GLAS: General Loop Amplitude System

A Practical Manual and Tutorial for Automated One-Loop Workflows

M. Houmani
High Energy Physics Group
mahdi.houmani@email.com

January 25, 2026

Abstract

GLAS (General Loop Amplitude System) is a workflow orchestrator for perturbative calculations in high-energy physics. It automates and manages the standard one-loop pipeline by integrating established tools for diagram generation, symbolic manipulation, and reduction. Rather than replacing these tools, GLAS provides a run-based execution model, a reproducible directory layout, parallel driver generation, and a consistent command interface. This manual focuses on *how to use* GLAS as a tutorial and operational reference: how to run a process end-to-end, where outputs are written, how the stages connect, and how to debug and extend the system. All external packages used by GLAS are cited throughout.

Contents

1 What GLAS Is (and What It Is Not)

GLAS is designed for practitioners who already rely on specialized HEP software and want a robust way to orchestrate complex multi-stage computations.

GLAS is

- A **workflow orchestrator** and **run manager** that stitches together multiple external tools.
- A **parallel driver generator** for FORM tasks and a structured logging system.
- A **reproducible execution environment**: each calculation lives in a self-contained run directory.

GLAS is not

- A replacement for QGRAF [1], FORM [2,3], MATHEMATICA, or reduction software.
- A monolithic symbolic engine: it delegates heavy symbolic work to established packages and focuses on coordination and reproducibility.

2 External Toolchain and Citations

GLAS relies on a standard ecosystem of tools. The exact installation method may differ between machines, but the conceptual dependency graph stays the same.

2.1 Core toolchain

- **Diagram generation**: QGRAF [1]
- **Symbolic manipulation and code generation**: FORM [2,3]
- **Topology discovery and integral processing**: MATHEMATICA with FEYNCALC [4]
- **IBP reduction**: BLADE (Mathematica package) [10]
- **Finite-field reconstruction and linear relations**: FINITEFLOW [5,6]

2.2 Algebra backends used by Mathematica scripts

- **Partial fractioning / multivariate rational simplification**: MULTIVARIATEAPART [9]
- **Rational algebra backend**: FERMAT [7]
- **Gröbner basis / polynomial computations**: SINGULAR [8]

2.3 Optional / environment-dependent

Depending on your setup, your IBP stage may also integrate with additional tooling (e.g. KIRA). If present, cite accordingly [11].

3 Architecture Overview

3.1 Run-based workflow

All computations occur in a **run directory**:

```
runs/{tag}_{nnnn}/
```

A run is a self-contained snapshot of a calculation: inputs, generated drivers, intermediate products, and logs.

3.2 Key idea: deterministic outputs

If you re-run a command inside the same run directory, GLAS reads the run metadata and uses the same directory conventions to locate inputs and write outputs.

3.3 The three-phase pipeline

GLAS structures the computation into three operational phases:

1. **Generation** (`generate`): create diagrams and prepare a run skeleton using QGRAF.
2. **Evaluation** (`evaluate`, `contract`, `uvct`): apply rules and build symbolic expressions via FORM.
3. **Reduction and simplification** (`extract topologies`, `ibp`, `reduce`, `micoef`, `linrels`, `ratcombine`): extract integrals/topologies in Mathematica, run IBP reduction, and simplify coefficients using finite fields.

4 Project Layout

A typical GLAS installation follows:

```
glas.py
glaslib/
resources/
  formlib/procedures/
  diagrams/
mathematica/scripts/
runs/
```

A typical run contains:

```
runs/{tag}_{nnnn}/
  meta.json
  diagrams/{0l,1l}/
  form/Files/...
  Mathematica/...
  logs/{command}/...
```

4.1 The `meta.json` contract

Each run has a `meta.json` tracking:

- process definition (tokens, tag),
- diagram counts at each loop order,

- kinematic configuration (Mandelstam definition),
- job configuration (requested/effective workers),
- gluon polarization reference momenta,
- number of master integrals discovered after IBP.

5 Quickstart Tutorial

This section is meant to be copy-paste friendly.

5.1 Important note: the `qQtT` run already exists

Your repository already contains a prepared run for the `qQtT` process inside `runs/`. That means you can start by attaching to it without generating anything:

```
glas> runs qQtT
glas> use qQtT_0001
```

(If the exact suffix differs, use `runs qQtT` to list available runs and pick the one present in your local checkout.)

5.2 Standard end-to-end workflow (REPL)

A typical full sequence looks like:

```
glas> verbose on

# 1) Evaluate amplitudes (tree + one-loop)
glas> evaluate lo --jobs 8
glas> evaluate nlo --jobs 8 --dirac

# 2) Contract: |M0|^2 and 2Re(M0*M1)
glas> contract lo --jobs 4
glas> contract nlo --jobs 4

# 3) UV counterterms (renormalization ingredients)
glas> uvct

# 4) Extract and map loop topologies (Mathematica + FORM formatting stage)
glas> extract topologies

# 5) IBP reduction (Blade + auxiliary algebra backends)
glas> ibp

# 6) Apply reduction rules to contracted expressions
glas> reduce --jobs 4

# 7) Extract master-integral coefficients
glas> micoef --jobs 4

# 8) Find linear relations over finite fields
glas> linrels

# 9) Combine rational functions into a minimal basis
glas> ratcombine
```

5.3 What you should expect after each step

This is a practical "sanity checklist":

- **After `evaluate`**: amplitude fragments under `form/Files/Amps/`.
- **After `contract`**: interference and squared objects under `form/Files/` (mode-dependent).
- **After `extract topologies`**: topology mapping outputs and integral rule headers usable by FORM.
- **After `ibp`**: reduction rule files (Mathematica + FORM headers) per topology.
- **After `reduce`**: reduced expressions under `form/Files/Reduced/`.
- **After `micoef`**: per-master coefficient files under `Mathematica/Files/MasterCoefficients/`.
- **After `linrels/ratcombine`**: simplified relations/basis used to compress the final representation.

6 Commands (Operational Reference)

6.1 Run navigation

```
glas> runs [tag]
glas> use {tag}           # attach to latest matching
glas> use {tag}_{nnnn}    # attach to a specific run
glas> show
```

6.2 Generation and preparation

```
glas> generate g g > t t~ --jobs 8
```

This initializes a new run and invokes QGRAF [1]. It also prepares FORM driver scaffolding in the run.

6.3 Evaluation and contraction

```
glas> evaluate lo  --jobs K
glas> evaluate nlo --jobs K --dirac
glas> contract lo  --jobs K
glas> contract nlo --jobs K
```

These steps generate and execute FORM drivers [2, 3]. The `-dirac` flag enables additional algebraic simplification inside the symbolic stage.

6.4 Gluon polarization references

Some contractions require polarization reference momenta for external gluons. GLAS stores these in `meta.json` and can prompt when missing:

```
glas> setrefs
```

6.5 Topology extraction and IBP

Topology extraction uses Mathematica + FeynCalc [4] to identify and map loop integrals onto completed topologies:

```
glas> extract topologies
```

IBP reduction is performed by Blade [10], and uses algebra backends such as Fermat [7], Singular [8], and MultivariateApart [9] for simplification:

```
glas> ibp
```

6.6 Coefficient extraction and finite-field simplification

```
glas> reduce --jobs K
glas> micoef --jobs K
glas> linrels
glas> ratcombine
```

Linear relations and basis selection are done using FiniteFlow [5, 6], which is the standard approach for large-scale rational reconstruction tasks in modern amplitude pipelines.

7 Logging, Debugging, and Reproducibility

7.1 Verbose streaming

`verbose on` streams subprocess output in real time with informative prefixes (e.g. `[form ...]`, `[mma ...]`, `[py ...]`). Regardless of streaming mode, logs are written to `runs/{run}/logs/`.

7.2 Where to look when something fails

- **FORM failures**: check `form_*.stdout.log` and `form_*.stderr.log` in the run.
- **Mathematica failures**: check the corresponding `logs/` stage file and the script output.
- **External tool paths**: confirm your environment variables (e.g. `FERMATPATH`, `SINGULARPATH`).

7.3 Reproducibility note

Everything required to reproduce the pipeline (inputs, intermediate products, drivers, metadata) remains in the run directory. This is intentional: it makes post-mortem debugging and result verification straightforward.

8 Extending GLAS (Developer Notes)

8.1 Adding a new command

The recommended pattern is:

1. implement the command module under `glaslib/commands/`,
2. register it in the REPL shell,
3. ensure it reads/writes `meta.json` rather than storing state elsewhere,
4. follow the existing parallel execution pattern to generate drivers and capture logs.

8.2 Do not hardcode absolute paths

All paths should be derived from the run context and project root helpers. This is required for portability and reproducibility.

9 Acknowledgments

This project is built on top of widely-used community tools: QGRAF [1], FORM [2,3], FeynCalc [4], FiniteFlow [5,6], and algebra systems including Fermat [7] and Singular [8]. We also acknowledge MultivariateApart [9] and the IBP reduction tooling (Blade) [10] used in the reduction stage.

References

[1] P. Nogueira, "Automatic Feynman graph generation," J. Comput. Phys. **105** (1993) 279.

[2] J. A. M. Vermaseren, "New features of FORM," arXiv:math-ph/0010025.

[3] J. Kuipers, T. Ueda, J. A. M. Vermaseren and J. Vollinga, "FORM version 4.0," Comput. Phys. Commun. **184** (2013) 1453.

[4] V. Shtabovenko, R. Mertig and F. Orellana, "FeynCalc 9.3: New features and improvements," Comput. Phys. Commun. **256** (2020) 107478.

[5] T. Peraro, "Scattering amplitudes over finite fields and multivariate functional reconstruction," JHEP **12** (2016) 030.

[6] T. Peraro, "FiniteFlow: multivariate functional reconstruction using finite fields and dataflow graphs," JHEP **07** (2019) 031.

[7] R. H. Lewis, "Fermat: a computer algebra system for polynomial and matrix computation," (Software documentation and distribution).

[8] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "Singular 4-0-2 — A computer algebra system for polynomial computations," https://www.singular.uni-kl.de.

[9] M. Heller and A. von Manteuffel, "MultivariateApart — multivariate partial fraction decomposition," (Software documentation and distribution; bundled in this repository).

[10] "Blade — IBP reduction package for Mathematica," (Software documentation and distribution; used by GLAS in the IBP stage).

[11] P. Maierhöfer, J. Usovitsch and P. Uwer, "Kira — A Feynman integral reduction program," Comput. Phys. Commun. **230** (2018) 99.