EIC Software Overview

David Lawrence[1,*]
on behalf of the *ePIC* Collaboration

[1]Thomas Jefferson National Accelerator Facility

Abstract. Development of the EIC project detector "ePIC" is now well underway and this includes the "single software stack" used for simulation and reconstruction. The stack combines several non-experiment-specific packages including ACTS, DD4hep, JANA2, and PODIO. The software stack aims to be forward looking in the era of AI/ML and heterogeneous hardware. A formal decision making process was implemented to choose the components that involved everyone in the collaboration that was interested. This talk will present an overview of the software stack currently used for development of the ePIC detector and on which we expect to execute the experiment.

1 Introduction

The ePIC detector is currently under development as the first detector of the new Electron Ion Collider (EIC) facility being constructed at Brookhaven National Laboratory(BNL). BNL and Jefferson Lab will be the host laboratories for the EIC Experimental Program. The detector is designed to measure energetic collisions of electrons and ions (including protons) with center of mass energies in the range of 20GeV - 140GeV and luminosities up to $10^{34} cm^{-2} s^{-1}$[1]. Figure 1 shows a diagram of the basic design of the central detector. The full, integrated detector, including the far forward and far backward detector regions, is shown in figure 2. With several subsystems the detector is expected to produce data at rates of $O(100)$Gbps out of the counting house[2]. Software is a critical component for integrating simulation, reconstruction, and analysis, all of which are required to complete and understand the detector design details and to execute the experiment when it begins data taking in the early 2030's. Choosing the components of the software stack has been done to ensure long term support, utilization of modern technologies, and flexibility to incorporate new technologies in the near future.

The ePIC software stack relies primarily on the C++ and Python programming languages. Currently, the C++17 standard is used and compilation assured using the gcc compiler family. A transition to the C++20 standard is expected soon with plans to follow new standards throughout the lifetime of the experiment as warranted. Support of older C++ standards will be phased out with details of the exact policy for doing so yet to be determined. Table 1 lists the core packages comprising the ePIC software stack, some of which are highlighted in later sections.

*e-mail: davidl@jlab.org

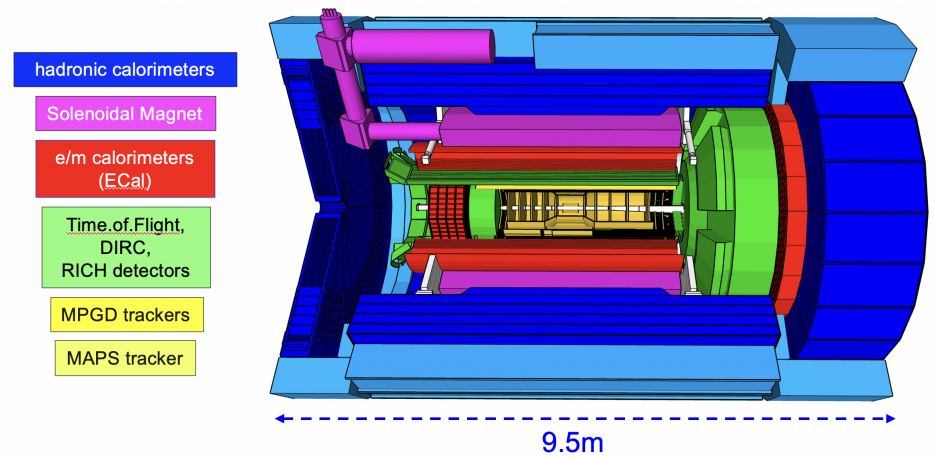


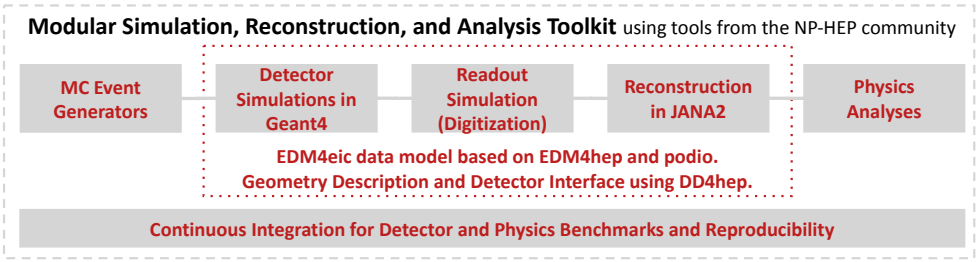
Figure 1. Profile view of the ePIC central detector at its present design stage. Far forward and far backward detectors near the beam lines are not shown.

Core packages in ePIC Software Stack	
PACKAGE	FUNCTION
geant4 python boost root	*(commonly use core packages)*
fmt spdlog	logging
JANA2	reconstruction framework
PODIO EDM4hep EDM4eic	data model + I/O
DD4hep	geometry
ACTS[3]	tracking

Table 1. Core pieces of the ePIC software stack for simulation, reconstruction, and analysis

2 Design Philosophy

Early in the decision making process of selecting the components of what would become the ePIC software stack, the EIC community developed a *Software Statement of Principles*[4]. The purpose was to provide a reference which could be used to help guide the decision making process. This proved to be very useful in framing discussions around specific packages and helped expedite the selection process. The software principles guided the definition of requirements for various parts of the software stack and informed decisions based on those requirements. An important theme contained in the *Software Statement of Principles* was modularity. This is considered critical for a software stack that will be used for the next 20 years. The goal of a modular software stack is to allow for new components and technologies to be adopted as they arise without having to redevelop other components. Modularity is implemented at multiple levels within the stack, The following diagram illustrates the top-level.



3 Containerization and CI

The computing plan for ePIC anticipates utilizing varied distributed compute resources including the Open Science Grid (OSG)[5][6], the SDCC at BNL, and the SciComp farm at JLab. The OSG alone includes a wide variety of operating systems so to impose uniformity, containerization is used. Singularity/Apptainer is supported in most of the available compute resources and so is the preferred system used by ePIC. Docker images for Intel and ARM architectures are also available for users running on local non-Linux resources such as Windows and MacOS. Singularity images are distributed via *cvmfs* (see */cvmfs/singularity.opensciencegrid.org/eicweb*). They contain not only binaries of all of the packages in the software stack, but a C++ compiler, Python interpreter, and the development tools (e.g. *cmake*, *git*, etc...) needed to develop software for ePIC. The containers are thus the primary tool for development and production. They are also used for the Continuous Integration (CI) system used to ensure software entering the repositories passes a series of integration tests before being merged into the main branch. Linux users are typically able to retrieve a container and run the software locally on their laptop or PC with a small set of commands (e.g. *curl - -location https://get.epic-eic.org | bash*).

The CI system uses GitHub runners to automatically run a series of integration and other tests periodically and upon every Pull Request (PR). On GitHub, we use a cvmfs and singularity action to load the environment for each job step (using caching for minimal overhead). After an initial set of compilation checks and unit tests, GitHub triggers more extensive benchmark suites on a self-hosted GitLab instance at Argonne National Lab, where 256 cores stand ready to run longer simulations of multiple detector configurations and reference physics processes. Completed benchmarks are reported back to the PRs on GitHub, where they can be consulted for review.

4 The ePIC Software Stack

4.1 Detector Geometry

For the detector geometry definition and exchange the *DD4hep* package[7] was chosen. This uses an XML format to specify the geometry that can be used for both the simulation (via GEANT4) and the reconstruction. The schema includes support for specifying a *readout* type for each detector that a generic GEANT4 program called *ddsim* can use to identify a C++ plugin to implement the readout for the detector. For the purposes of the EIC, an augmented version of this program called *npsim*[8] is used which allows tuning the physics as well as additional levels of customization over the generic *ddsim*. Both *npsim* and *ddsim* are distributed in the container images. The geometry is organized in a set of directories containing the files for a particular group of detectors while the files themselves correspond to specific detectors or components[9]. The system also supports specifying views of the

detector that can be used for visualization in yaml files. Several of these are maintained in the same repository as the geometry and readout plugin code.

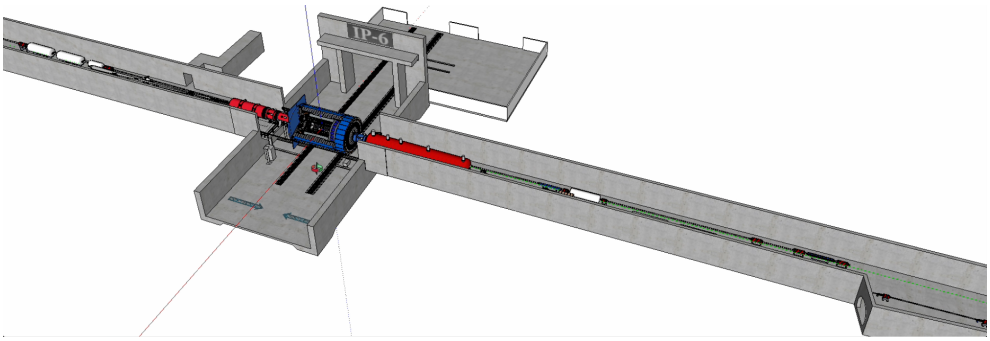


Figure 2. ePIC detector including the far forward an backward regions.

Figure 2 shows an example rendering of the full detector geometry. This includes the central detector and the for forward and far backward detector regions. The ePIC detector will be placed at the "6 o'clock" interaction point position or *IP6*.

4.2 Data Model and I/O

The data model used for storing simulated and reconstructed data is defined in a set of yaml files compatible with the *PODIO*[10] package. *PODIO* provides tools to generate C++ code from the yaml source that provide data structures and memory management routines that can be used to manage the objects in memory, write them to a ROOT file, and read them from a ROOT file. The ePIC data model is comprised of a base model (*EDM4hep*[11]) and an augmentation (*EDM4eic*[12]). The *EDM4hep* model is generic and intended to be non-experiment specific. It is maintained outside of the ePIC collaboration, but ePIC members do contribute to it regularly. The *EDM4eic* data model is written and maintained by the ePIC collaboration. It augments the *EDM4hep* data model, but allows for faster implementation into tagged releases.

```
205     edm4eic::CalorimeterHit:
206       Description: "Calorimeter hit"
207       Author: "W. Armstrong, S. Joosten"
208       Members:
209         - uint64_t          cellID           // The detector specific (geometrical) cell id.
210         - float             energy           // The energy for this hit in [GeV].
211         - float             energyError      // Error on energy [GeV].
212         - float             time             // The time of the hit in [ns].
213         - float             timeError        // Error on the time
214         - edm4hep::Vector3f position         // The global position of the hit in world coordinates [mm].
215         - edm4hep::Vector3f dimension        // The dimension information of the cell [mm].
216         - int32_t           sector           // Sector that this hit occurred in
217         - int32_t           layer            // Layer that the hit occurred in
218         - edm4hep::Vector3f local            // The local coordinates of the hit in the detector segment [mm].
```

Figure 3. Example code from the EDM4eic data model yaml file. (See https://github.com/eic/EDM4eic/blob/main/edm4eic.yaml for full source.)

Figure 3 shows a snippet of the *EDM4eic* data model yaml file. PODIO encourages flat data structures consisting mainly of "*plain 'ol data*" types such as floats and integers. This

makes the ROOT files it produces easy to work with for people familiar with ROOT. Support for specifying one-to-many and many-to-many relations is also present in the *PODIO* schema.

4.3 Reconstruction Framework

The reconstruction framework used in the ePIC software stack is JANA2[13][14], a multi-threaded framework designed for particle physics experiments in NP and HEP[15]. The JANA framework has been successfully used in the past for processing raw data at large HTC/HPC facilities[16]. JANA2 also has new features specifically for supporting streaming systems[17][18]. Active support for the project at Jefferson Lab, a host laboratory for the EIC experimental program, motivated its selection for ePIC.

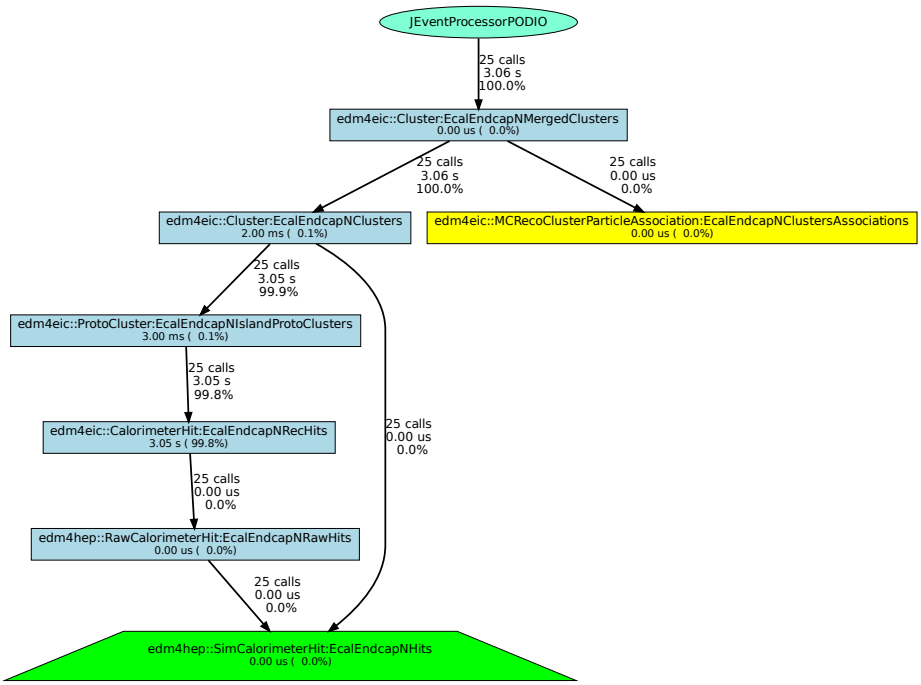


Figure 4. Augmented dependency graph of the reconstruction chain for a single detector in JANA2 framework (see text for details).

Figure 4 shows an augmented dependency graph for the reconstruction chain of a single detector. In this case the Electromagnetic Endcap Calorimeter for the electron-going direction. The cyan colored oval at the top represents the top-level *processor* object that initially requests the fully reconstructed objects. The blue rectangles represent algorithms required for various stages of the reconstruction. The green trapezoid at the bottom represents objects obtained from the event source (e.g. data file). The yellow rectangle represents associations objects that are also created by the clusters factory. In this graph, the requests for objects flows from the top to the bottom while the reconstructed data objects flow from the bottom to the top in response.

Support for AI/ML using heterogeneous hardware in the reconstruction will be supported through the sub-event feature in JANA2. This allows algorithms to place work requiring heterogeneous hardware in a special queue in the topology. From there it can be batched from multiple events before being sent to the hardware and the output assets recombined with their respective events upon return. Details of the implementation are still under development.

5 Detector Design

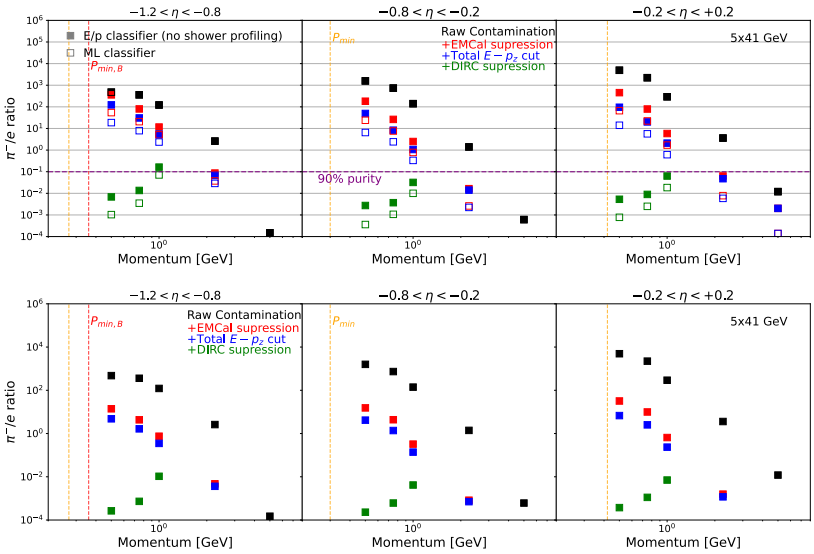


Figure 5. Example of comparative study for different designs of the barrel EM calorimeter using parts of the ePIC software stack. TOP: SciGlass calorimeter technology (*courtesy Dmitry Kalinkin*). BOTTOM: Imaging calorimeter technology (*courtesy Maria Żurek*).

The software stack has been useful in helping to finalize design decisions between different detector technologies considered for ePIC. Even without a fully developed reconstruction chain available, the *DD4hep*, *GEANT4*, and *PODIO* packages can be used to simulate different detector designs and geometries for comparison. Figure 5 shows an example of this where two candidate technologies for the EM barrel calorimeter where studied to determine particle identification capabilities as a function of momentum in different pseudo-rapidity regions. The PODIO output from the simulation was in the form of standard ROOT files that made them easily accessible for ad-hoc analyses.

6 Summary

The ePIC collaboration at the EIC has selected a software stack based on a guiding set of software principles. The stack includes the general use packages *DD4hep*, *GEANT4*, *PODIO*, and *JANA2* among others. Containerization is used both for a full featured CI chain and for software development by collaborators. The software ecosystem is designed to meet the needs of the EIC which is expected to begin data taking in the early 2030s.

Acknowledgement

This material is based upon work supported, in part, by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics under contract DE-AC05-06OR23177.

References

[1] R. Abdul Khalek, A. Accardi, J. Adam, D. Adamiak, W. Akers, M. Albaladejo, A. Albataineh, M. Alexeev, F. Ameli, P. Antonioli et al., Nuclear Physics A **1026**, 122447 (2022)

[2] J.C. Bernauer, C.T. Dean, C. Fanelli, J. Huang, K. Kauder, D. Lawrence, J.D. Osborn, C. Paus, J.K. Adkins, Y. Akiba et al., *Scientific Computing Plan for the ECCE Detector at the Electron Ion Collider* (2022), https://arxiv.org/abs/2205.08607

[3] X. Ai, C. Allaire, N. Calace, A. Czirkos, M. Elsing, I. Ene, R. Farkas, L.G. Gagnon, R. Garg, P. Gessinger et al., Computing and Software for Big Science **6**, 8 (2022)

[4] *EIC Software Statement of Principles*, https://eic.github.io/activities/principles.html

[5] T.O.S.G.E.B. on behalf of the Osg Consortium:Ruth Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Würthwein et al., Journal of Physics: Conference Series **78**, 012057 (2007)

[6] I. Sfiligoi, D.C. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, F. Wurthwein, *The Pilot Way to Grid Resources Using glideinWMS*, in *2009 WRI World Congress on Computer Science and Information Engineering* (2009), Vol. 2, pp. 428–432

[7] *DD4hep Github site*, https://github.com/AIDASoft/DD4hep

[8] *npsim program Github site*, https://github.com/eic/npsim

[9] *ePIC Geometry Github site*, https://github.com/eic/epic

[10] *PODIO Github site*, https://github.com/AIDASoft/podio

[11] *EDM4hep Github site*, https://github.com/key4hep/EDM4hep

[12] *EDM4eic Github site*, https://github.com/eic/EDM4eic

[13] *JANA2 Multi-threaded HENP Event Reconstruction* (2023), https://doi.org/10.5281/zenodo.8173154

[14] *JANA2 Github site*, https://github.com/JeffersonLab/JANA2

[15] Lawrence, David, Boehnlein, Amber, Brei, Nathan, *JANA2 Framework for Event Based and Triggerless Data Processing* (2020), https://doi.org/10.1051/epjconf/202024501022

[16] Lawrence, David, *Offsite Data Processing for the GlueX Experiment* (2020), https://doi.org/10.1051/epjconf/202024507037

[17] F. Ameli, M. Battaglieri, V.V. Berdnikov, M. Bondì, S. Boyarinov, N. Brei, L. Cappelli, A. Celentano, T. Chiarusi, R.D. Vita et al., *Streaming readout for next generation electron scattering experiment* (2022), 2202.03085

[18] Ameli, Fabrizio, Battaglieri, Marco, Bondí, Mariangela, Celentano, Andrea, Boyarinov, Sergey, Brei, Nathan, Chiarusi, Tommaso, De Vita, Raffaella, Fanelli, Cristiano, Gyurjyan, Vardan et al., *Streaming Readout of the CLAS12 Forward Tagger Using TriDAS and JANA2* (2021), https://doi.org/10.1051/epjconf/202125104011