Accessing Particle Reference Data in High Energy Physics with Kotlin

Aleksei Kaplin *

Posted Date: 5 June 2026

doi: 10.20944/preprints202606.0482.v1

Keywords: Kotlin language; HEP software; high energy physics; scientific software; particle reference data

 doi:10.20944/preprints202606.0482.v1

Article

Accessing Particle Reference Data in High Energy Physics with Kotlin

Aleksei Kaplin

HSE University, Russia; mov.dx.010b@gmail.com

Abstract

High energy physics software uses reference particle data at all stages of modeling, reconstruction, and analysis. While programmatic access to Particle Data Group data is available through REST and Python-based solutions, there has been no native Kotlin implementation for the Java Virtual Machine ecosystem until now. This paper presents KParticle, a Kotlin-based solution for providing structured access to particle reference data using the SQLite database distribution. The system uses a multi-tiered architecture that separates client applications from database-specific implementation details, ensuring modularity and scalability. KParticle is suitable for integration into modeling environments, data analysis pipelines, validation tools, and machine learning applications, improving the integration of particle reference information.

Keywords: Kotlin language; HEP software; high energy physics; scientific software; particle reference data

1. Introduction

Modern experimental high-energy physics (HEP) is increasingly defined by large data volumes and complex, software-driven analysis pipelines. Physics analyses rely on both experimentally recorded data and simulated data samples. Simulated data are produced through multiple stages, including event generation and detector simulation, while experimental data undergo reconstruction and physics analysis. Across these workflows, consistent and authoritative particle reference information is required. As a result, structured and machine-readable particle data have become a critical dependency of contemporary HEP software.

Particle reference data are deeply embedded throughout the high-energy physics software stack. Detector simulation frameworks such as GEANT4 [1–3] depend on this information to accurately model particle interactions with detector materials. Event generators, including PYTHIA [4], HERWIG [5,6], and SHERPA [7], incorporate reference particle data to simulate hard scattering processes and subsequent event evolution. Reconstruction and analysis frameworks, typically built around ROOT [8] based workflows, rely on these data for a range of tasks in object identification and physics interpretation. In all these contexts, particle reference data serve as an implicit but essential component.

The Particle Data Group (PDG) collaboration provides the authoritative source of particle properties through the Review of Particle Physics [9]. The PDG serves as the de facto standard source for particle properties, fundamental interactions, and related metadata, ensuring coherence across theoretical and experimental studies. To support programmatic access to PDG data, several mechanisms are currently available. These include a REST-based API provided by the PDG [10], the PDG Python API, distributed as the pdg package [11], the scikit-hep/particle [12] Python package, part of the broader Scikit-HEP [13] project, and downloadable PDG database files.

At the same time, the Kotlin [14] programming language is gaining increasing attention as a candidate for the development of next-generation scientific software. Kotlin combines expressive language constructs with full compatibility with the Java Virtual Machine (JVM) ecosystem. The potential of Kotlin for scientific computing has been demonstrated across a growing number of projects: KMath, a Kotlin-based mathematics library [15]; a REST API and web interface for the Event

 doi:10.20944/preprints202606.0482.v1

Metadata System of the BM@N experiment [16]; detector simulation software for calculating the total muon flux observed by the Muon Monitor experiment [17]; VisionForge, a framework for 3D visualization of particle physics experiments [18]; and DataForge, a framework for automated data acquisition and storage [19], with demonstrated application to the Troitsk nu-mass experiment in the search for sterile neutrinos [20]. Despite this potential, there is currently no Kotlin implementation for structured access to PDG particle reference data.

This work presents a Kotlin solution, KParticle, designed to fill this gap. The goal of the presented solution is to enable the use of particle data in Kotlin-based HEP software while adhering to modern software development principles. The following sections of this paper describe the KParticle architecture, its implementation details, and application scenarios, demonstrating its potential in high-energy physics research.

2. Background

The PDG Review of Particle Physics is updated annually and published every two years in the journal HEP. It includes a compilation and evaluation of measurements of the properties of known elementary particles and summarizes searches for hypothetical new particles. For the 2024 edition, 2,717 new measurements from 869 papers were added, in addition to 46,838 measurements from 12,909 papers that first appeared in previous editions. 120 individual review articles discuss topics such as Higgs bosons, supersymmetry, Big Bang nucleosynthesis, probability, statistics, accelerators and detectors [21].

The particle data provided by PDG is distributed in various formats. These include a downloadable SQLite [22,23] database file.

The SQLite distribution integrates the complete PDG dataset into a single relational database [24]. This schema contains many interrelated tables that describe particle identifiers, particle properties, decay channels, experimental measurements, bibliographic references, and supporting documentation. The main tables of the database are presented in Table 1, and the relational structure and foreign key dependencies are shown in Figure 1. The figure provides a summary of the database schema and the columns of each table. The arrows indicate foreign key relationships. The columns in bold indicate constraints.

Table 1. Overview of the main tables in the PDG SQLite database.

No	Table	Description
1	pdgid	Central index of PDG Identifiers; referenced by other tables
2	pdgdata	Summary numerical data from the Review of Particle Physics
3	pdgparticle	Particle names, MC numbers, and quantum numbers (per charge state)
4	pdgitem	Reference names, aliases, and API strings
5	pdgitem_map	Mapping between generic and specific particle names or aliases
6	pdgdecay	Decay definitions with products and structure information
7	pdgmeasurement	Metadata for individual literature measurements
8	pdgmeasurement_values	Numerical results and uncertainties of measurements
9	pdgreference	Bibliographic reference information
10	pdgfootnote	Footnote texts for particle listings
11	pdgmeasurement_footnote	Mapping between measurements and footnotes
12	pdgtext	Section header text entries
13	pdgid_map	Mapping between related PDG Identifiers
14	pdginfo	Database edition and metadata information
15	pdgdoc	Documentation of schema codes and flags

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 June 2026 doi:10.20944/preprints202606.0482.v1

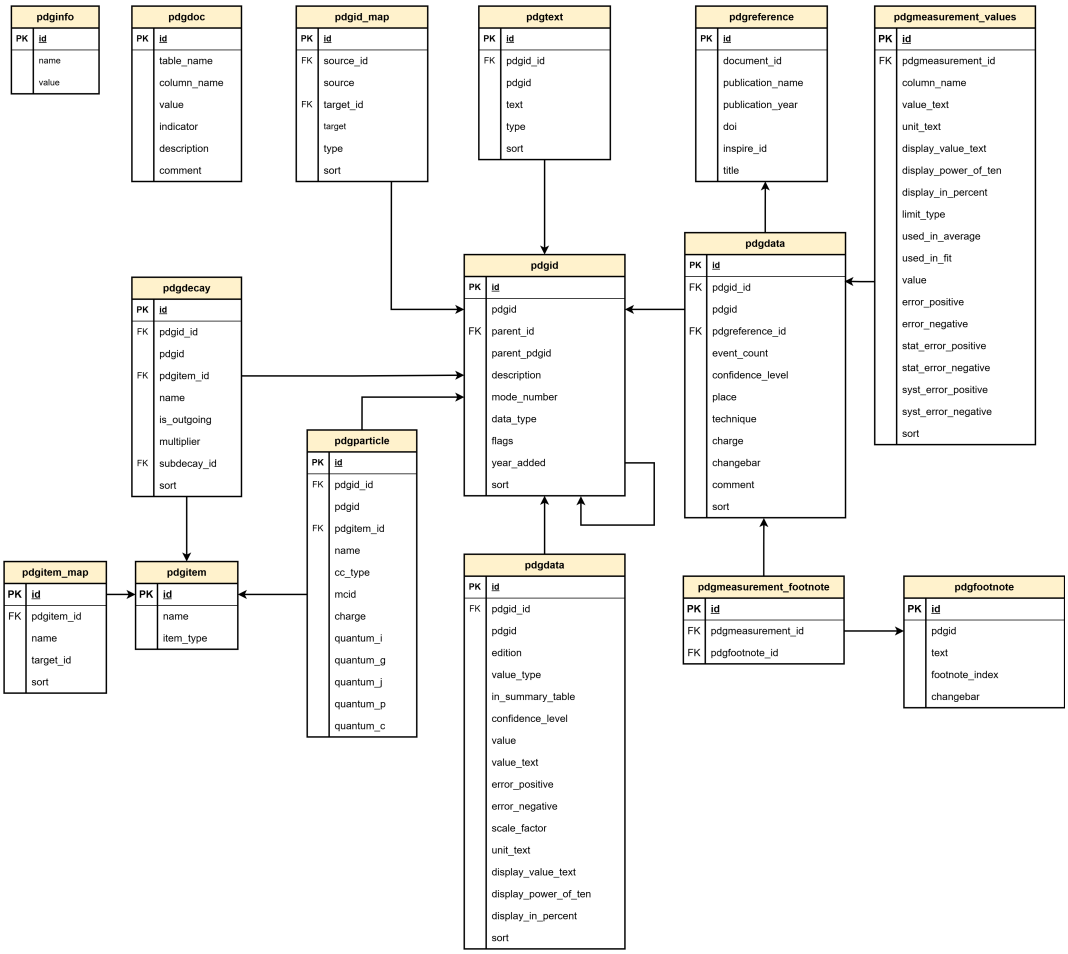

Figure 1. PDG database schema (reproduced from [24]).

3. KParticle

At a high-level, the architecture is a three-tier model consisting of a client tier, a middle-tier, and a data source tier (Figure 2, left).

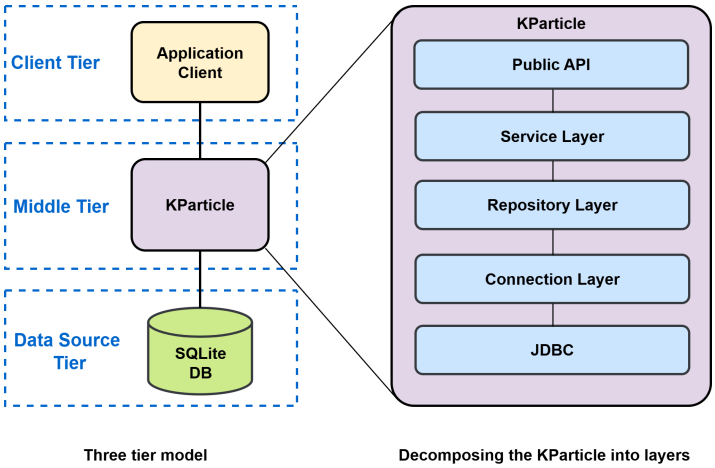

Figure 2. KParticle acts as a middle tier in a three-tier architecture, with the middle tier internally divided into logical layers.

The client tier includes external client applications, which are HEP software that depends on particle data. These may include simulation frameworks or analysis tools written in Kotlin. Clients interact exclusively with the public interfaces provided by KParticle. To ensure encapsulation, direct access to the database is intentionally prohibited.

 doi:10.20944/preprints202606.0482.v1

The middle tier is represented by the KParticle. This layer acts as an intermediary between clients and the underlying data source. It encapsulates domain logic related to particle properties, identifiers, etc., and provides a stable programming interface independent of data storage details.

The data source tier consists of a PDG SQLite database file. This database provides particle data and is accessible only through mechanisms implemented in the middle tier.

This separation ensures that the application code is independent of the database schema and storage format, thus increasing portability and reproducibility. Since the PDG database is updated every two years, with new versions replacing older ones, KParticle, by encapsulating all versioning logic and providing version-aware APIs, protects client applications from disruptive schema changes.

The internal structure of the KParticle is further illustrated in (Figure 2, right), which shows its internal decomposition into logically distinct layers.

The public API layer provides the interface through which client applications interact with the KParticle. The API is designed to remain stable between different versions, even as the underlying layers evolve.

Immediately beneath the public API is a service layer that acts as an intermediary and coordinates the processing of requests. It coordinates incoming API calls and data access components, applying any necessary validation, transformation, or workflow management, while remaining independent of data persistence issues.

The repository layer abstracts all data retrieval operations through repositories and data access objects (DAO).

The connection layer is responsible for loading database files and configuring JDBC.

Based on this conceptual diagram, Figure 3 provides a detailed view of the internal layers of KParticle at the class level. On the left side (Figure 3, left) the system is organized into five main layers, and each layer is represented by a set of classes associated with its functional role. The repository layer is further decomposed into internal components – repository, DAO, query, mapper, and model – highlighting the separation between domain modeling, query construction, data access abstraction, and object mapping. Surrounding classes such as `ParticleService`, `PdgIdRepository`, `ParticleRepository`, `PdgId`, `Particle`, `PdgIdDAO`, `ParticleDAO`, `PdgIdQuery`, `ParticleQuery`, and `DBConnection` are positioned to indicate that they belong to the main layer.

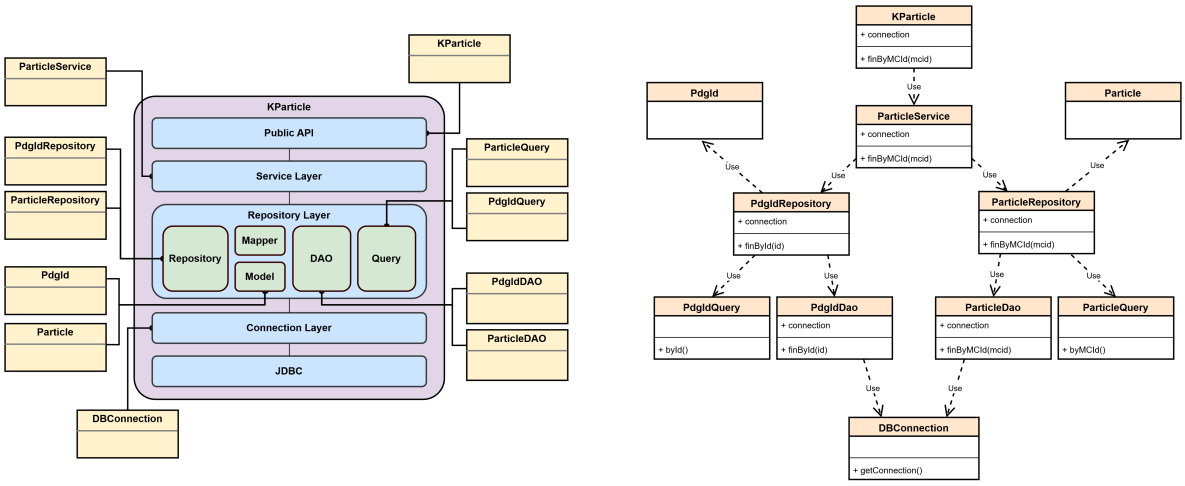

Figure 3. Architectural overview and UML class diagram of the KParticle: (left) layered system decomposition; (right) UML class diagram.

On the right side (Figure 3, right) is a UML class diagram. The `KParticle` class delegates operations to the `ParticleService` class, which interacts with repository classes such as `PdgIdRepository` and `ParticleRepository`. Each repository uses the corresponding DAO and Query classes (`PdgIdDao`, `ParticleDao`, `PdgIdQuery`, `ParticleQuery`) to build and execute database queries. To provide a connection to the database, DAO objects use `DBConnection`. Dashed arrows labeled "Use" indicate usage

 doi:10.20944/preprints202606.0482.v1

dependencies, clarifying the direction of interaction between classes. It should be noted that the diagram is representative rather than exhaustive; for clarity, only a sufficient subset of classes is shown, and the remaining classes are omitted.

4. Results

KParticle is implemented as a modular library in Kotlin that follows the structure described in the previous section and is built and managed using the Gradle [25] build system. Using Kotlin ensures full compatibility with the JVM ecosystem, and Gradle provides structured dependency management, automated testing support, and a reproducible build configuration. The resulting artifact can be integrated as a standard dependency into JVM-based HEP software projects.

Several representative use cases illustrate the applicability of KParticle to real-world HEP workflows.

1. Extraction of particle properties.
   Access to the basic properties of particles specified by the PDG identifier, such as mass, electric charge, lifetime, and spin, is a fundamental requirement for software in the field of high energy physics. Typical applications include initializing event records, setting simulation parameters, and validating particle definitions used in various software components.
2. Integration with Monte Carlo Simulations.
   Monte Carlo (MC) models used in high-energy physics require accurate data-driven input parameters to describe particle properties—such as energy, momentum, type—to accurately simulate primary interactions and subsequent particle trajectories.
3. Preparing a dataset for machine learning applications.
   Machine learning applications in high energy physics often require numerical feature vectors derived from particle properties such as mass, charge, spin, and lifetime.

The following examples demonstrate the practical use of KParticle. For instance, a particle can be retrieved by name as follows:

```
val particle = KParticle.findParticleByName("mu-")
println(particle)
```

```
Particle(name=mu-, pdgId=S004, ccType=S, mcId=13, charge=-1.0, I=null,
 ↪  G=null, J=1/2, P=null, C=null, flags=[L])
```

Similarly, particle retrieval can be executed using a Monte Carlo ID (MCID), followed by the verification of its particle classification.

```
val particle = KParticle.findParticleByMCId(13)
println(particle)
println(particle.isLepton())
```

```
Particle(name=mu-, pdgId=S004, ccType=S, mcId=13, charge=-1.0, I=null,
 ↪  G=null, J=1/2, P=null, C=null, flags=[L])
true
```

For example, historical particle mass measurements can be retrieved and visualized using Plotly.kt, a Kotlin wrapper for the popular Plotly visualization library [26].

```
val particle = KParticle.findParticleByMCId(13)
val (editions, masses, errors) = particle.getMasses()

```

 doi:10.20944/preprints202606.0482.v1


```
        val plot = Plotly.plot {
                trace {
                        x.numbers = editions
                        y.numbers = masses
                        error_y {
                                type = ErrorType.data
                                array = errors
                                visible = true
                                color("orange")
                        }
                }

                layout {
                        xaxis { title = "Edition" }
                        yaxis { title = "Mass [MeV]" }
                }
        }

        plot.makeFile()
```

The example below illustrates the energy loss of muons in standard rock as a function of kinetic energy.

```
        val particle = KParticle.findParticleByMCId(13)
        val (energies, dedx) = particle.getEnergyLoss("Standard Rock")

        val trace = Trace {
                x.set(energies)
                y.set(dedx)
        }

        val plot = Plotly.plot {
                traces(trace)
                layout {
                        xaxis {
                                title = "Muon Kinetic Energy [MeV]"
                                type = AxisType.log
                        }
                        yaxis {
                                title = "dE/dx [MeV cm²/g]"
                                type = AxisType.log
                        }
                }
        }

        plot.makeFile()
```

 doi:10.20944/preprints202606.0482.v1

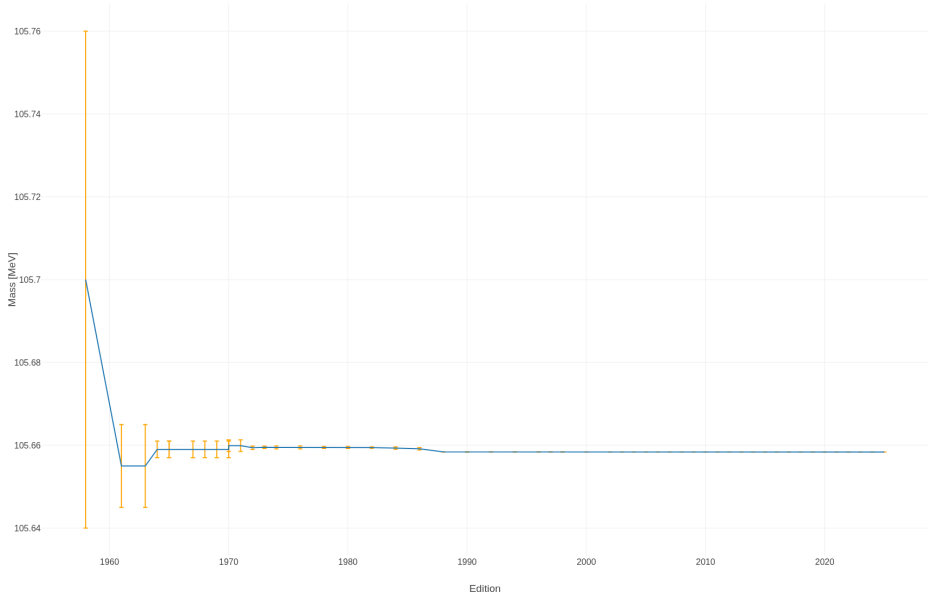

Figure 4. Historical measurements of the muon mass.

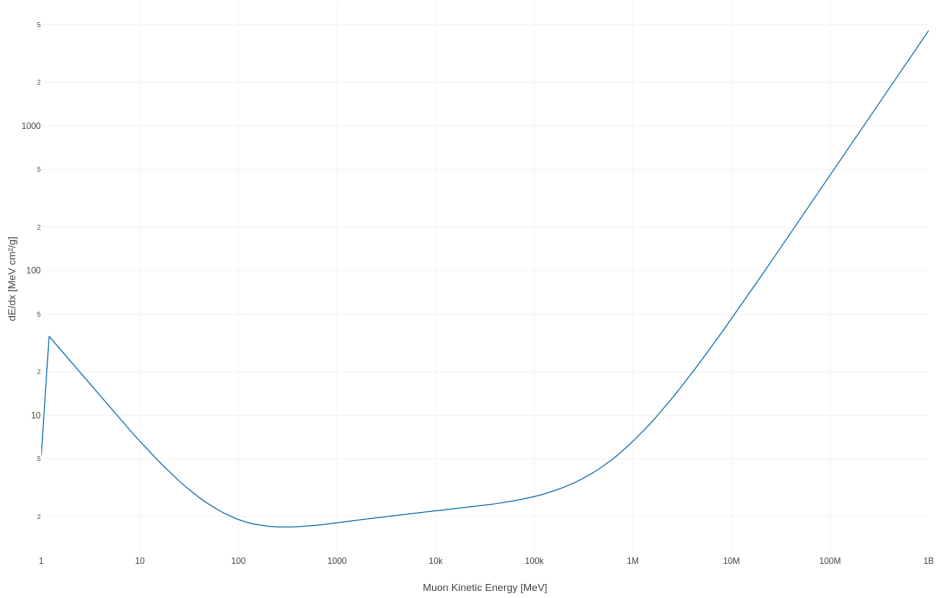

Figure 5. Muon energy loss in standard rocks as a function of kinetic energy.

5. Conclusions

This work presents KParticle, a software solution implemented in Kotlin that provides access to particle reference data in the JVM ecosystem. The system is designed with an emphasis on ease of use, modularity, and scalability, meeting the requirements of modern scientific computing. Its layered architecture ensures that client applications interact exclusively with high-level particle abstractions without access to database-specific implementation details. This solution can be used in scenarios involving analysis, validation, simulation, and machine learning. Further development will focus on integration with modeling and machine learning.

References

1. Allison, J.; Amako, K.; Apostolakis, J.; Arce, P.; Asai, M.; Aso, T.; Bagli, E.; Bagulya, A.; Banerjee, S.; Barrand, G.; et al. Recent developments in Geant4. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **2016**, *835*, 186–225. https://doi.org/https://doi.org/10.1016/j.nima.2016.06.125.

 doi:10.20944/preprints202606.0482.v1

2. Allison, J.; Amako, K.; Apostolakis, J.; Araujo, H.; Arce Dubois, P.; Asai, M.; Barrand, G.; Capra, R.; Chauvie, S.; Chytracek, R.; et al. Geant4 developments and applications. *IEEE Transactions on Nuclear Science* **2006**, *53*, 270–278. https://doi.org/10.1109/TNS.2006.869826.
3. Agostinelli, S.; Allison, J.; Amako, K.; Apostolakis, J.; Araujo, H.; Arce, P.; Asai, M.; Axen, D.; Banerjee, S.; Barrand, G.; et al. Geant4—a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **2003**, *506*, 250–303. https://doi.org/https://doi.org/10.1016/S0168-9002(03)01368-8.
4. Bierlich, C.; Chakraborty, S.; Desai, N.; Gellersen, L.; Helenius, I.; Ilten, P.; Lönnblad, L.; Mrenna, S.; Prestel, S.; Preuss, C.T.; et al. A comprehensive guide to the physics and usage of PYTHIA 8.3, 2022, [arXiv:hep-ph/2203.11601].
5. Bellm, J.; et al. Herwig 7.0/Herwig++ 3.0 release note. *Eur. Phys. J. C* **2016**, *76*, 196, [arXiv:hep-ph/1512.01178]. https://doi.org/10.1140/epjc/s10052-016-4018-8.
6. Bähr, M.; Gieseke, S.; Gigg, M.A.; Grellscheid, D.; Hamilton, K.; Latunde-Dada, O.; Plätzer, S.; Richardson, P.; Seymour, M.H.; Sherstnev, A.; et al. Herwig++ physics and manual. *The European Physical Journal C* **2008**, *58*, 639–707. https://doi.org/10.1140/epjc/s10052-008-0798-9.
7. Bothmann, E.; et al. Event generation with Sherpa 3. *JHEP* **2024**, *12*, 156, [arXiv:hep-ph/2410.22148]. https://doi.org/10.1007/JHEP12(2024)156.
8. Brun, R.; Rademakers, F. ROOT: An object oriented data analysis framework. *Nucl. Instrum. Meth. A* **1997**, *389*, 81–86. https://doi.org/10.1016/S0168-9002(97)00048-X.
9. Navas, S.; et al. Review of particle physics. *Phys. Rev. D* **2024**, *110*, 030001. https://doi.org/10.1103/PhysRevD.110.030001.
10. Group, P.D. The PDG REST API. https://pdgapi.lbl.gov/doc/restapi.html, 2026.
11. Group, P.D. The PDG Python API. https://pdgapi.lbl.gov/doc/pythonapi.html, 2026.
12. Rodrigues, E.; Schreiner, H. scikit-hep/particle: Version 0.26.1, 2026. https://doi.org/10.5281/zenodo.18197451.
13. Rodrigues, E.; et al. The Scikit HEP Project – overview and prospects. *EPJ Web Conf.* **2020**, *245*, 06028, [arXiv:physics.comp-ph/2007.03577]. https://doi.org/10.1051/epjconf/202024506028.
14. JetBrains. The Kotlin Programming Language. https://kotlinlang.org/, 2026.
15. Nozik, A. Kotlin language for science and Kmath library. *AIP Conference Proceedings* **2019**, *2163*, 040004. https://doi.org/10.1063/1.5130103.
16. Chebotov, A.; Degtyarev, A.; Gertsenberger, K.; Klimai, P. REST API and Web Interface for the Event Metadata System of the BM@N Experiment. *Phys. Part. Nucl. Lett.* **2023**, *20*, 1527–1530. https://doi.org/10.1134/S1547477123060092.
17. Bandac, I.; Bayo, A.; Bezrukov, L.; Enqvist, T.; Fazliakhmetov, A.; Ianni, A.; Inzhechik, L.; Joutsenvaara, J.; Kuusiniemi, P.; Loo, K.; et al. Calculation of total muon flux observed by Muon Monitor experiment. *Journal of Physics: Conference Series* **2017**, *934*, 012019. https://doi.org/10.1088/1742-6596/934/1/012019.
18. Nozik, A.; Dmitrieva, K.; Klimai, P.; teldufalsari.; SPC-code.; Postovalov, I. SciProgCentre/visionforge: 0.5.1, 2026. https://doi.org/10.5281/zenodo.18208430.
19. Nozik, A.; SPC-code.; mmkolpakov2002.; Postovalov, I.; Melgizin, M. SciProgCentre/dataforge-core: 0.10.2, 2025. https://doi.org/10.5281/zenodo.17971712.
20. Nozik, A. Declarative analysis in "Troitsk nu-mass" experiment. *Journal of Physics: Conference Series* **2020**, *1525*, 012024. https://doi.org/10.1088/1742-6596/1525/1/012024.
21. Group, P.D. About the Particle Data Group. https://pdg.lbl.gov/2025/html/about_pdg.html, 2026.
22. Hipp, R.D. SQLite, 2020.
23. Gaffney, K.P.; Prammer, M.; Brasfield, L.C.; Hipp, D.R.; Kennedy, D.R.; Patel, J.M. SQLite: Past, Present, and Future. *Proc. VLDB Endow.* **2022**, *15*, 3535–3547. https://doi.org/10.14778/3554821.3554842.
24. (PDG), P.D.G. PDG database file, 2024.
25. Inc., G. Gradle Build Tool. https://gradle.org/, 2026.
26. Nozik, A.; Samorodova, E.; SPC-code.; ArtificialPB.; Bion, J.; Joseph-Hui.; Nieboer, T.; zak cloudnc. SciProgCentre/plotly.kt: 0.7.1.1, 2024. https://doi.org/10.5281/zenodo.14380404.