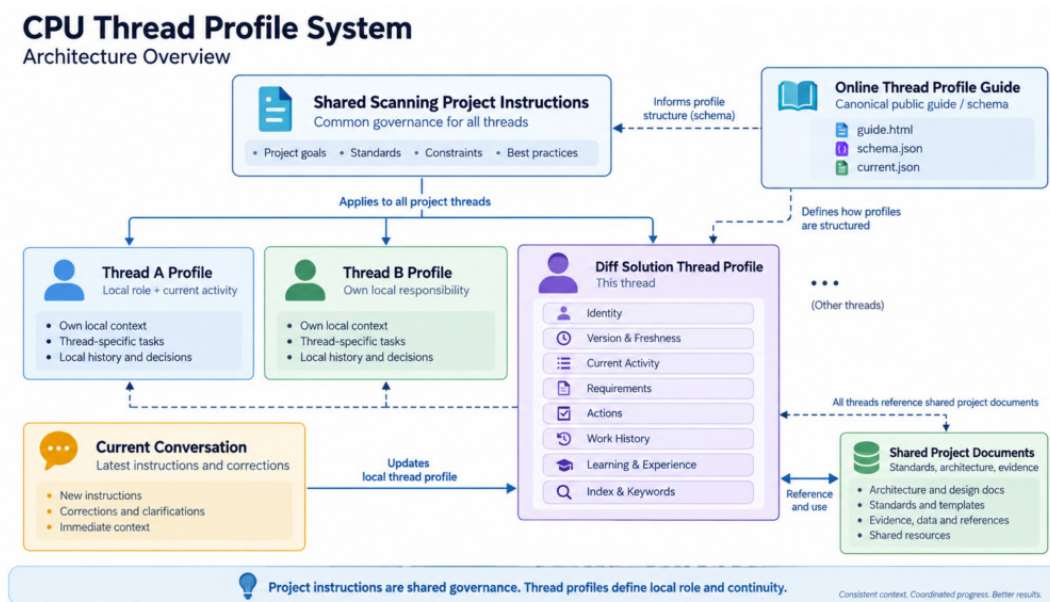


# AI Assistant: Thread Profile Guide

Mental Revolution

Published: September 12, 2026



## AI Assistant: Thread Profile Guide

### Index

1. Purpose
2. Profile hierarchy
3. File format
4. File naming
5. Internal file labelling
6. Versioning and revision control
7. Structure map
8. Current activity
9. Requirements
10. Actions
11. Work history
12. Training and role readiness
13. Learning and experience
14. Provenance
15. Indexing and keywords

16. Input rules
17. Dates
18. Status and priority values
19. Profile maintenance
20. Loading and resuming a thread

## 1. Purpose

A Thread Profile provides continuity for one working thread within a wider project.

It records the thread's local role, current activity, requirements, actions, relevant work history, learning, experience and role-specific training needs.

The profile is maintained by the assistant working in that thread.

It does not replace:

- project instructions;
- shared project documents;
- the conversation;
- source material;
- test evidence.

## 2. Profile hierarchy

Information should be interpreted in this order:

1. Current user instructions
2. Shared project instructions
3. Shared project documents
4. Thread Profile
5. Earlier thread history

A Thread Profile defines the local role of its own thread only.

Thread-specific information must not be treated as applying to every thread in the parent project.

## 3. File format

Thread Profiles use JSON.

Required formatting:

- UTF-8

- valid JSON
- one top-level object
- no comments
- consistent two-space indentation
- lowercase machine keys using snake\_case
- arrays used for multiple entries
- null used only where a known field currently has no value
- empty arrays [] used where a collection is valid but currently contains no entries

A .json profile file must contain JSON only.

## 4. File naming

Use:

--thread-profile-v.json

Example:

scanning-diff-thread-profile-v0.1.0-rev3.json

## Naming rules

- lowercase filenames;
- words separated with hyphens;
- no spaces;
- project name first;
- local role or subject second;
- thread-profile identifies the artifact;
- version appears immediately before .json.

The filename version and internal profile version must agree.

The internal profile metadata is authoritative if a discrepancy is discovered.

## 5. Internal file labelling

The profile must identify itself internally.

Example:

```
{ "schema": { "name": "CPU Thread Profile", "version": "0.5" }, "profile_versioning": { "profile_version": "0.1.0-rev3" } }
```

Human-readable titles may use normal title case.

Machine keys remain lowercase snake\_case.

Stable entry prefixes may include:

- REQ — Requirement
- ACT — Action
- IDX — Index entry
- TRN — Training requirement
- SUB — Sub-project
- PLG — Plugin
- EXP — Experience
- LES — Lesson
- DEC — Decision
- REF — Reference

Example:

REQ-001

IDs must be unique within the profile.

## 6. Versioning and revision control

The profile has two separate versions.

### Schema version

Describes the structure and rules used by the JSON profile.

Example:

0.5

### Profile version

Describes the current issue of the individual thread profile.

Use:

MAJOR.MINOR.PATCH

During revision before an up-issue:

MAJOR.MINOR.PATCH-revN

Example:

0.1.0-rev3

A profile revision should also record:

- last updated date;
- previous profile version;
- freshness checkpoint;
- revision history.

The freshness checkpoint identifies the latest substantive thread activity included in the profile.

## 7. Structure map

The Thread Profile is organised into six functional layers.

### 7.1 Profile Control

Defines what the profile is, whose thread it belongs to, which schema it follows, which version is current, and how recently it was updated.

### 7.2 Current Handover

Contains the minimum information required for a new assistant instance to resume the work correctly.

### 7.3 Role Capability

Records whether the thread has sufficiently current role-specific knowledge and what additional training or research remains necessary.

### 7.4 Navigation and Discovery

Provides the structure map, keywords and index used to locate relevant information without reading the entire profile.

### 7.5 Work and Experience

Preserves useful context, work history, learning, experience, case studies, corrections and reusable knowledge gained through the thread.

### 7.6 Project Control

Records requirements, current direction, actions and their status.

### Recommended reading order

A new instance should normally read the profile in this order:

1. schema 2. profile\_versioning 3. thread\_identity 4. current\_activity 5. role\_training 6. additional\_training\_required 7. requirements\_list 8. current\_direction 9. action\_list 10. knowledge\_index 11. keyword\_taxonomy 12. relevant\_work\_history 13. relevant\_learning\_experience 14. status\_semantics 15. provenance where required

The knowledge\_index should be used to retrieve only the historical or technical information relevant to the current task.

### 8. Current activity

current\_activity is the primary handover section.

It should normally contain:

```
{ "current_focus": "", "carry_over_summary": "", "last_completed": "", "next_step": "", "resume_context": "",  
  "blocked_by": [], "related_requirements": [], "related_actions": [], "updated_on": "" }
```

It should remain concise enough for a new instance to read first.

### 9. Requirements

Requirements are stored in requirements\_list.

Each requirement should contain only one requirement.

Example:

```
{ "id": "REQ-001", "requirement": "", "status": "established", "priority": "high", "added_on": "", "notes": "" }
```

Do not create additional explanatory fields unless they add useful information.

### 10. Actions

Actions are stored in action\_list.

Each entry represents one actionable item.

Example:

```
{ "id": "ACT-001", "action": "", "status": "pending", "priority": "high", "depends_on": [], "added_on": "", "completion_evidence": "" }
```

An action being present does not itself authorise implementation.

## 11. Work history

work\_history may contain:

sub\_projects plugins components

Only work relevant to the thread should be recorded.

A plugin entry may include:

```
{ "id": "PLG-001", "name": "", "platform": "", "upstream_version": "", "local_version": "", "status": "", "first_worked_on": "", "last_worked_on": "", "summary": "" }
```

The profile is not intended to duplicate an existing asset or plugin inventory.

## 12. Training and role readiness

Use role\_training to record the currency of role-specific preparation.

Example:

```
{ "last_reviewed_on": "", "last_updated_on": "", "basis": [] }
```

This records role-specific preparation, not the underlying model training date.

Use additional\_training\_required for identified knowledge gaps.

Example:

```
{ "id": "TRN-001", "topic": "", "reason": "", "priority": "high", "status": "required", "identified_on": "", "completed_on": null }
```

## 13. Learning and experience

Record useful learning where it may prevent repeated work or improve future decisions.

Suitable material includes:

- implementation findings;
- compatibility findings;
- test experience;
- failed approaches;

- successful approaches;
- corrections to assumptions;
- reusable design lessons;
- user-interface findings;
- platform-specific behaviour.

Do not store ordinary conversation simply because it occurred.

## 14. Provenance

The thread is the default provenance source.

The profile should declare:

```
{ "provenance": { "primary_source": "current_thread", "rule": "Unless otherwise stated, requirements, decisions, learning and experience recorded in this profile are derived from this thread." } }
```

Additional references are only needed where information comes from elsewhere, such as:

- files;
- source code;
- logs;
- tests;
- generated artifacts;
- external documentation;
- research;
- another project resource.

## 15. Indexing and keywords

The profile should contain a `knowledge_index`.

An index entry should normally contain:

```
{ "id": "IDX-001", "path": "", "type": "", "status": "", "title": "", "summary": "", "keywords": [], "added_on": "" }
```

The index is intended to locate useful material without requiring the whole profile to be read.

A `keyword_taxonomy` may group recurring terms by area.

Example:

```
{ "platforms": [ "ClassicPress", "WordPress" ], "activities": [ "quarantine", "diff", "scanning" ] }
```

Keywords should be short search terms rather than sentences.

## 16. Input rules

Input restrictions should remain proportionate.

### Hard rules

Apply hard validation to:

- valid JSON type;
- unique IDs;
- valid dates;
- recognised status values;
- recognised priority values;
- valid profile version format;
- reasonable maximum keyword count.

### Soft limits

Suggested advisory limits:

Input

Advisory limit

ID

64 characters

title

160 characters

summary

1,000 characters

requirement

2,000 characters

action

2,000 characters

notes

4,000 characters

resume context

4,000 characters

keyword

80 characters

keywords per entry

30

references per entry

50

These are warnings rather than reasons to discard useful information.

If an entry becomes excessively large, summarise it and reference the detailed source.

The whole profile should generate a size warning at approximately 2 MB rather than impose an arbitrary hard limit.

## 17. Dates

Use ISO date format:

YYYY-MM-DD

Common date fields include:

- added\_on
- updated\_on
- completed\_on
- identified\_on
- first\_worked\_on
- last\_worked\_on
- last\_reviewed\_on

added\_on means the date the information was added to the profile.

It does not necessarily mean the date the underlying event occurred.

## 18. Status and priority values

### Requirements

established candidate unresolved satisfied superseded

### Actions

pending research ready in\_progress blocked completed deferred cancelled

### Priority

critical high medium low deferred

Additional values should only be introduced where the existing set cannot represent the required state.

## 19. Profile maintenance

The assistant working in the thread is responsible for maintaining the profile.

A profile update should be considered when:

- role or scope changes;
- a requirement changes;
- an action changes;
- current activity changes;
- the next step changes;
- a significant result is obtained;
- useful learning or experience is gained;
- a plugin or sub-project is worked on;
- an important assumption is corrected;
- training requirements change.

Minor conversation does not require a profile revision.

When a substantive update is made:

1. update the affected entries;
2. add relevant dates;
3. update current activity;
4. update index and keywords where necessary;
5. update the freshness checkpoint;
6. increment the profile revision;

7. update the integrity value where used.

## 20. Loading and resuming a thread

When a Thread Profile is available, a new instance should read it before substantive work.

It should first establish:

1. what role this thread performs;
2. whether the profile is current;
3. what activity is presently underway;
4. what must happen next;
5. which requirements constrain that work;
6. whether additional training is required;
7. which actions remain open.

The index should then be used to retrieve only the additional learning or experience relevant to the current task.

If the thread contains substantive activity later than the profile freshness checkpoint, the profile should be treated as potentially stale and updated before it is relied upon as the current handover record.