AD FALCON API Manual

Postprocessing Outputs (Files, Formats, and Controls)

Javad Ghorbani

March 26, 2026

Contents

1 Postprocessing Outputs (Files, Formats, and Controls) 3

1.1 1) What gets written (overview) . . . 3
1.1.1 Field outputs (time series) . . . 3
1.1.2 Targeted CSV outputs . . . 3
1.2 2) Output schedule (frequency) . . . 4
1.2.1 Step-based vs time-based output . . . 4
1.2.2 CSV output frequencies . . . 4
1.3 3) Requesting variables (syntax) . . . 4
1.4 4) GenericXDMF output (`falcon.xdmf` + binary payloads) . . . 5
1.4.1 Output directory . . . 5
1.4.2 Files written and on-disk data layout . . . 5
1.4.3 Parsing in external scripts . . . 6
1.5 5) GiD Postprocess export: `.post.msh` (mesh) and `.post.res` (results) . . . 6
1.5.1 File naming and splitting . . . 6
1.5.2 `.post.msh` data structure (mesh) . . . 7
1.5.3 `.post.res` data structure (results) . . . 7
1.5.4 Result-name mapping (what to search for in `.post.res`) . . . 8
1.5.5 Parsing `.post.res` in external scripts . . . 8
1.6 6) ParaView/VTK export: VTK legacy `.vtk` files . . . 9
1.6.1 Output directory . . . 9
1.6.2 Data structure . . . 9
1.7 7) CSV postprocessing outputs (data structure + cadence) . . . 9
1.7.1 `% DOFOutput` (CSV) . . . 9
1.7.2 `% ReactionForceSum` (CSV) . . . 10
1.7.3 `% MasterForceContact` (CSV) . . . 10
1.7.4 `% PointStateOutput` / `% LineStateOutput` (CSV) . . . 10
1.7.5 `% PicardLoopOutput` (CSV) . . . 10

1 Postprocessing Outputs (Files, Formats, and Controls)

This page documents FALCON's postprocessing outputs as written by the solver, including:

- **Generic XDMF output** (recommended for external scripts/tools): `falcon.xdmf` plus binary payload files (RAW `.bin` or HDF5, depending on build)
- **GiD Postprocess export**: `.post.msh` (mesh) and `.post.res` (results)
- **ParaView/VTK export**: VTK legacy `.vtk` files (and an XDMF time series via **Generic XDMF output**, which ParaView can also open)
- **CSV outputs** for targeted queries (reaction sums, node histories, contact forces, etc.)

It focuses on **data structure** (so you can parse outputs with external scripts), **output frequency**, and **syntax** for requesting variables.

For path configuration (CLI + `FALCON.env`), see CLI Paths & Working Directory.

1.1 1) What gets written (overview)

1.1.1 Field outputs (time series)

Field outputs are controlled **per step** in `% Step Definitions`:

Where	Key(s)	What it controls	Notes
`@Step <id>`	`@@OutputTypes:`	**Which variables** to write (per output frame)	Space/comma/semicolon separated list; see Output Variables.
`@Step <id>`	`@@OutputControlType:` + `@@OutputControlValue:`	**When** to write frames (output frequency)	`ByStep` (every N substeps) or `ByTime` (every Δt).
`@Step <id>`	`@@PostprocessTool:` / `@@PostprocessOutputs:`	**Which file formats** to write	One-or-more outputs (comma/space/semicolon separated): `GiD`, `ParaView/VTK`, `Generic XDMF`.

1.1.2 Targeted CSV outputs

These outputs are configured in their own sections and are intended to be easy to postprocess in scripts:

- `% MasterForceContact` (CSV)
- `% ReactionForceSum` (CSV)

- `% DOFOutput` (CSV)
- `% PointStateOutput` / `% LineStateOutput` (CSV)
- `% PicardLoopOutput` (CSV)

1.2 2) Output schedule (frequency)

1.2.1 Step-based vs time-based output

Inside a `@Step` block:

```
@@OutputControlType: ByStep
@@OutputControlValue: 10
```

- `ByStep`: write every **N converged substeps** within that step.
- `ByTime`: write every Δ**t** of analysis time (the solver emits frames when the running time crosses multiples of `OutputControlValue`).

  Legacy shorthand:

```
@@OutputInterval: 10
```

  This is equivalent to `ByStep` with `OutputControlValue = 10`.

1.2.2 CSV output frequencies

CSV-style outputs each define their own cadence:

- `% DOFOutput`: `@Every` / `@Frequency` (every N converged substeps)
- `% ReactionForceSum`: `@Steps` + `@Frequency` (every N converged substeps within selected steps)
- Other CSV outputs define their own trigger rules; see their dedicated sections/pages.

1.3 3) Requesting variables (syntax)

Variables are selected per step:

```
% Step Definitions
@Step 1:
  @@OutputTypes: Displacement EffStress PW
%%%
```

Rules:

- Tokens are **space/comma/semicolon-separated**.
- Built-in output types are matched **case-insensitively**.
- **Custom variable names** (UMAT custom state variables) are matched by name; keep the exact spelling you used when registering them in the material.

The supported output variables are listed in Output Variables. Some variables are format-specific:

- "VTK export only" variables are written only when `@@PostprocessTool: ParaView / VTK`.
- "GenericXDMF export only" variables are written only when `@@PostprocessTool: GenericXDMF / XDMF / HDF5`.
- "GiD export only" variables (notably custom variable names) are written only when `@@ PostprocessTool: GiD`.

1.4 4) GenericXDMF output (`falcon.xdmf` + binary payloads)

Set the postprocess tool in `% Step Definitions`:

```
@@PostprocessTool: GenericXDMF
```

GenericXDMF is intended for **generic scripting and external postprocessors**. It writes an XDMF time series (`falcon.xdmf`) plus binary payload files in an output directory.

1.4.1 Output directory

XDMF output requires an output directory path configured from outside the input file (CLI or `FALCON.env`):

- CLI: `--xdmf-dir <dir>`
- `FALCON.env`: `XDMF_OUTPUT_DIR_RELATIVE_PATH=<dir>`

If the XDMF output directory is not set/usable, XDMF output is skipped.

1.4.2 Files written and on-disk data layout

The output directory contains:

- `falcon.xdmf` (XML time series that points to the binary payload files)
- A fixed mesh payload:
  - `Mesh_Coordinates.bin` – `float64` little-endian, shape `(numNodes, 3)` (XYZ)
  - `Mesh_Topology.bin` – `int64` little-endian, shape `(topologyLength,)` (XDMF "Mixed" topology stream)

- Per-frame payloads (one file per attribute per output frame), named:
  - `Step000000_<Attribute>.bin`, `Step000001_<Attribute>.bin`, ...

  For example:

- `Step000010_Displacement.bin` – `float64` little-endian, shape `(numNodes, 3)`
- `Step000010_PW.bin` – `float64` little-endian, shape `(numNodes,)`

If the solver is built with HDF5 support enabled, the payloads are stored in `falcon.h5` instead, and `falcon.xdmf` references the HDF5 datasets. The XDMF "view" (`falcon.xdmf`) remains the entry point.

1.4.3 Parsing in external scripts

The recommended approach is:

1. Read `falcon.xdmf` to determine `numNodes`, topology length, and which per-frame attributes exist.
2. Load the referenced binary files as little-endian `float64` (coordinates/attributes) or `int64` (topology) and reshape according to the dimensions in the XDMF.

1.5 5) GiD Postprocess export: `.post.msh` (mesh) and `.post.res` (results)

1.5.1 File naming and splitting

By default FALCON writes one mesh and one results file:

- `<base>.post.msh`
- `<base>.post.res`

When GiD splitting is enabled (for step segmentation and/or element-activity changes), FALCON writes "segments":

- `<base>.segN.fromStepX.post.msh`
- `<base>.segN.fromStepX.post.res`

These are controlled by GiD-related keys in `% Step Definitions` (see Steps and Activation and Deactivation of Elements):

- `@@GIDSplitPerStep: Yes`
- `@@GIDSplitOnElementActivity: Yes`
- `@@GIDStartNewFiles: Yes` (per-step)
- `@@GIDOmitInactiveElements: Yes`
- `@@GIDExcludeElements: ...`
- `@@GIDOutputFormat: Ascii|Binary` (Binary currently falls back to ASCII)

1.5.2 `.post.msh` data structure (mesh)

FALCON writes the GiD “post mesh” in ASCII. Conceptually it contains:

1. A `MESH ...` header describing dimension and element topology.
2. A `Coordinates` block: node IDs and coordinates.
3. An `Elements` block: element IDs, connectivity, and (typically) a material/group ID.

Minimal shape (illustrative):

```
MESH "mesh" dimension 3 ElemType Tetrahedra Nnode 4
Coordinates
  1  0.0  0.0  0.0
  2  1.0  0.0  0.0
End Coordinates
Elements
  1  1 2 3 4  <groupId>
End Elements
```

If you are scripting, you can usually rely on node/element IDs matching your input file, and parse `.post.msh` only when you need the written connectivity (for example when element exclusion/omission is enabled).

1.5.3 `.post.res` data structure (results)

The GiD results file is an ASCII time series consisting of repeated **result blocks**.

The file begins with a header line:

```
GiD Post Results File 1.0
```

Each block has:

1. A `Result ...` header line (result name + time + kind).
2. A `Values` section listing values per node.
3. An `End Values` terminator.

Typical shape (illustrative):

```
Result "Displacement" "Isochrones" 0.0 Vector OnNodes
Values
  1  0.0  0.0  0.0
  2  1.2e-6  0.0  0.0
End Values
```

Important scripting notes:

- Result blocks are typically **on nodes** (nodal fields). Your script should not assume a fixed ordering of result blocks.
- The second quoted string in the `Result ...` header is a label (currently written as `Isochrones`); scripts should not rely on it.
- Different steps can output different `@@OutputTypes`, so a result may be present at some times and absent at others.
- The time value in the `Result ...` header is the analysis time written for that frame.

1.5.4 Result-name mapping (what to search for in `.post.res`)

When parsing `.post.res`, you typically search by the **Result name** (the first quoted string in `Result "..." ...`). For common built-in outputs, the names written to `.post.res` are:

`@@OutputTypes:` token	Result name in `.post.res`
`Displacement`	`Displacement`
`ReactionForce`	`Reaction Force`
`EffStress`	`EffStress`
`NetStress`	`NetStress`
`TotalStress`	`TotalStress`
`Strain`	`Strain`
`VoidRatio`	`Void Ratio`
`PW`	`Pore Water`
`EXPW`	`Excess Pore Water`
`PA`	`Pore Air`
`EXPA`	`Excess Pore Air`
`Sw`	`Saturation Degree`
`PC`	`Suction`
`PermW`	`Permeability of Water`
`PermA`	`Permeability of Air`
`Alpha_p_c`	`Alpha_p_c`
`PWHead`	`PWHead`
`PAHead`	`PAHead`

For UMAT custom state variables (GiD-only), the result name is typically the same token you list in `@@OutputTypes:`.

1.5.5 Parsing `.post.res` in external scripts

If you just need one variable at one time, you can scan `Result ...` blocks and read values inside `Values ... End Values`.

1.6 6) ParaView/VTK export: VTK legacy `.vtk` files

Set the postprocess tool in `% Step Definitions`:

```
@@PostprocessTool: ParaView
```

When `@@PostprocessTool: ParaView` (alias `VTK`) is used, FALCON writes **VTK legacy unstructured grid** files (`.vtk`). For ParaView workflows that prefer XDMF time-series, see **Generic XDMF output** (section 4), which writes `falcon.xdmf` plus binary payloads.

1.6.1 Output directory

VTK output requires an output directory path configured from outside the input file (CLI or `FALCON.env`):

- CLI: `--vtk-dir <dir>`
- `FALCON.env`: `VTK_OUTPUT_DIR_RELATIVE_PATH=<dir>`

If the VTK output directory is not set, VTK output is skipped.

1.6.2 Data structure

VTK output is intended to be readable by ParaView and by scripts that parse VTK legacy files. The file header begins with:

```
# vtk DataFile Version 3.0
```

FALCON writes one `.vtk` file per output frame into the configured directory. File names are implementation-defined; scripts should enumerate `*.vtk` in the directory rather than relying on a specific naming pattern.

The fields written follow the same `@@OutputTypes` selection rules, with the additional constraint that some variables are “VTK export only” (see Output Variables).

1.7 7) CSV postprocessing outputs (data structure + cadence)

These outputs are designed to be easy to read in Python/Matlab/R:

1.7.1 `% DOFOutput` (CSV)

- Purpose: time history of selected DOFs at selected nodes.
- Cadence: every `@Every` (or `@Frequency`) converged substeps.
- Format: one CSV row per write; first column is `Time`, followed by one column per node/DOF.

See: Special Outputs.

1.7.2 % ReactionForceSum (CSV)

- Purpose: sum reaction forces over a node set, and record per-node DOF values.
- Cadence: within selected `@Steps`, write every `@Frequency` converged substeps.
- Format: columns include `StepID`, `Time`, summed reactions per requested DOF, and per-node DOF values.

See: Special Outputs.

1.7.3 % MasterForceContact (CSV)

- Purpose: contact force reporting for selected contact pairs.
- Cadence and columns: defined by the contact output section.

See: Contact Mechanics in FALCON.

1.7.4 % PointStateOutput / % LineStateOutput (CSV)

- Purpose: sample state variables at arbitrary coordinates/lines with step/time triggering.
- Data structure: CSV with coordinates + requested variable columns (depends on mode).

See: Point & Line State Output.

1.7.5 % PicardLoopOutput (CSV)

- Purpose: diagnostic CSV for Picard-loop initialization (when executed).
- Data structure: per Gauss point statistics and convergence metrics.

See: Special Outputs.