Key4hep: Status and Plans

Placido Fernandez Declara[1], *Wenxing* Fang[2], *Frank* Gaede[3], *Gerardo* Ganis[1], *Benedikt* Hegner[1,4], *Clement* Helsens[1], *Xingtao* Huang[5], *Sang Hyun* Ko[6], *Teng* Li[5], *Weidong* Li[2], *Tao* Lin[2], *Thomas* Madlener[3], *Marko* Petric[1], *Andre* Sailer[1], *Valentin* Volkl[1], *Joseph* Wang[7], *Xiaomei* Zhang[2], and *Jiaheng* Zou[2]

[1]CERN, Geneva, Switzerland
[2]IHEP, Beijing, China
[3]DESY, Hamburg, Germany
[4]Institute for Advanced Computational Science, Stony Brook University, New York, USA
[5]Shandong University, Qingdao, Shandong, China
[6]Seoul National University, Seoul, Republic of Korea
[7]Bitquant Digital Services, Hong Kong

Abstract. Detector optimisation and physics performance studies are an integral part of the development of future collider experiments. The Key4hep project aims to design a common set of software tools for future, or even present, High Energy Physics projects. The proceeding describes the main components that are developed as part of Key4hep: the event data model EDM4hep, simulation interfaces to Delphes and Geant4, the k4MarlinWrapper to integrate iLCSoft components, as well as build and validation tools to ensure functionality and compatibility among the components. They also include the different adaptation processes by the CEPC, CLIC, FCC, and ILC communities towards this project, which show that Key4hep is a viable long term solution as baseline software for high energy experiments.

1 Introduction

The Key4hep project aims at creating a complete turnkey software stack for detector optimisation and performance studies for future experiments. Since the last CHEP conference [1, 2] much progress has been made in this project. The *vision* of Key4hep is to connect and extend individual packages from event generation to simulation and reconstruction into a complete data processing framework. To help users get started quickly, the stack has to be fully functional and come with plenty of examples that will allow new users to adapt it for their specific use case.

The source code of the software developed as part of this project is hosted on GitHub [3]. Documentation to start using the available software is available [4]. This documentation includes instructions for using existing installations from CVMFS, building the complete software stack, and for running simulation and reconstruction examples. Progress and issues of the project are discussed in a bi-weekly meeting.

At the time of the previous conference the Key4hep project consisted of the vision cited above, a prototype of the *k4MarlinWrapper* and the event data model EDM4hep was under

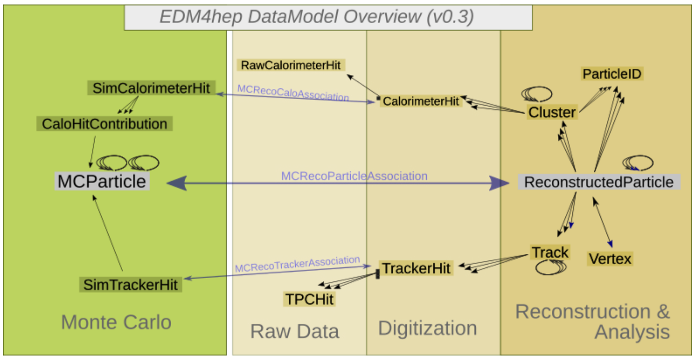

Figure 1. Schematic view of the hierarchical event data model of EDM4hep. Starting out with *MCParticle* objects for the Monte Carlo truth information on the left, additional data classes are defined following the typical event processing chain in HEP ending in the *ReconstructedParticle* class at the right.

development. Now the EDM4hep event data model (Section 2) is implemented via PODIO [5] and started to be used for fast and full simulation (Section 3). Section 4 describes the continued development of the wrapper around the Marlin based processors in iLCSoft. Sections 5 to 7 describe the approach of different communities in their adaptation towards Key4hep. Section 8 describes the status of the use of the Spack [6] package manager for the Key4hep stack, and explains the developments of continuous integration and validation systems. The contribution ends with a summary and outlook in Section 9.

2 Event Data Model

At the core of the Key4hep software stack is the new common event data model EDM4hep [7]. The EDM4hep event data model is based on LCIO [8], successfully used in the linear collider community for more than 15 years, and on FCC-EDM [9], combining the best of the two approaches. It is implemented, using the PODIO [2, 5] EDM-toolkit.

The hierarchical event data model implemented in EDM4hep and shown in Figure 1 follows the typical processing chain in HEP: starting with the *MCParticle* objects for the Monte Carlo truth information, combined with dedicated classes for simulated (*SimTrackerHit, SimCalorimeterHit*) and digitized hits (*TrackerHit, CalorimeterHit)*, to reconstructed *Track* and *Cluster* objects, to finally build *ReconstructedParticles*. Higher level object always point back to their constituent objects, for example a ReconstructedParticle points to the calorimeter clusters it contains, which point to Calorimeter Hits. By design there are no build-in links to Monte Carlo truth information in order to be applicable for real data, however dedicated relation objects to the truth information can be introduced for simulation. While the EDM focuses on the needs for precision physics at future lepton colliders, that often follow a Particle Flow approach, aiming at the reconstruction of every individual particle created in the collision, it should be equally well be applicable to future hadron collider projects. More details about the EDM4hep project can be found in [10].

In order to allow a smooth transition, involving the porting of the large code base in iLCSoft [11], originally developed for ILC and CLIC, and also used by CEPC and others, a dedicated k4LCIOReader has been implemented. This module allows one to read events from existing LCIO files and convert them to EDM4hep for further processing in Key4hep. Other conversion modules are under way.

3 Simulation Interfaces

Full and fast simulation in Key4hep are currently addressed based on the Delphes fast simulation and two options to interface to Geant4 with the option to add additional implementations in the future. As the EDM4hep output format of these simulation modules is equivalent or identical it will be possible to transparently exchange them.

3.1 k4SimDelphes

Delphes [12] is a C++ framework, performing a fast multipurpose detector response simulation. The simulation includes a tracking system, embedded into a magnetic field, calorimeters and a muon system. The framework is interfaced to standard file formats (e.g. Les Houches Event File or HepMC) and outputs observables such as isolated leptons, missing transverse energy and collection of jets which can be used for dedicated analyses. The simulation of the detector response takes into account the effect of the magnetic field, the granularity of the calorimeters and sub-detector resolutions. In the latest pre-release 3.4.3pre09 [13] Delphes provides parametrised track information with the full covariance matrix using the FastTrackCovariance software [14, 15].

The k4SimDelphes [16] project aims at converting Delphes objects to EDM4hep. For each reader already available in Delphes, an executable is created that writes EDM4hep output, for example, DelphesPythia8_EDM4HEP. To answer specific needs of using exclusive hadron decays, another executable using EvtGen [17] called DelphesPythia8EvtGen_EDM4HEP is also produced. The contents of the produced output files are configurable at runtime via an additional configuration file. These configuration settings steer which Delphes branches should be converted and also on how to combine different branches into different EDM4hep collections. A default configuration that should cover the majority of use cases and works with the default Delphes cards is available and is shipped with k4SimDelphes. In addition to these standalone executables, an integration of Delphes into the Gaudi [18] framework is currently under development, to facilitate its usage in more complex workflows.

3.2 k4SimGeant4

A Geant4 based simulation framework has been developed based on Gaudi which transparently supports full and fast simulation, where the default geometry is based on DD4hep detector models. A Gaudi algorithm is in charge of the simulation workflow, including the event generation, event simulation, storing MC truth information. The event generation is implemented as Gaudi tools to create a Geant4 event object for each event. For example, a Gaudi tool is implemented to convert the EDM4hep MCParticles to Geant4. The event simulation is wrapped with a Gaudi service and a customized Geant4 RunManager. The service provides the parameters to customize the random number engine, magnetic field, physics list, fast simulation. The customized run manager controls the event loop of Geant4 and is invoked by the simulation service. The MC truth information is handled in the so called Geant4 user actions, which are also implemented as Gaudi components.

3.3 DD4hep and DDG4

For standalone Geant4 simulations of detectors implemented in a DD4hep geometry, the DDG4 [19] package of DD4hep can also be used with the ddsim executable. The newly developed EDM4hep output plugin allows for a seamless integration with the other tools using the EDM4hep information.

4 k4MarlinWrapper for integration of iLCSoft components

iLCSoft [11] contains many tools to provide particle reconstruction functionality, such as the detector description or pattern recognition for track or calorimeter reconstruction. With *Marlin* [20], it also provides the orchestration layer that allows one to control the other pieces of these tools. To provide compatibility with *Marlin* inside Key4hep, the *k4MarlinWrapper* provides a *wrapper* that allows one to use its already proven [21] algorithms as part of a common framework based on Gaudi.

It is written in C++, and Python is used to configure the algorithm list, following the *Gaudi* framework usage. *Marlin* Processors are run by configuring *Gaudi* Algorithms that serve as a wrapper to instantiate them. This approach keeps the original *Marlin* processors functionality intact, allowing one to obtain the same behaviour and results as in using the Marlin framework alone. *k4MarlinWrapper* uses other Key4hep components for its functionality: k4FWCore is used for handling data inputs on reading events; k4LCIOReader is used as part of the event data model converters.

The EDM4hep Event Data Model is integrated into the Key4hep components, including *k4MarlinWrapper*. To interface between the different file formats used to configure the particle reconstruction, a converter from XML *Marlin* steering files to Python *Gaudi* options file is used. This converter parses the XML structure of the original file, to produce the corresponding *Gaudi* options file. This converter now accounts for optional processors that can be set on the XML file; these are converted as regular Marlin Processors and the addition of them to the algorithm list is left to the user to decide whether to include them or not. These algorithms are left *commented* as in `# algList.append(MyFastJetProcessor) # Config.OverlayNotFalse`, with a comment indicating the original configuration. The XML to Python converter also implements support for *constants* that can be used in the original XML files. These constants are provided as a Python dictionary with *key: value* entries that can be easily modified to propagate the changes to the rest of the Python file. To propagate the key-value pairs, *Python String formatting* is used in the rest of the file: `"%(DD4hepXMLFile_subPath)s" % CONSTANTS`. Because the original *constants* dictionary can itself contain references to other constants in the same dictionary, a pre-processing function is added at the beginning of the Python options file to guarantee the correct *String formatting* of all instances in the file.

Marlin Processors use the LCIO event data model; Key4hep will use EDM4hep across its software stack as a common event data model. *k4MarlinWrapper* supports conversion between the two different event models to be able to interface between both, add EDM4hep input to Marlin Processors, and be able to convert back the output of *Marlin* Processors to EDM4hep format. The converters are implemented to be run *in-memory*, and are only processed at run time when configured. The converters are implemented as *Gaudi* Tools to make them accessible from the entire framework, and for the converters to be able to access the different services and events. To configure them, the selected tool must be imported in the Python options file, its corresponding parameters must be indicated, and the Tool needs to be added to the corresponding algorithm that will make use of the conversions. For the converter Tools, the user needs to indicate tuples of three parameters: the Collection type to be converted, the name of the input (EDM4hep/LCIO) Collection, and the name of the output (LCIO/EDM4hep) Collection.

5 Adaptation of iLCSoft

The ILC and CLIC communities have jointly developed and maintained the iLCSoft framework over the last 15 years. This includes realistic and detailed detector models implemented in DD4hep and a large set of digitization and reconstruction algorithms implemented in Marlin, using the LCIO event data model. Both communities have created very large Monte Carlo data sets for detector optimization and physics studies. For both communities it is essential to keep the existing iLCSoft ecosystem and tools maintained and available for ongoing studies and provide an easy and straight forward path for migrating eventually to Key4hep. The latter is greatly facilitated by the *k4MarlinWrapper* in combination with *k4LCIOReader* as it allows to initially run the existing iLCSoft based full reconstruction chain in Key4hep. In the transition phase one can then port individual algorithms (Marlin processors) one by one in a mixed environment, until eventually all relevant central algorithms have been ported to Key4hep. ILC and CLIC will develop their individual schedules for the transition.

6 Adaptation of FCCSW

The experiment software for the FCC design studies has based itself on the Gaudi framework and the DD4hep detector description toolkit from the beginning. Adopting the Key4hep conventions is thus a smaller step than for other experiments in Key4hep. In addition, the event data model used for the studies up to the conceptual design report was based on the PODIO library, like EDM4hep. Still, the data model needed to be changed from FCC-EDM to EDM4hep.

Motivated by this upgrade process, it was decided to change the repository structure of FCCSW, splitting up the single FCCSW repository along its subdirectories into the following:

- k4FWCore [22]: Framework components for input/output and other fundamental functionality.
- k4Gen [23]: Framework components for generator integrations and particle guns.
- k4SimDelphes [16]: Framework components for Delphes simulations.
- k4SimGeant4 [24]: Framework components for Geant4 simulations.
- k4RecCalorimeter [25]: Framework components for calorimeter reconstruction, currently implementing both a Sliding Window and a Topoclustering Algorithm. Some parts specific to the FCC LAr calorimeter, but others of more general use.
- fccDetectors [26]: DD4hep models of the FCC detector geometries.
- dual-readout [27]: DD4hep model and simulation code for the FCC-ee DREAM dual readout calorimeter.

The resulting set of software packages now have more clearly defined dependencies and scopes. k4FWCore is already migrated to the Key4hep GitHub organization and used as a dependency for other framework packages. k4SimDelphes has been significantly rewritten to improve the original FCCSW code. All other packages prefixed k4* are foreseen to be migrated pending a review by the Key4hep software group. A first physics prospect study for the FCC-ee using k4SimDelphes and the EDM4hep data format was performed [28], demonstrating that the full chain is up and running for physics results.

7 Adaptation of CEPCSW

The CEPC experiment first started its detector design by reusing the iLCSoft software, based on which the CEPC conceptual design report [29] was completed. To facilitate further R&D activities, the experiment developed new CEPC software called CEPCSW which is fully integrated with the Key4hep. The CEPCSW adopted Gaudi as the underlying framework, EDM4hep as the official event data model and DD4hep for detector geometry description, and has started using the *k4FWCore* package (cf. Section 6). CEPC has thus completed the transition to Key4hep.

Within the CEPCSW, the simulation software was developed based on Geant4. The generator interface is implemented to read kinematics information from StdHep or LCIO files, then convert them into HepMC objects and further to the Geant4 event for detector simulation. The detector geometry managed by DD4hep is converted to the Geant4 geometry with converters from DDG4. The detector responses have been implemented as Gaudi tools and the produced intermediate hit objects are described by the DDG4 event data model. At the end of an event simulation, all the hit objects are converted to EDM4hep objects.

In the migration from iLCSoft to Gaudi, the Marlin processors used in CEPCSW were re-implemented as Gaudi algorithms. In order to facilitate the software transmission from iLCSoft to CEPCSW, *k4LCIOReader* was developed to convert the ILC format data to EDM4hep objects on the fly. With the help of k4LCIOReader, results from the same reconstruction algorithm can be conveniently compared between two different frameworks using identical inputs. After a complete validation, the full data processing chain consisting of physics generator, detector simulation, digitization and reconstruction was re-built in the Gaudi framework.

In addition to porting CEPC's reconstruction algorithms, adding new software packages like Pandora [30] to Key4hep also proved to be worthwhile practice, which could benefit both an individual experiment and the Key4hep project. The k4Pandora has been developed as an adapter to Pandora, which is responsible for translation of geometry as well as conversion of event objects between EDM4hep and Pandora's own event data model.

The CEPCSW software is built on the top of the Key4hep software and many other external libraries. All the external libraries can be obtained either by copying the pre-compiled libraries or generating them from the source code with the Spack and key4hep-spack tools.

8 Packaging, Deployment and Testing

Building, packaging and deploying software is a time-consuming necessity for HEP experiments. The difficulties in these areas arise from the need to integrate a large number of systems and packages from different sources while maintaining the ability to rapidly add new features and functionality. In addition, the fact that HEP software is typically used and developed by different groups with different infrastructures requires that any systems used for packaging, deployment, and testing be portable across different institutions and take into account the fact that versions, schedules, and development processes cannot be coordinated.

There is significant potential for common (community-wide) installations and tooling to make these processes more efficient, and to use systems and processes from the open source community and from outside of HEP. However, there are many challenges that need to be addressed. Scientific software may originate from many different institutions, and the distributed nature of scientific development makes it impossible to synchronize users into fixed release schedules. Moreover, the distributed nature of scientific development makes it quite likely that changes in one package will have unforeseen consequences in other systems which the author may not be aware of. In addition, different groups have different processes and will make different trade-offs between stability and development speed. Finally, the resources available for system integration and system wide testing are limited, resulting in the need to automate as many processes as possible.

In spite of these difficulties, experiment-independent software is already being built and deployed to CVMFS by the EP-SFT group at CERN. These LCG releases provide experiments with common HEP and general purpose libraries to use as dependencies for builds of experiment software. The Key4hep project tries to go beyond using a common installation of experiment-independent software and use the same tooling to build also experiment-specific software packages. Rather than use the LCGCMake [31] build tool currently used to build the LCG releases, the Key4hep group collaborated with SFT to evaluate the Spack package manager, which has already been discussed as a possible replacement for LCGCMake. A separate contribution in these proceedings reports in detail on the experiences building Key4hep software with Spack, and its potential to replace LCGCMake to build the LCG releases.

While the common build infrastructure has clear benefits, it imposes certain constraints on the participating experiments, as they need to agree on a common set of dependencies. In order to use new library versions with breaking changes, developers need to synchronise their upgrades in order to keep benefiting from a common software stack.

To give an example, this was recently the case with version v35r0 of the Gaudi framework toolkit, which required extensive changes to the CMake configuration of downstream packages. However, all Key4hep packages using Gaudi (FCCSW, CEPCSW, and the k4MarlinWrapper) were able to update their code in time for a common release, and also be more efficient doing so by sharing their expertise and experiences.

However, although synchronizing dependencies is necessary to take maximum advantage of shared code, the package manager Spack has a sophisticated dependency management system allowing packages to be built with different dependencies. As a result, development can continue while packages are standardized with newer code.

Still, with the large number of changes made to Key4hep code and underlying packages there is a likelihood that some change will be introduced that will break the build. To deal with this issue Key4hep uses a continuous integration process with virtualization to ensure that the additions to the software do not break the build. These systems make heavy use of open source virtualization systems to insure that code developed is easily portable, and the Key4hep distribution can be built as an OCI container [32] that can be run under docker [33], podman [34], or singularity [35].

To enable rapid continuous integration, a system called cacher has been developed that caches compiles and downloads and reduces build times and avoids the need for new complete builds [36]. Therefore images can be rapidly produced allowing for a continuous integration system such as buildbot [37] to continuously compile the system and insures that additions to the Spack distribution and Key4hep do not cause compilation errors, and to alert software authors immediately should system integration issues arise.

Both cacher and Key4hep OCI containers internally use the non-profit community-supported linux distribution Mageia Linux [38], which has incorporated various changes to the distribution in order to support Key4hep within the containers. The containers themselves are portable and can be run in any system that supports OCI containers.

Both Key4hep, Spack, and the containers have features that have proven useful for allowing rapid continuous development of complex software. At the Spack level and the continuous integration level, the systems can incorporate patches, which allows support staff to rapidly introduce minimal fixes to allow a system to continue to build and run, without requiring the author of the system to immediately make any changes. In addition, Spack allows for multiple versions of packages and dependencies to be built which allows for different packages to have different version dependencies in a reproducible manner.

Software validation ensures the software development is on the right track and provides a smooth development and releasing environment during the long life cycle of Key4hep. To develop an industry standard software validation system, we aim to build a CI/CD system based on the GitHub Action system. The validation system is currently under development, and we aim to provide the following features:

- Based on Catch2 and Pytest, unify the toolkit for making unit tests and performance tests.
- Construct the self-hosted GitHub runners on the dedicated or shared resources.

- Provide a friendly dashboard showing the results of the nightly build and unit tests for each project.

9 Conclusion

The Key4hep project aims at creating a complete software stack for detector optimisation and performance studies for future experiments. All major future collider groups: CEPC, CLIC, FCC and ILC are actively contributing to Key4hep and have developed migration schemes, have already started the migration of their existing software stacks, including first physics studies for the FCC-ee, or in case of CEPC, completed the adoption of Key4hep. Significant progress has been made in the underlying EDM4hep and implementing some first simulation tools, as well as in providing packaging, deployment and testing for Key4hep.

More effort is needed in the consolidation of the interfaces to the Geant4 simulation to avoid duplicated effort in the communities. With the availability of fast and full simulation, the *k4MarlinWrapper* and the core framework components the turnkey software stack is ready for use, as shown by the CEPC community. Better integration of existing functionality will be the next step for the project.

This group will actively continue to further develop the Key4hep software ecosystem and support the future collider projects in their transition to using it. New users and collaborators to this exiting experiment are highly welcome.

Acknowledgements

This work benefited from support by the CERN Strategic R&D Programme on Technologies for Future Experiments (https://cds.cern.ch/record/2649646/, CERN-OPEN-2018-006). This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 871072.

References

[1] A. Sailer, G. Ganis, P. Mato, M. Petrič, G.A. Stewart, EPJ Web Conf. **245**, 10002 (2020)
[2] F. Gaede, B. Hegner, G.A. Stewart, EPJ Web Conf. **245**, 05024 (2020)
[3] *Key4hep Github Repository*, `https://github.com/key4hep`
[4] Key4hep Software Group et al., *key4hep-doc: Documentation for key4hep-software* (2021), `https://doi.org/10.5281/zenodo.4564650`
[5] F. Gaede, B. Hegner, P. Mato, J. Phys. Conf. Ser. **898**, 072039 (2017)
[6] T. Gamblin, M. LeGendre, M.R. Collette, G.L. Lee, A. Moody, B.R. de Supinski, S. Futral, *The Spack package manager: bringing order to HPC software chaos*, in *SC15: International Conference for High-Performance Computing, Networking, Storage and Analysis* (IEEE Computer Society, Los Alamitos, CA, USA, 2015), `https://doi.org/10.1145/2807591.2807623`
[7] `https://github.com/key4hep/EDM4hep`
[8] F. Gaede, T. Behnke, N. Graf, T. Johnson, *LCIO — A persistency framework for linear collider simulation studies*, in *CHEP 2003* (La Jolla, California, 2003)
[9] `https://github.com/HEP-FCC/fcc-edm`
[10] F. Gaede, G. Ganis, B. Hegner, C. Helsens, T. Madlener, A. Sailer, G. Stewart, V. Volkl, J. Wang, *EDM4hep and PODIO- The event data model of the Key4hep project and its implementation* (2021), submitted to this conference

[11] *iLCSoft GitHub Repository*, `https://github.com/iLCSoft`

[12] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, M. Selvaggi (DELPHES 3), JHEP **02**, 057 (2014), `1307.6346`

[13] `https://github.com/delphes/delphes/tree/3.4.3pre09`

[14] `https://www.pi.infn.it/~bedeschi/RD_FA/Software/`

[15] `https://indico.cern.ch/event/783429/contributions/3376675/attachments/1829951/3712651/Oxford_April2019_V1.pdf`

[16] V. Volkl et al., *k4SimDelphes: Delphes Integration in the Key4hep Framework* (2021), `https://doi.org/10.5281/zenodo.4564683`

[17] `https://evtgen.hepforge.org/`

[18] LHCb Collaboration, ATLAS Collaboration, *Gaudi v33r0* (2019), `https://doi.org/10.5281/zenodo.3660964`

[19] M. Petric, M. Frank, F. Gaede, S. Lu, N. Nikiforou, A. Sailer, J. Phys. Conf. Ser. **898**, 042015 (2016), `https://doi.org/10.1088/1742-6596/898/4/042015`

[20] F. Gaede, Nucl. Instrum. Meth. **A559**, 177 (2006)

[21] D. Arominski, J.J. Blaising, E. Brondolin, D. Dannheim, K. Elsener, F. Gaede, I. García-García, S. Green, D. Hynds, E. Leogrande et al., *A detector for CLIC: main parameters and performance* (2018), `1812.07337`, `https://cds.cern.ch/record/2649437`

[22] V. Volkl et al., *k4FWCore: Core I/O Components for the Key4hep Framework* (2021), `https://doi.org/10.5281/zenodo.4564605`

[23] V. Volkl et al., *k4Gen: Generator components for the Key4hep framework* (2021), `https://doi.org/10.5281/zenodo.4564609`

[24] V. Volkl et al., *k4SimGeant4: Key4hep Framwework Integration for Geant4* (2021), `https://doi.org/10.5281/zenodo.4564574`

[25] V. Volkl et al., *k4RecCalorimeter: Calorimeter Reconstruction in the Key4hep Framework* (2021), `https://doi.org/10.5281/zenodo.4564669`

[26] V. Volkl et al., *fccDetectors: DD4hep Models for FCC Detectors* (2021), `https://doi.org/10.5281/zenodo.4564612`

[27] S. Ko et al., *dual-readout: DD4hep Geometry for a Dual-Readout Calorimeter* (2021), `https://doi.org/10.5281/zenodo.4564615`

[28] Y. Amhis, C. Helsens, D. Hill, O. Sumensari (2021), `2105.13330`

[29] M. Dong et al. (CEPC Study Group) (2018), `1811.10545`

[30] J.S. Marshall, M.A. Thomson, *Pandora Particle Flow Algorithm*, in *International Conference on Calorimetry for the High Energy Frontier* (2013), pp. 305–315, `1308.4537`

[31] J. Cervantes Villanueva, G. Ganis, D. Konstantinov, G. Latyshev, P. Mato Vila, P. Mendez Lorenzo, R. Pacholek, I. Razumov, EPJ Web Conf. **214**, 05020 (2019)

[32] J.C. Wang, *Key4Hep Container* (2021), `https://doi.org/10.5281/zenodo.4553189`

[33] D. Merkel, Linux J. **2014** (2014)

[34] *podman*, `http://docs.podman.io/`

[35] G.M. Kurtzer, V. Sochat, M.W. Bauer, PLOS ONE **12**, 1 (2017)

[36] J.C. Wang, *Cacher* (2021), `https://doi.org/10.5281/zenodo.4553191`

[37] *buildbot*, `https://buildbot.net/`

[38] Mageia.org Foundation, *Mageia Linux* (2019), `https://doi.org/10.5281/zenodo.4553891`