AD FALCON API Manual

Checkpoint & Restart System

Javad Ghorbani

March 26, 2026

Contents

1 Checkpoint & Restart System

FALCON provides a comprehensive checkpoint/restart system that enables saving and resuming simulation state. This functionality is essential for:

- **Long-running simulations**: Save progress periodically to guard against crashes or system failures
- **Parametric studies**: Start multiple analyses from a common converged state
- **Resource management**: Split simulations across job scheduler time limits

---

1.1 Overview

The checkpoint system saves the **numerical simulation state** to binary files (`.adfalcon` extension), including:

Saved Data	Description
Mesh topology	Nodes, elements, connectivity, per-element material IDs/types
Displacement fields	Current and incremental displacement (`U`, `DeltaU`)
Velocity & acceleration	`DU_DT`, `D2U_DT2`, `linearVel`, `linearACC`
Force vectors	External forces (`FEXT`, `globalExtForce`), internal forces (`F_internal`), saved forces (`FSAVE`)
Stress/strain state	All Gauss point stress tensors and state variables
Pore pressures	Water and air pressure forces (`GlobalPw Force`, `GlobalPaForce`)
Internal variables	Void ratio, plastic strain, hardening parameters at Gauss points
Solver state	Global stiffness (`globalK`), mass (`globalM`), coupling matrices, DOF mappings
Time integration	Current time, step ID, sub-step counter, `dt`, remaining step time
Boundary conditions	Load factors, stress boundaries, body forces, prescribed values
Session state	Auto-increment time stepping state (dt history, success streaks)

Saved Data	Description
Contact state	Contact pairs + contact state (including Augmented Lagrangian state when enabled) and rigid bodies
Element-node adjacency	`ej` map for efficient neighbor lookups

File format features:

- Binary format for efficient storage and fast I/O
- CRC64 checksum for data integrity verification
- Version tracking for backward compatibility
- Automatic checkpoint rotation (keep N most recent)
- Seamless mid-step restart support

!!! note "What is NOT stored" Some configuration is intentionally expected to come from the input file on restart: - Full material model definitions (polymorphic `MaterialProperties` instances) - UMAT shared libraries/handlers and compiled function pointers - Shape-function objects and some per-element cached data (recreated after restart) - Full simulation-step configuration (solver controls, auto-increment tuning, output directives). The checkpoint contains only a minimal subset needed for stream alignment; the input file remains the source of truth.

1.2 Syntax

Checkpointing and restart are configured with two optional sections:

```
% CheckpointControl
  KEY = VALUE
  ...
%%%

% RestartFrom
  KEY = VALUE
  ...
%%%
```

Notes:

- Keys are case-insensitive (for example, `ENABLED`, `Enabled`, and `enabled` are treated the same).
- Whitespace around `=` is optional (`ENABLED=ON` and `ENABLED = ON` both work).

- Boolean values accept common forms such as `ON/OFF`, `Yes/No`, `True/False`, and `1/0` (case-insensitive).
- Unknown keys are ignored with a warning.

1.2.1 Multiple checkpoint/restart sections

- If `% CheckpointControl` appears multiple times, the last one takes effect.
- If `% RestartFrom` appears multiple times, the last one takes effect.

1.3 Enabling Checkpoints

Configure checkpoint saving via the `% CheckpointControl` section in your input file.

1.3.1 Section Placement

```
% CheckpointControl
  ENABLED = ON
  OUTPUT_PATH = ./checkpoints/simulation
  SAVE_FREQUENCY = 10
  SAVE_AT_STEP_END = ON
  SAVE_AT_SUBSTEP = OFF
  MAX_CHECKPOINTS = 5
  VALIDATE_ON_RESTART = ON  # currently unused
  VERBOSE_LOGGING = ON
%%%
```

1.3.2 Parameters

Table 1: Checkpoint configuration parameters

Parameter	Required?	Default	Type	Description
`ENABLED`	Optional	`OFF`	`ON/OFF`	Master switch to enable checkpoint saving
`OUTPUT_PATH`	Optional	`./checkpoints/sim`	`string`	Base path for checkpoint files (without extension)

Parameter	Required?	Default	Type	Description
SAVE_FREQUENCY	Optional	0	int	Save every N sub-steps (0 = disabled). Uses the global sub-step counter printed in diagnostics and filenames.
SAVE_AT_STEP_END	Optional	ON	ON/OFF	Save checkpoint at end of each analysis step
SAVE_AT_SUBSTEP	Optional	OFF	ON/OFF	Save at every sub-step (overrides SAVE_FREQUENCY)
MAX_CHECKPOINTS	Optional	3	int	Maximum number of checkpoint files to retain
VALIDATE_ON_RESTART	Optional	ON	ON/OFF	Parsed but currently unused. Restart validation is controlled by VALIDATE_MESH in % RestartFrom.
VERBOSE_LOGGING	Optional	ON	ON/OFF	Print checkpoint status messages to console

!!! note "Checkpoint File Naming" Checkpoint files are automatically named using the pattern: <OUTPUT_PATH>_step<StepID>_sub<SubstepID>.adfalcon For example: ./checkpoints/simulation_step2_sub150.adfalcon

!!! note "Path Resolution" OUTPUT_PATH is resolved relative to FALCON's working directory (and stored internally as an absolute path). FALCON creates the parent directory if needed. If the location is not writable, FALCON falls back to ./checkpoints/sim under the working directory.

1.4 Restarting from Checkpoint

To resume a simulation from a checkpoint, use the % RestartFrom section.

1.4.1 Section Placement

```
% RestartFrom
  FILE = ./checkpoints/simulation_step2_sub150.adfalcon
  VALIDATE_MESH = ON
  TARGET_STEP = -1
%%%
```

1.4.2 Parameters

Table 2: Restart configuration parameters

Parameter	Required?	Default	Type	Description
`FILE`	**Yes**	—	`string`	Path to checkpoint file to load
`VALIDATE_MESH`	Optional	`ON`	`ON/OFF`	Validate the input mesh against the checkpoint (dimensions + fingerprint when available)
`TARGET_STEP`	Optional	`-1`	`int`	Reserved (not implemented); select the desired `.adfalcon` file explicitly via `FILE`

!!! warning "Input File Consistency" When restarting, ensure your input file matches the checkpoint:

- Same mesh definition (nodes, elements)
  - Same material parameters (or intentional changes)
  - Compatible step definitions

The system can validate mesh geometry, topology, material configuration, analysis type, time-integration settings, PML configuration, and infinite-element flags via fingerprint hashing (when `VALIDATE_MESH = ON`).

!!! note "Restart File Path Resolution" `FILE` may be relative or absolute. If the provided path is not readable, FALCON attempts a working-directory fallback resolution.

1.4.3 What Happens on Restart

When FALCON loads a checkpoint, it restores the saved numerical state and then determines how to resume the step loop:

- **Step-end checkpoint**: If the saved time is at (or beyond) the end of the saved step, FALCON treats that step as complete and resumes from the **next** step.
- **Mid-step checkpoint**: If the saved time lies inside a step (or the checkpoint stores a nonzero remaining step time), FALCON resumes **within the same step** and **skips step-start initialization** for that step to avoid double-applying one-time actions.

Step-start initialization includes (when present in your input):

- Element activation/deactivation scheduled for the step
- `% Step Initial Assignments` (applied at step start; overwrites the current Gauss-point state)
- Add/Release DOF boundaries
- Tie/untie constraints scheduled per step
- Step-boundary actions (e.g., post-step equilibrium/reset actions)
- Output segmentation actions (for example, starting a new GiD segment)

Time stepping:

- The checkpoint restores the current `dt` and (when enabled) the auto-increment controller state.
- For mid-step restarts, the remaining step time is restored when available so the solver continues toward the original step end time.

Output on restart:

- With GiD output enabled, FALCON starts fresh `.msh/.res` files on restart to avoid corruption from mixing output across runs.
- With VTK output enabled, FALCON refreshes the VTK output directory on restart so time series do not mix.

In typical usage (same executable, same input, same platform), the restart is intended to be **bit-exact**. If you change the input configuration, material/UMAT code, compiler, or platform, results may differ.

1.5 Checkpointing Strategies

1.5.1 Step-End Checkpointing (Recommended for Most Cases)

Save checkpoints only at the end of each analysis step:

```
% CheckpointControl
  ENABLED = ON
  OUTPUT_PATH = ./checkpoints/simulation
  SAVE_AT_STEP_END = ON
```

```
  SAVE_AT_SUBSTEP = OFF
  MAX_CHECKPOINTS = 3
%%%
```

Advantages:

- Low overhead (infrequent saves)
- Natural restart points at step boundaries
- Suitable for most production runs

1.5.2 Frequency-Based Checkpointing

Save every N sub-steps for fine-grained recovery:

```
% CheckpointControl
  ENABLED = ON
  OUTPUT_PATH = ./checkpoints/simulation
  SAVE_FREQUENCY = 50
  SAVE_AT_SUBSTEP = OFF
  MAX_CHECKPOINTS = 5
%%%
```

Use when:

- Running very long steps with many sub-steps
- Using automatic time stepping (`ModernAutoInc: Yes`)
- System instability concerns require frequent saves

1.5.3 Minimal Checkpointing (Final State Only)

Save only at the very end:

```
% CheckpointControl
  ENABLED = ON
  OUTPUT_PATH = ./results/final_state
  SAVE_FREQUENCY = 0
  SAVE_AT_STEP_END = ON
  MAX_CHECKPOINTS = 1
%%%
```

1.5.4 HPC/Cluster Considerations

For cluster jobs with time limits, save frequently enough to avoid losing significant progress:

```
% CheckpointControl
  ENABLED = ON
  OUTPUT_PATH = /scratch/$USER/job123/checkpoint
  SAVE_FREQUENCY = 100
  SAVE_AT_SUBSTEP = OFF
  SAVE_AT_STEP_END = ON
  MAX_CHECKPOINTS = 3
  VERBOSE_LOGGING = OFF
%%%
```

Tips:

- Use fast local storage (e.g., `/scratch`) for checkpoints
- Set `MAX_CHECKPOINTS = 2-3` to limit disk usage
- Disable verbose logging in batch jobs (`VERBOSE_LOGGING = OFF`)
- Copy final checkpoint to permanent storage post-job

1.6 Complete Examples

1.6.1 Example 1: Standard Checkpointing

A typical setup for a multi-step consolidation analysis:

```
% CheckpointControl
  ENABLED = ON
  OUTPUT_PATH = ./checkpoints/consolidation
  SAVE_FREQUENCY = 0
  SAVE_AT_STEP_END = ON
  SAVE_AT_SUBSTEP = OFF
  MAX_CHECKPOINTS = 5
  VALIDATE_ON_RESTART = ON  # currently unused
  VERBOSE_LOGGING = ON
%%%

% Step Definitions
@Step 1:
  @@SolverType: Direct
  @@StartStep: 0
  @@StepTime: 10000
  @@SimMode: Consolidation
  @@ModernAutoInc: Yes
  @@InitialStepIncrement: 100
  @@MinTimeStep: 1
```

```
  @@MaxTimeStep: 500
  @@ErrorTarget: 1e-3
%%%
```

Checkpoint files created:

```
./checkpoints/consolidation_step1_sub85.adfalcon
./checkpoints/consolidation_latest.adfalcon  (copy of most recent)
```

1.6.2 Example 2: Restart a Failed Analysis

After a crash or timeout, restart from the last checkpoint:

```
% RestartFrom
  FILE = ./checkpoints/consolidation_step1_sub47.adfalcon
  VALIDATE_MESH = ON
%%%

% CheckpointControl
  ENABLED = ON
  OUTPUT_PATH = ./checkpoints/consolidation
  SAVE_AT_STEP_END = ON
  MAX_CHECKPOINTS = 5
%%%

% Step Definitions
@Step 1:
  @@SolverType: Direct
  @@StartStep: 0
  @@StepTime: 10000
  @@SimMode: Consolidation
  @@ModernAutoInc: Yes
%%%
```

The simulation resumes from sub-step 47 with the exact `dt` and auto-increment state preserved.

1.6.3 Example 3: Multi-Stage Analysis with Parametric Variation

Run geostatic initialization once, then branch into multiple analyses:

Stage 1: Geostatic (run once)

```
% CheckpointControl
  ENABLED = ON
```

```
  OUTPUT_PATH = ./baseline/geostatic
  SAVE_AT_STEP_END = ON
  MAX_CHECKPOINTS = 1
%%%

% Step Definitions
@Step 1:
  @@Geostatic: Yes
  @@SolverType: Direct
  @@StartStep: 0
  @@StepTime: 1
%%%
```

Stage 2a: Loading Case A (starts from geostatic)

```
% RestartFrom
  FILE = ./baseline/geostatic_step1_sub10.adfalcon
%%%

% CheckpointControl
  ENABLED = ON
  OUTPUT_PATH = ./results/case_A
%%%

% Step Definitions
@Step 2:
  @@StartStep: 1
  @@StepTime: 50000
  ... (Case A loading)
%%%
```

Stage 2b: Loading Case B (also starts from geostatic)

```
% RestartFrom
  FILE = ./baseline/geostatic_step1_sub10.adfalcon
%%%

% CheckpointControl
  ENABLED = ON
  OUTPUT_PATH = ./results/case_B
%%%

% Step Definitions
```

```
@Step 2:
  @@StartStep: 1
  @@StepTime: 50000
  ... (Case B loading)
%%%
```

1.7 File Management

1.7.1 Checkpoint Rotation

When `MAX_CHECKPOINTS` is reached, the oldest checkpoint is automatically deleted. The rotation keeps only the N most recent files based on timestamps.

Note: rotation tracks checkpoints created in the current run; older `.adfalcon` files already in the directory are not automatically pruned.

Retained files:

```
simulation_step3_sub200.adfalcon  (most recent)
simulation_step3_sub150.adfalcon
simulation_step3_sub100.adfalcon
simulation_latest.adfalcon        (copy of most recent)
```

1.7.2 Storage Requirements

Checkpoint file size depends on mesh complexity:

Mesh Size	Typical Checkpoint Size
1,000 nodes	~1-5 MB
10,000 nodes	~10-50 MB
100,000 nodes	~100-500 MB
1,000,000 nodes	~1-5 GB

Factors affecting size:

- Number of Gauss points per element
- Number of custom state variables
- Coupled analysis (pore pressures add ~50%)
- Contact (additional contact state data)
- Sparse matrix storage

1.7.3 Inspecting Checkpoint Files

When restarting, FALCON prints restart status to the console. Typical output looks like:

```
CHECKPOINT: Attempting to restore from:
./checkpoints/simulation_step2_sub100.adfalcon
CHECKPOINT: Loading from ./checkpoints/simulation_step2_sub100.adfalcon
CHECKPOINT: Loaded successfully - Step 2, Substep 100, Time 5432.5
CHECKPOINT: Successfully restored simulation state
CHECKPOINT: Resuming from step index 1 (Step ID 2)
CHECKPOINT: Skipping step initialization (resuming mid-step at t=5432.5, ...)
```

If the checkpoint was taken at a step boundary, you may instead see a message indicating the saved step is already complete and FALCON is advancing to the next step.

```
CHECKPOINT: Step 2 is already complete at t=... (stepEnd=...). Advancing to next
step.
```

For deeper inspection, you can enable checkpoint debug logging via the `% Debug` section (`@Checkpoint:` path) to dump detailed state snapshots during save/load.

1.8 Error Handling

1.8.1 Checkpoint Result Codes

Code	Description	Action
`Success`	Operation completed successfully	Continue
`FileNotFound`	Checkpoint file does not exist	Check path
`InvalidMagic`	File is not a valid FALCON checkpoint	Verify file
`VersionMismatch`	Checkpoint version incompatible	Use compatible FALCON version
`Checksum Mismatch`	Data corruption detected	Re-run from earlier checkpoint
`MeshDimension Mismatch`	Node/element count differs from input	Fix input file
`MeshGeometry Mismatch`	Node coordinates differ from checkpoint	Fix input file / use matching mesh
`MeshTopology Mismatch`	Element connectivity differs from checkpoint	Fix input file / use matching mesh
`MaterialConfig Mismatch`	Element material IDs differ from checkpoint	Fix input file / use matching materials
`AnalysisType Mismatch`	Analysis type differs from checkpoint	Fix input file / use matching analysis type

Code	Description	Action
`TimeIntegrationMismatch`	Time integration settings differ from checkpoint (Implicit/Explicit/IMEX and related options)	Use matching step time-integration settings
`PmlConfigMismatch`	PML configuration differs from checkpoint	Use matching PML setup (or disable validation)
`InfiniteElementMismatch`	Infinite-element flags differ from checkpoint	Use matching % Infinite Elements configuration
`ReadError`	I/O error during load	Check permissions/disk
`WriteError`	I/O error during save	Check disk space/permissions
`ValidationFailed`	Validation constraints not satisfied	Review input consistency
`CompressionError`	Compression/decompression failed	Re-run / rebuild
`OutOfMemory`	Insufficient memory to load	Use machine with more RAM

1.8.2 Common Issues and Solutions

Issue: Checkpoint fails to save

```
[ERROR] CheckpointManager: WriteError - Cannot create directory
```

Solution: Ensure the output directory parent exists and has write permissions.

Issue: Restart fails with MeshDimensionMismatch

```
[ERROR] CheckpointManager: MeshDimensionMismatch - Expected 12847 nodes, found 12000
```

Solution: The input file mesh must match the checkpoint. Verify you're using the correct mesh definition.

Issue: ChecksumMismatch on load

```
[ERROR] CheckpointManager: ChecksumMismatch - Data integrity check failed
```

Solution: The checkpoint file is corrupted. Use an earlier checkpoint or re-run the analysis.

Issue: Large checkpoint files filling disk

```
Warning: Disk space low - checkpoint may fail
```

Solution: Reduce `MAX_CHECKPOINTS`, use `SAVE_AT_SUBSTEP = OFF`, or use larger storage.

1.9 Best Practices

1.9.1 Do's

Enable checkpoints for any analysis longer than a few minutes
Use `SAVE_AT_STEP_END = ON` as the baseline strategy
Keep `MAX_CHECKPOINTS` ≥ 2 to have a backup
Use `VALIDATE_MESH = ON` in `% RestartFrom` to catch mismatches early
Store checkpoints on fast local storage when possible
Use relative paths for portability between systems
Keep the same input file structure when restarting

1.9.2 Don'ts

Don't set `SAVE_FREQUENCY = 1` (creates excessive I/O overhead)
Don't set `MAX_CHECKPOINTS = 0` (disables rotation, fills disk)
Don't modify mesh topology between checkpoint and restart
Don't rely on checkpoints for long-term archival (use full output files)
Don't delete checkpoints while simulation is running

1.10 Troubleshooting

1.10.1 Q: Why is my checkpoint file much larger than expected?

A: Check for:

1. Many custom state variables at Gauss points
2. Dense contact surfaces (stores contact state per pair)
3. Large number of Gauss points (higher-order elements)
4. Coupled analysis with multiple pressure DOFs

1.10.2 Q: Can I restart with a modified mesh?

A: No. The mesh topology must be identical. For mesh modifications, you must:

1. Complete the analysis to a certain point
2. Export results (displacements, stresses)
3. Create new input with modified mesh
4. Use initial assignment to map old results to new mesh

1.10.3 Q: How do I resume from the "latest" checkpoint automatically?

A: FALCON maintains a `_latest.adfalcon` copy of the most recent checkpoint. Use:

```
% RestartFrom
  FILE = ./checkpoints/simulation_latest.adfalcon
%%%
```

This file is automatically updated after each successful checkpoint save.

1.10.4 Q: Why does my restarted simulation produce slightly different results?

A: If results differ after restart, check:

1. Input file consistency (same boundary conditions, materials)
2. Step definitions match the checkpoint's step configuration
3. No modifications to load application parameters

With proper configuration, restarts should be **bit-exact**.

1.11 Technical Details

1.11.1 Binary Format Structure

```
+————————————————+
|  Header (152 bytes, packed)                 |
|  - Magic number: 0x46414C434F4E4350         |
|  - Version (4 bytes): Currently 9           |
|  - Simulation state (step, substep, time)   |
|  - Mesh dimensions (nodes, elements, DOFs)  |
|  - Mesh + config fingerprints (hashes)      |
|  - CRC64 checksum                           |
|  - Timestamp                                |
|————————————————|
|  Mesh Scalars                               |
|  - Time state, DOF counts, flags            |
|  - stepRemainingTime, stepStartTimeSnapshot |
|————————————————|
|  Global Vectors (Eigen VectorXd)            |
|  - FEXT, globalExtForce, F_internal         |
|  - U, DeltaU, DU_DT, D2U_DT2                |
|  - Pressure forces, saved forces            |
|————————————————|
|  Nodes Section                              |
|  - Count, then serialized Node structs      |
|  - Coordinates, DOF indices, constraints    |
```

```
|————————————————|
|  Elements Section                          |
|  - Count, then serialized Element structs  |
|  - Includes all Gauss point state          |
|  - State variables, stress/strain tensors  |
|————————————————|
|  Simulation Steps                          |
|  - Minimal step fields (stream alignment)  |
|————————————————|
|  Contact & Rigid Bodies                    |
|  - Contact pairs, rigid body definitions   |
|————————————————|
|  Boundary Conditions                       |
|  - Body forces with load factors           |
|  - Stress boundaries with load factors     |
|  - Prescribed values with load factors     |
|  - Load application parameters             |
|————————————————|
|  Sparse Matrices                           |
|  - globalK, globalM, globalC, etc.         |
|————————————————|
|  Adjacency Map (ej)                        |
|  - Node-to-element connectivity            |
|————————————————|
|  Session State                             |
|  - Auto-increment state per step           |
|  - lastSuccessfulDt, successStreak         |
|  - dynamicMax, growthCooldown              |
|————————————————+
```

1.11.2 Checksum Verification

FALCON uses CRC64 for data integrity:

1. **On save:** Compute checksum over entire data payload (after header) using streaming
2. **Store:** Write checksum into header, then update header in file
3. **On load:** Read header, read data, recompute checksum
4. **Verify:** Compare stored vs. computed checksum
5. **Fail:** Return `ChecksumMismatch` if they differ

1.11.3 Memory Considerations

Checkpoint loading verifies the checksum in a streaming manner and then deserializes directly into the in-memory mesh state.

As a rule of thumb, peak memory is dominated by the **mesh state itself** (vectors, Gauss-point data, contact state, sparse matrices), not by buffering the full checkpoint file.

1.12 Quick Reference

1.12.1 Minimal Checkpoint Setup

```
% CheckpointControl
  ENABLED = ON
  OUTPUT_PATH = ./checkpoints/sim
%%%
```

1.12.2 Minimal Restart Setup

```
% RestartFrom
  FILE = ./checkpoints/sim_step2_sub100.adfalcon
%%%
```

1.12.3 All Parameters with Defaults

```
% CheckpointControl
  ENABLED = OFF
  OUTPUT_PATH = ./checkpoints/sim
  SAVE_FREQUENCY = 0
  SAVE_AT_STEP_END = ON
  SAVE_AT_SUBSTEP = OFF
  MAX_CHECKPOINTS = 3
  VALIDATE_ON_RESTART = ON  # currently unused
  VERBOSE_LOGGING = ON
%%%

% RestartFrom
  FILE =
  VALIDATE_MESH = ON
  TARGET_STEP = -1
%%%
```
