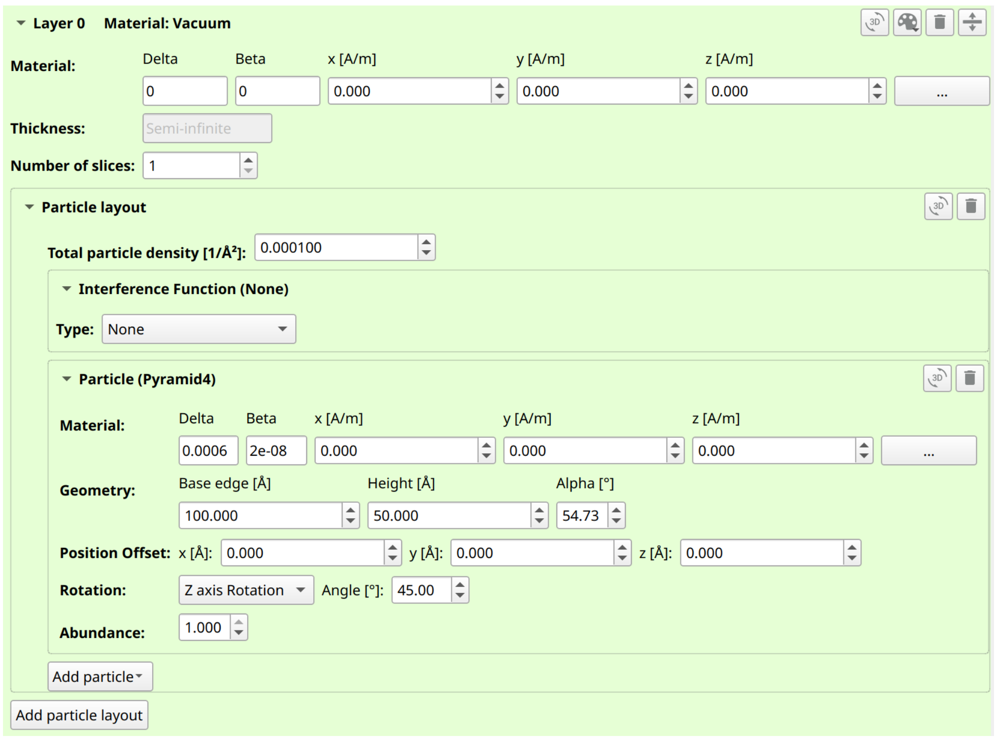
7 API changes

The Python Application Programming Interface (API) of BornAgain mostly consists of class constructor and class member function calls. Its ergonomy depends on the pertinence, expressivity and uniformity of function names and argument lists. If the principle of least surprise [15, 16] is well respected the user can work by analogy and develop an intuition how to use classes and functions. If the API is inconsistent the user needs to look up each function signature in the reference.

To edge out inconsistencies and make the API as uniform and unsurprising as possible, each release is currently breaking some user scripts. We are aware that it is annoying, but under the premise that BornAgain will have a lifespan of decades it is the right choice to sacrifice backward compatibility now for a more consistent API and a more maintainable code base in the future. All API changes are listed in the change log [4].

8 GUI, new sample editor

Release 20 brought two important changes to the graphical user interface (GUI): Full serialization, and a new sample editor.


Layer 0 Material: Vacuum

Material: Delta 0, Beta 0, x [A/m] 0.000, y [A/m] 0.000, z [A/m] 0.000

Thickness: Semi-infinite

Number of slices: 1

Particle layout

Total particle density [1/Å²]: 0.000100

Interference Function (None)

Type: None

Particle (Pyramid4)

Material: Delta 0.0006, Beta 2e-08, x [A/m] 0.000, y [A/m] 0.000, z [A/m] 0.000

Geometry: Base edge [Å] 100.000, Height [Å] 50.000, Alpha [°] 54.73

Position Offset: x [Å]: 0.000, y [Å]: 0.000, z [Å]: 0.000

Rotation: Z axis Rotation, Angle [°]: 45.00

Abundance: 1.000

Figure 2. Screenshot of the new sample editor GUI widget, featuring a random assembly of uncorrelated pyramids in an otherwise empty surface layer.

Through serialization, the full state of a GUI session, including sample and instrument models, can be saved to and reloaded from a *project file*. Uses include resuming work after a break, reverting unsuccessful modifications of a model, sharing with collaborators, and creating fully reproducible bug reports.

The next couple of releases will likely break project file compatibility for certain model components, for the same reasons as discussed above for the Python API. In the longer run, however, we intend to ensure backward compatibility. For this purpose, each serialized feature carries a version number.

Up to release 1.19, the GUI had a graphical sample editor inspired inspired by the LabVIEW way of graphical programming. Users assembled models from components by drag and drop. This allowed to visualize the stack of layers and highlighted the internal hierarchy of layers, layouts, particles, interference functions and their modifications. Acclaimed by many novice users, this editor was aesthetically pleasing and had an almost haptic appeal. However, the block diagrams did not show model parameters, which could only be modified in a separate widget. Experienced users therefore requested a more compact sample representation.

In response to that, release 20 has a completely new sample editor, based on expandable tables. Selectable lists, parameters, their labels and editable fields are organized as grids inside nested collapsible boxes. This allows both handling layers with arbitrary internal complexity and the direct presentation of exhaustive parameter set to the user.

9 Documentation

The online documentation, accessible from the project home page [1], is now versioned: starting with 1.19, the documentation for each release will indefinitely remain readable.

Up to 1.19, the main part of the documentation was a very long tutorial on Python scripting. Usage was mostly taught by code examples, followed by explanations. In place of a reference, a documentation of the C++ application programming interface (API) was auto-generated by Doxygen.

The tutorial turned out to be cumbersome to maintain. Readability suffered from numerous repetitions. The C++ API reference was found inadequate by Python users. For these reasons, the Python scripting documentation was reorganized in release 20. It was split into three parts: a short tutorial, a collection of examples, and a human-written reference. The new, concise style is inspired by best-practice examples like the classical Unix man pages or the Qt online documentation, and is guided by the assumption of an intelligent, active reader who is able to follow links and draw obvious consequences [11, 16].

The conversion to this new style is a work in progress. Many pages in the reference section are still in old teaching-from-example style. They will be converted piece by piece in the next release cycles, preferably in conjunction with simplification and unification of the API. The "Simulation model" section is already fully converted to the new style and gives an impression how the entire reference will look like in the future.

We are aware that users of the Python API want a documentation that is more "pythonic" in style and can be queried in an interactive session. Any action on this has to wait until we have decided whether we continue to generate Python bindings through Swig (which has severe limitations especially for functional programming) or whether we switch to some alternative technology.

10 Further plans

Our work plan for the next years comprises the following categories of tasks, in roughly decreasing order of priority:

(1) User support. Questions and suggestions should preferentially be submitted to the issue tracker at [2] so that discussions are publicly visible, and will aggregate into a searchable knowledge base. Confidential requests can be sent by mail to contact@bornagainproject.org. Our biennial School and User Meeting, interrupted by the Covid-19 pandemic, shall be relaunched soon.

(2) Further code consolidation, along with API unification and simplification. In the BornAgain core, we are currently unifying the handling of scans, coordinate axes and unit conversions. In the sample model, per legacy from IsGisaxs [17, 18], inter-particle correlations are currently treated as modifiers of particles. This hierarchy shall be inverted; particles shall be attached to lattices or other structural models, which in turn are attached to coherent surface or bulk domains. In the GUI, we want to reduce the amount of repetitive ("boilerplate") code and to further simplify some data structures and class hierarchies in order to facilitate future modifications and enhancements.

(3) Improvement of usability. We need to complete the online reference, make the example collection more orthogonal, and rework parts of the GUI, taking inspiration from other reflectometry software [19–23]. Developer-oriented code instrumentation shall be withdrawn from the published examples. To support different workflows, we will probably make the GUI multi-window capable so that users can freely select and arrange the data and model views they want to access simultaneously. In a related effort, we need to support simultaneous fits of multiple data sets, for instance from different measurements of the same sample.

(4) Acceleratation of computations. Absorption shall be handled at the level of horizontally averaged sliced layers, not the level of scattering particles. This will allow us to return from complex to real valued scattering vectors, which will substantially accelerate all form factor computations. Recently obtained series expansions [24] shall be used for the efficient computation of rotationally averaged form factors.

(5) Extension of functionality, mostly in response to user requests. The modelling and computational handling of instrumental resolution and modelling of mesoparticles shall be improved, the latter needs to be fully documented. To make progress on magnetic reflectivity and scattering, we depend on users proposing theoretical models.

Acknowledgements

We thank Randolf Beerwerth, Ludwig Jäck, Tobias Knopff, Andrew McCluskey, Gennady Pospelov, Matthias Puchner and Dmitry Yurov for contributions after release 1.16. We thank Thomas Arnold, Artur Glavic, Andrew McCluskey, Karolina Mothander, Thomas Holm Rod for criticism and suggestions. The extension of BornAgain to reflectometry was partly funded under schedule NIK4#11 as part of the data analysis and modeling work package to the contribution agreement between the European Spallation Source ERIC and Forschungszentrum Jülich. J.W. thanks Andreas Meyer and Stéphane Rols for hospitality at ILL in Grenoble.

References

[1] BornAgain home page, https://bornagainproject.org.

[2] BornAgain source repository, https://jugit.fz-juelich.de/mlz/bornagain.

[3] G. Pospelov, W. Van Herck, J. Burle, J. M. Carmona Loaiza, C. Durniak, J. M. Fisher, M. Ganeva, D. Yurov and J. Wuttke, J. Appl. Cryst. **53**, 262 (2020).

[4] BornAgain change log, https://jugit.fz-juelich.de/mlz/bornagain/-/blob/main/CHANGELOG.

[5] Helmholtz Authentication and Authorisation Infrastructure, https://hifis.net/aai.

[6] How to log in to Jugit, https://computing.mlz-garching.de/tech/jugit-login.

[7] BornAgain download directory, https://bornagainproject.org/ext/files.

[8] BornAgain Debian package, https://tracker.debian.org/pkg/bornagain.

[9] D. Holth, *PEP 427 — The Wheel Binary Package Format 1.0.* https://peps.python.org/pep-0427 (2012).

[10] BornAgain without GUI as Python package, https://pypi.org/project/BornAgain.

[11] J. Wuttke, S. Cottrell, M. A. Gonzalez, A. Kæstner, A. Markvardsen, T. H. Rod and G. Vardanyan, J. Neutron Res. **24**, 33 (2022).

[12] libheinz, C++ base library of Heinz Maier-Leibnitz Zentrum, https://jugit.fz-juelich.de/mlz/libheinz.

[13] J. Wuttke, J. Appl. Cryst. **54**, 580 (2021).

[14] libformfactor, C++ library for the computation of scattering form factors, https://jugit.fz-juelich.de/mlz/libformfactor.

[15] G. James, *The Tao of Programming*, InfoBooks: Santa Monica, Calif. (1987).

[16] E. S. Raymond, *The Art of Unix Programming*, Addison-Wesley: Boston (2003).

[17] R. Lazzari, J. Appl. Cryst. **35**, 406 (2002).

[18] R. Lazzari, *IsGISAXS*. Version 2.6. http://www.insp.jussieu.fr/oxydes/IsGISAXS/isgisaxs.htm (2006).

[19] A. R. J. Nelson and S. W. Prescott, J. Appl. Cryst. **52**, 193 (2019).

[20] O. V. Penkov, I. A. Kopylets, M. Khadem and T. Qin, SoftwareX **12**, 100528 (2020).

[21] M. Svechnikov, J. Appl. Cryst. **53**, 244 (2020).
[22] A. Koutsioubas, J. Appl. Cryst. **54**, 1857 (2021).
[23] A. Glavic, J. Appl. Cryst. **55**, 1063 (2022).
[24] M. Wagener and S. Förster, Sci. Rep. **13**, 780 (2023).