Agent Memory spec [draft v2]

Problem

- Most harnesses implement some sort of memory solution, but there is no standardization beyond “memory as markdown”
    - Some harnesses re-use `AGENTS.md` (DeepAgents), some use `MEMORY.md` (Claude Code, Codex), some use `USER.md` / `HUMAN.md` (OpenClaw, Hermes)
- Lack of **memory portability** (value for agent users)
    - Non-trivial to move “memory” from harness-to-harness (should be able to “move” a stateful agent by dragging and dropping
    - Cannot design systems/models/algorithms for memory without a standard target
- Lack of **memory consistency** (value for agent developers)
    - If you’re a developer it’s unclear how you should structure your file-based memory system (How many files? What names? Nesting / no-nesting?)

Existing memory standards

Existing standards cover some forms of memory:

- **AGENTS.md**
    - *Think of AGENTS.md as a README for agents: a dedicated, predictable place to provide the context and instructions to help AI coding agents work on your project.*
    - Support for nested AGENTS.md within large repositories
    - Project/repository-specific
- **Agent Skills**
    - Procedural memory for embedding specific expertise/capabilities into agents - must be able to define *when* to use it.
        - Skills require an explicit trigger condition (the description).
    - Can be project, environment, or agent-scoped
- **MCP**
    - Protocol of retrieving/storing external context. Does not define a context hierarchy for agent memory (in-context vs out of context).

Current standards lack specifications for:

- Core memory (system-prompt/in-context memory)
- External memory that is *not* procedural (e.g. episodic, semantic, or other generic context owned by the agent) - including standards for how to progressively disclose external memory.

Why should an “agent memory” spec define? Key job: **progressive memory disclosure**

- Assumption: memory storage is “markdown files in a folder” (everyone does this)
    - We also expect agent memory to grow quickly over time
- What memory is loaded into the context window is incredibly important
    - E.g. a “persona” *needs* to be injected into the system prompt or it won’t condition the LLM’s outputs properly. Daily logs of activity should not be injected into the system prompt.
- Main job of the spec: make it clear what memory goes **inside** the context window (or system prompt), and what will stay **outside** the context window

What’s out-of-scope:

- Git-tracking (eg MemFS), length enforcement, storage mechanism, non-markdown files
- Specifying the structure of memory beyond what is necessary for progressive disclosure of agent-owned context

Agent Memory Specification

The agent memory specification is a lightweight format that specifies how memory for an agent should be structured.

Agent Memory is a directory containing a top level `MEMORY.md` file, and additional markdown files and subdirectories:

```
None

# minimal
memory/
├── MEMORY.md

# expanded
memory/
├── MEMORY.md
├── persona.md
├── human.md
├── projects/
│   └── project.md
└── notes/
```

```
|    └── 2026-08-12.md
```

Memory must be structured to provide agents with a context hierarchy and mechanism for progressive memory disclosure:

- The memory hierarchy is defined by the file hierarchy:
    - Top-level files are at the top of the memory hierarchy (what is most important for an agent to know)
    - **Subdirectories are a mechanism for progressive disclosure**: placing context in a deeper subdirectory places it lower in the memory hierarchy.
- Within a directory level, the `MEMORY.md` file should provide necessary context for the agent to progressively discover context stored in immediate subdirectories (if they exist).

Valid memory configurations

Top-level `MEMORY.md` only, with collapsed logs (similar to Claude Code):

```
None
memory/
├── MEMORY.md
├── example-repository-name/
|    └── 03082026.md
|    └── 03092026.md
```

Core memory only, with no collapsed logs (similar to Hermes/OpenClaw):

```
None
memory/
├── MEMORY.md
├── SOUL.md
├── USER.md
```

```
None
memory/
├── MEMORY.md
```

Nested subdirectories for progressive disclosure of external memory (similar to Letta Code):

```
None
memory/
├── MEMORY.md
├── projects/
│   ├── MEMORY.md
│   ├── project_1/
│       ├─ MEMORY.md
```

Invalid memory configurations

No top-level MEMORY.md file:

```
None
memory/
├── project-1/
│   ├── MEMORY.md
│   └── user_preferences.md
│   └── bad_bugs.md
├── project-2/
│   ├── MEMORY.md
│   └── memories_03082026.md
│   └── memories_03092026.md
```

Harness contract

A conforming harness follows the following four rules:

Top-level .md files are at least partially visible to the agent at all times (in-context). The harness may truncate the contents to minimize

Defer nested .md files. Markdown below the root is not automatically loaded into context.

Surface deferred memory. If nested memory exists, the harness gives the agent enough information to know of deferred memory 1-level down (i.e. context in immediate subdirectories). This metadata is placed in MEMORY.md and can be learned by the agent or deterministically generated by the harness.

Support selective reads. The agent can load individual deferred files when needed, through file tools or an equivalent memory-read interface.

Size guidance [recommendation]

Top-level memory should be constrained in size, with less critical memory being deferred by being placed in subdirectories to avoid context bloat. The recommended size for top-level memories is:

- 500 tokens (~20,000 characters) per top-level file
- 20,000 tokens total for in-context memory (total tokens across all top-level files)

Harnesses can enforce size limitations at edit time, e.g. by blocking a memory append if over a limit.
Agents should also be guided to make use of nested for progressive memory disclosure and to keep the root context cost small.

Nesting subdirectories

The memory specification does not constrain the degree of nesting

Optional metadata

Memory files may include additional metadata in markdown headers to instruct harnesses on how to

Field	**Description**
description	Describes the purpose of the memory contained in the file. The description can help guide agents writing to memory about what memory to save where. Examples: ● What I know about the person I am interacting with ● Historical bug investigation in {{repo_name}}.
metadata	Optional metadata about the memory file Examples: ● {“session_evidence”: {“session_1”, “session_2”} ● {“read_only”: true}

Scope

Agent Memory standardizes the directory and loading behavior. It does **not** define:

- how memory is edited or organized (beyond core/root + external)
    - specific harnesses may require specific files
- who owns or may write each file
- where memory is stored (locally vs mirror) or how it is synchronized
- whether memory is version-controlled (e.g. w/ git)
- when a running conversation refreshes the prefix after an edit

Harnesses may implement those features however they choose while preserving the directory contract above.