AD FALCON API Manual

Include and Modular Input Files

Javad Ghorbani

March 26, 2026

Contents

1 Include and Modular Input Files

FALCON supports modular input files via **Include directives**. These allow you to split a large model into smaller files (for example, geometry, materials, and boundary conditions) and stitch them together at parse time.

At a high level, each Include directive behaves like a **copy–paste of the included file's contents** at the point where the directive appears, with extra safety checks for paths and cycles.

1.1 Syntax

Include directives are **single-line commands**, not full sections, and do **not** use %%% delimiters.

- **Percent-style Include** (commonly used between sections):

```
% Include path/to/other_input.txt
```

- **At-style Include** (parsed identically, useful if you prefer the @ marker):

```
@Include path/to/other_input.txt
```

Both forms are treated the same by the input reader:

- The line must start with either `% Include` or `@Include`.
- Anything after `Include` on that line is interpreted as the **path argument** (after trimming whitespace and optional quotes).

1.1.1 Paths with spaces and quoting

You can use either raw paths or quotes:

```
% Include subdir/materials.txt
% Include "path with spaces/bcs_stage1.txt"
@Include 'configs/contact setup.txt'
```

- Leading/trailing whitespace around the path is ignored.
- If the path is wrapped in single (`'...'`) or double (`"..."`) quotes, the quotes are stripped before resolution.

1.2 Path Resolution Rules

Include directives follow these rules for resolving file paths:

- **Relative paths** are resolved with respect to the **directory of the file that contains the Include directive**, not the working directory where FALCON is launched.
  - Example: if `/models/main.txt` contains `"% Include sub/mesh.txt"`, then `sub/mesh.txt` is resolved as `/models/sub/mesh.txt`.
- **Absolute paths** are used as-is.
- The resolved path is then **normalized** (using `std::filesystem::weakly_canonical` when possible) before being used for cycle detection and file opening.

If the included file cannot be opened, you will see a `DataFormatException` with a message similar to **`[IR-0009] Failed to open included file: <path>`**.

1.3 Recursion, Cycles, and Safety

Include directives are processed **recursively**:

- When the input system encounters an Include directive in a file, it:
  1. Resolves the target path as described above.
  2. Opens that file.
  3. Parses it using the same logic, so included files can themselves contain further Include directives.

To prevent infinite loops and unintended duplication, the reader keeps a **global set of already-included absolute paths**:

- If a file is about to be included and its normalized absolute path has **already been seen**, you will see a `DataFormatException` with a message similar to **`[IR-0008] Cyclic Include detected: <path>`**.
- Practically, this means:
  - **Each physical file may be included at most once** in the expanded input graph.
  - Any attempt—direct or indirect—to re-include the same file is treated as a cycle and rejected.

> **Tip:** If you need the same logical content in multiple places, duplicate the content into a second file rather than including the same file twice.

1.4 Interaction with Core Section Ordering

Before the main parse, the input system runs a **pre-parse sanity check** on the root input file:

- It scans the main file **and all included files** reachable via `% Include` / `@Include`.
- It records the order of the following **core sections** as they appear across the entire expanded input:

  1. `% AnalysisType`
  2. `% Nodes`
  3. `% Materials`
  4. `% Elements`
  5. `% Step Definitions`

If, across all files, these appear out of order (for example, `% Elements` before `% Materials` in any combination of main and included files), you will see a `DataFormatException` with a message similar to **`[IR-0012]`**, explaining:

- Which file and line triggered the problem, and
- What the expected global ordering of core sections is.

In practice:

- You are free to place Include directives **before, between, or after sections**, but
- The **combined sequence** of all core-section markers (in the root file plus all includes) must respect the required order.

1.5 Where Include Directives Are Allowed

Include directives are recognized in the **top-level input stream** handled by `InputReader::readInput`:

- They are intended to appear **between section blocks**, not inside the bodies of `% Materials`, `% Elements`, etc.
- A typical pattern is:

```
% AnalysisType
PLCoupled
%%%

% Include geometry/mesh.txt
% Include materials/soil_library.txt

% Step Definitions
```

```
@Step 1
...
%%%
```

This structure keeps each logical part of the model in a separate file while preserving the required core-section ordering.

1.6 Usage Patterns and Best Practices

- **Split by concern**:
  - `mesh_geometry.txt` for `% Nodes` and `% Elements`
  - `materials.txt` for `% Materials`
  - `bc_stage1.txt`, `bc_stage2.txt` for loading and boundary conditions
- **One owner per core section**:
  - Avoid defining the **same core section** (e.g., `% Materials`) in multiple files unless you are certain of their merged ordering.
- **Stable paths**:
  - Prefer **relative paths** within a project folder so that the model is portable across machines.
- **No duplication via Include**:
  - Remember that each file can only be included once; if you need reused content, either:
    * Factor it into smaller, dedicated include files, or
    * Copy the needed lines into multiple files.