The Journal of Open Source Software

Go-HEP: libraries for High Energy Physics analyses in Go

Sebastien Binet[1], **Bastian Wieck**[3], **David Blyth**[2], **Emmanuel Busato**[1], **Michaël Ughetto**[4], **and Peter Waller**[5]

1 IN2P3 **2** Argonne National Laboratory **3** Georgia Southern University **4** Stockholm University **5** University of Liverpool

DOI: 10.21105/joss.00372

Software

- Review
- Repository
- Archive

Summary

Go-HEP provides tools to interface with CERN's ROOT (Brun and Rademakers 1997) software framework and carry analyses or data acquisition from within the Go (Authors, n.d.) programming language.

Go-HEP exposes libraries to read and write common High Energy Physics (HEP) file formats (HepMC (Dobbs and Hansen 2000), LHEF (Alwall et al. 2007), SLHA (Skands and others 2004)) but, at the moment, only read interoperability is achieved with ROOT file format. Go-HEP also provides tools to carry statistical analyses in the form of 1-dim and 2-dim histograms, 1-dim and 2-dim scatters and n-tuples. Go-HEP can also graphically represent these results, leveraging the Gonum (T. G. authors, n.d.) plotting library.

Motivations

Writing analyses in HEP involves many steps and one needs a few tools to successfully carry out such an endeavour. But - at minima - one needs to be able to read (and possibly write) ROOT files to be able to interoperate with the rest of the HEP community or to insert one's work into an already existing analysis pipeline.

Go-HEP provides this necessary interoperability layer, in the Go programming language. This allows physicists to leverage the great concurrency primitives of Go, together with the surrounding tooling and software engineering ecosystem of Go, to implement physics analyses.

Packages

In this section, we list the packages that Go-HEP provides together with a short description.

`go-hep/rootio`

`go-hep/rootio` provides read access to data stored in ROOT files. `rootio` is a pure-Go implementation of the ROOT file format, reimplementing from scratch the decoding and interpretation of ROOT's low-level file blocks (`TDirectory`, `TKey`, `TBuffer` and `TFile`) together with file metadata, such as `TStreamerInfo` and `TStreamerElement` that allow to serialize and deserialize C++ class' instances. 1- and 2-dimensional histograms (`TH1x` and `TH2x`) can be extracted from ROOT files using `rootio`, as well as n-tuple data (`TTree`).

Go-HEP can also convert ROOT histograms into YODA's (T. Y. authors, n.d.) or Go-HEP's native format.

`go-hep/rootio` is the gateway to the wealth of data generated, simulated and acquired by LHC experiments.

Right now, `go-hep/rootio` can only read ROOT data. Work is in progress to also create ROOT-compatible data files.

`go-hep/fmom`

The `go-hep/fmom` package provides different representations of Lorentz vectors (4-components vectors that represent the momentum and energy of particles.) `go-hep/fmom` currently implements the `(px,py,pz,e)`, `(e,eta,phi,m)`, `(inv-pt, cot-theta, phi, m)` and `(pt, eta, phi, m)` Lorentz vector representations and operations that apply on these 4-vectors.

`go-hep/hbook` and `go-hep/hbook/ntup`

`go-hep/hbook` provides 1- and 2-dimensional histograms, 1-dim profiles and 2-dim scatters, for the statistical representation of variables. Notably, histograms allow to retrieve the mean, standard deviation, standard error and the sum of weights of a given variable.

`go-hep/hbook/ntup` provides a way to create, open and iterate over n-tuple data: rows of records. `ntup` can be seen as a database of HEP event data. Indeed, `ntup` is built on Go's `database/sql` package that allows to interact with various databases via plugins (`SQLite`, `MySQL`, …) `go-hep/hbook/ntup/ntupcsv` is a convenience package that allows to load a `CSV` file in memory and work on it as a database.

`go-hep/hepmc`, `go-hep/hepevt`, `go-hep/heppdt`, `go-hep/lhef` and `go-hep/slha`

`go-hep/hepmc` and `go-hep/hepevt` are packages used to represent HEP event interactions: they carry informations about a fundamental event interaction (incoming partons) and the decay chains, down to the hadronization and stable final particles. `hepevt` is the implementation of the old `FORTRAN` standard, while `hepmc` is the new `C++` representation. `hepmc` files are usually the result of a Monte-Carlo generator where the simulation of physics processes (branching ratios, hadronization, …) has been implemented. Go-HEP doesn't currently provide Monte-Carlo generators.

`heppdt` provides access to the HEP particle data table: a catalog of informations (mass, width, …) about all particles of the Standard Model (and beyond the Standard Model.)

`lhef` (Alwall et al. 2007) and `slha` (Skands and others 2004) are exchange format files for Monte-Carlo generators.

`go-hep/fwk`

`fwk` is a pure-Go, concurrent implementation of a typical HEP control framework. Control frameworks are used in HEP to collect the acquired data from *e.g.* the LHC experiments and channel it through the various reconstruction algorithms. Control frameworks are the angular stone of HEP experiments, allowing to organize and schedule the algorithms that identify and reconstruct electrons, muons, etc.. from electronic signals coming from the detectors.

`go-hep/fwk` leverages the concurrency primitives of Go to provide an easy to use toolkit, ready to harness the power of multicore machines. `go-hep/fwk` parallelizes work at the

event level, allowing to work on multiple events at a time. `go-hep/fwk` also parallelizes work at the sub-event level, scheduling in parallel algorithms that don't need inputs from each other. `go-hep/fwk` doesn't provide extra tools to perform parallel work at the sub-algorithm level, leaving the algorithms' implementors with all the tools from the Go language (*e.g.* `channels`, `goroutines` and `sync.Wait` or `golang.org/x/sync/errgroup`.)

`go-hep/fads`

`fads` is a Fast Detector Simulation toolkit. It uses `fwk` to provide a concurrency-friendly toolkit that performs so-called fast-simulation of detectors, with components for:

- particles' isolation
- b-tagging, tau-tagging
- calorimetry simulation
- energy scaling and energy smearing
- momentum smearing
- particles propagation in a magnetic field
- jets finding (using `go-hep/fastjet`)

`go-hep/fads` is a pure-Go reimplementation of a `C++` program called Delphes (Favereau et al. 2014). `go-hep/fastjet` is a pure-Go reimplementation of a `C++` library called FastJet (Cacciari, Salam, and Soyez 2012).

`go-hep/lcio`, `go-hep/sio` and `go-hep/rio`

`lcio` implements read/write access to the binary file format developed for the Linear Collider community. It is a pure-Go reimplementation of a C++ library of the same name (T. L. authors, n.d.). `sio` is the low-level access to this binary file format, `lcio` implements the event data model that has been used by the LC community.

`go-hep/rio` is a fork of `lcio` to provide 64b indexing and thus support >4Gb files.

`go-hep/hplot`

`hplot` is a package providing building blocks for creating plots. It leverages the Gonum (T. G. authors, n.d.) plotting package to create HEP oriented plots, using `go-hep/hbook` data as input. `hplot` can generate publication quality plots in PNG, SVG, PDF or EPS formats. `hplot` also provides a work-in-progress interactive shell with an OpenGL backend (on Linux or macOS; Windows will use the `win32` API) to display plots.

Commands

Go-HEP also provides a few commands that use Go-HEP packages.

`cmd/root2npy`

The command `root2npy` uses the `go-hep/rootio` package to convert n-tuple data stored in ROOT files into NumPy (Kern, n.d.) array data files. This command does not need any ROOT installation nor any NumPy installation: everything has been implemented in pure-Go from first principles. This is useful to quickly convert data and interoperate with analyses' pipelines that are mostly python based.

`cmd/root2yoda`

The `root2yoda` command converts ROOT files containing histogram values into their YODA equivalent, saved in a YODA ASCII file.

`go-hep/rootio/cmd/root-ls` and `root-srv`

The `root-ls` command allows to quickly inspect and display the content of a ROOT file from the shell.

The `root-srv` command launches a local web server that can be used to inspect graphically ROOT files, displaying histograms and n-tuple data.

References

Alwall, J., A. Ballestrero, P. Bartalini, S. Belov, E. Boos, A. Buckley, J.M. Butterworth, et al. 2007. "A Standard Format for Les Houches Event Files." *Computer Physics Communications* 176: 300–304. doi:10.1016/j.cpc.2006.11.010.

Authors, The Go. n.d. "The Go Programming Language." https://golang.org.

authors, The Gonum. n.d. "Gonum: Consistent, Composable and Comprehensible Scientific Code." https://gonum.org.

authors, The LCIO. n.d. "Linear Collider I/O." https://github.com/iLCSoft/LCIO.

authors, The YODA. n.d. "YODA: Yet More Objects for Data Analysis." https://yoda.hepforge.org.

Brun, R., and F. Rademakers. 1997. "ROOT: An object oriented data analysis framework." *Nucl. Instrum. Meth.* A389: 81–86. doi:10.1016/S0168-9002(97)00048-X.

Cacciari, Matteo, Gavin P. Salam, and Gregory Soyez. 2012. "FastJet User Manual." *Eur. Phys. J.* C72: 1896. doi:10.1140/epjc/s10052-012-1896-2.

Dobbs, M, and J B Hansen. 2000. "The HepMC C++ Monte Carlo Event Record for High Energy Physics," no. ATL-SOFT-2000-001 (June). CERN. http://cds.cern.ch/record/684090.

Favereau, J. de, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, and M. Selvaggi. 2014. "DELPHES 3, A modular framework for fast simulation of a generic collider experiment." *JHEP* 02: 057. doi:10.1007/JHEP02(2014)057.

Kern, Robert. n.d. "A Simple File Format for Numpy Arrays." https://docs.scipy.org/doc/numpy/neps/npy-format.html.

Skands, Peter Z., and others. 2004. "SUSY Les Houches accord: Interfacing SUSY spectrum calculators, decay packages, and event generators." *JHEP* 07: 036. doi:10.1088/1126-6708/2004/07/036.