ChatGPT

허브 구조

파일 기반 메시지 허브(`agents_hub` 디렉토리)는 **queue/processing/archive** 구조로 구성됩니다. 새 작업/메시지 등록(`invoke hub.send`) 시 JSON 파일이 `agents_hub/queue/`에 생성됩니다. 대기열 조회(`invoke hub.inbox`)는 queue 내 해당 에이전트宛 파일을 나열하고, 클레임(`invoke hub.claim`) 시 해당 JSON을 `agents_hub/processing/<agent>/`로 이동시켜 상태를 processing으로 표시합니다. 완료(`invoke hub.complete`) 시 상태(success/failed)를 설정하고 파일을 날짜별 `agents_hub/archive/<YYYYMMDD>/<status>/`로 옮깁니다 [1] [2]. 각 JSON에는 `id, sender, to, title, body, tags, status, created_at, claimed_at, completed_at` 등이 기록되어 작업 흐름을 관리합니다. 예를 들어, `hub.send`는 broker.py에 의해 JSON 파일을 큐에 생성하며, `hub.claim`은 이 파일을 처리중 폴더로 이동시키고 타임스탬프를 붙입니다 [1] [3].
허브 구조는 단순하지만 효과적으로 재현 가능한 로그를 제공합니다. 또한 `hub_manager.py`는 작업 완료 시 `docs/HUB.md`의 항목을 "Completed Tasks"로 이동해 **HUB 파일**을 동기화합니다. (`hub_manager.move_task_to_completed` 참조).

컨텍스트 관리

`context/` 디렉토리에는 에이전트 간 교차 메시징과 컨텍스트 데이터가 저장됩니다. `context/messages.jsonl`는 각 대화 메시지를 JSONL 형식으로 누적 기록하며, 다중 에이전트 협업 내용을 보관합니다. 예를 들어 작업 요약이나 다음 액션 같은 에이전트 간 메시지가 append 됩니다 (쉘로그 예 참조). 또한 `.gemini/context_policy.yaml` 파일을 사용하여 각 상황별 컨텍스트 조회 정책을 정의합니다. 이 YAML에는 `session_start_briefing` 등의 정책 이름 아래에 포함할 소스(`doc_tag`나 파일 기준 등)와 최대 토큰 제한을 기술합니다 [4] [5]. 런타임에 `scripts/prompt_builder.py`가 정책을 로드해 지정된 태그나 파일을 `ContextStore`로 조회하고, 컨텐츠를 그대로 또는 요약하여 프롬프트 컨텍스트로 조합합니다 [5]. 이를 통해 필요한 문서만 안전하게 참조하고, 허용된 정보(화이트리스트)만 활용할 수 있습니다. 예를 들어, 테스트 정책에 따라 "Active Tasks" 태그가 붙은 문서가 요약·삽입되어 프롬프트에 포함되는 구조입니다 [4] [5]. 이로써 불필요한 노출 없이 작업 관련 문서만 공유하며 컨텍스트를 관리합니다.

로그 관리 체계

`logs/`와 `terminal_logs/` 디렉토리는 세션별 기록을 보관합니다. `terminal_logs/YYYY-MM-DD/`에는 PowerShell 녹취(Transcript)가 저장되는데, 파일명에 `agent-<name>`과 터미널 환경명이 포함되어 **에이전트별 라벨링**이 이뤄집니다 (예: `session_025140__agent-claude__term-consolehost__pid-4276.txt`). 이 파일은 세션 시작·종료 시간, 사용자, 호스트 정보와 함께 대화 및 프롬프트 출력을 모두 기록하여 완전한 **세션 트랜스크립트**를 제공합니다. (위 예시처럼 에러 메시지나 완료 상태 등 전체 흐름을 포함합니다.)
또한 `usage.db`라는 SQLite DB 파일을 통해 모든 Invoke 태스크 실행 이력을 남깁니다. `scripts/usage_tracker.py`는 태스크명, 이벤트 유형(세션 시작/종료 등), 커맨드, 반환코드, 표준출력·에러 등을 `usage` 테이블(컬럼: `timestamp, task_name, event_type, command, returncode, stdout, stderr`)에 기록합니다 [6]. (예: 세션 시작 시 `log_usage('session','start',command='start')`가 호출되고, 종료 시 비동기로 기록합니다.) 이 이력은 워크스페이스 사용 통계와 디버그에 활용됩니다.

리포트/스크래치패드

`reports/` 폴더는 **공식 산출물**(예: 시스템 감사 리포트, 헬스체크 결과 등)을 담습니다. 반면 `scratchpad/`는 **임시 작업물**(아이디어 노트, 일일 로그, 계획안, 디버그 메모 등)을 위한 공간입니다. 예컨대 `scratchpad/2_proposals_and_plans/`에는 향후 개선안 문서가, `scratchpad/1_daily_logs/`에는 데일리 워크로그 초안이 위치합니다. 이 둘 간 경계는 명확합니다: scratchpad의 문서는 검토·수정 중인 초안이나 브레인스토밍 자료이고, reports는 완결된 분석·보고서입니다. 따라서 개발·분석 도중 임시로 쓰인 파일은 scratchpad에 남겨두고, 최종 결과는 reports로 이동·관리하는 방침입니다.

작업 수명주기와 회복성

HUB.md의 섹션(Active, Staging, Planned, Paused, Completed)과 연계된 상태 관리가 수립되어 있습니다. **Planned**는 대기 중인 작업 목록(백로그)이고, **Staging**은 곧 착수될 준비 단계, **Active**는 현재 작업 중인 항목입니다. **Paused**는 일시 중단된 작업, **Completed**는 완료된 작업으로 이동합니다 [7] [8]. 일반 흐름은 Planned→Staging→Active→Completed 순이며, 필요 시 Active→Paused→다시 Active로 재개할 수도 있습니다. 작업 실패나 중단 시 `hub.complete --status failed`로 기록하며, 후속 회복은 새 작업으로 재할당하거나 상태 전환으로 처리합니다. 작업 완료 시엔 `hub_manager.move_task_to_completed`가 호출되어 HUB.md의 Active 섹션에서 제거하고 Completed에 추가하며 타임스탬프를 갱신합니다. 또한 각 명령의 실행 결과와 변경 파일 목록은 `__lastSession__` 블록에 자동 기록되어 세션이력에 남습니다(예: `update_session_end_info` 기능). 이로써 작업 이력과 전환이 자동화되고 문서화됩니다.

장점과 한계

파일 기반 메시지 큐 방식의 장점은 **단순성**과 **재현성**입니다. 큐·처리·아카이브 상태가 모두 파일로 남아, 언제든 로그를 돌려보거나 복원할 수 있습니다. 추가 서버나 복잡한 종속성 없이 CLI 환경에서 작동하며, git 이력을 통해 변경 내용을 추적할 수 있습니다. 반면 단점으로는 동시성 충돌 위험이 있습니다. 예를 들어 두 에이전트가 동시에 같은 작업을 클레임하려 할 때 파일 이동 경합(race)이 발생할 수 있습니다. 또한 로그 파일이 산발적으로 많아져 관리가 복잡해질 수 있습니다. 예를 들어 하루치 세션 로그가 터미널별로 수십 개가 되면 정리와 검색이 번거롭습니다. 이밖에 파일 입출력 특성상 I/O 지연이나 파일 시스템 오류에 취약할 수 있습니다. (예: 잠금이 없으면 중간 상태 파일이 꼬일 위험.)

사용 맥락 요약

이 워크스페이스에서 메시징과 작업 관리는 모두 **파일 기반으로 동작**합니다. Gemini·Codex(향후 Claude) 에이전트는 `invoke hub.send/inbox/claim/complete` 명령어로 에이전트 간 작업을 주고받습니다 [1] [3]. 에이전트 메시지와 작업 요청은 `agents_hub/queue`에 JSON 파일로 기록되고, 처리 단계는 `processing`과 `archive`로 단계별로 옮겨지며 영구히 남습니다. 한편 컨텍스트 공유는 `context/messages.jsonl`과 인덱스/정책 파일(`context_policy.yaml`)을 통해 이루어집니다. 기록된 로그들은 `terminal_logs/`에 세션 녹취로, `usage.db`에 실행 히스토리로 남아 후속 검증과 분석에 활용됩니다 [6]. 이러한 구조 덕분에 워크플로우가 명확히 문서화되어, 새로운 에이전트(예: Claude)도 빠르게 시스템 흐름과 작업 단계를 파악할 수 있습니다.

---

1 3 tasks.py

https://github.com/etloveaui/multi-agent-workspace/blob/31b0105d66b544a46410212c77899d3c5194a2ca/tasks.py

2 hub_complete.ps1

https://github.com/etloveaui/multi-agent-workspace/blob/31b0105d66b544a46410212c77899d3c5194a2ca/tools/hub_complete.ps1

4 test_context_engine.py

https://github.com/etloveaui/multi-agent-workspace/blob/31b0105d66b544a46410212c77899d3c5194a2ca/tests/test_context_engine.py

5 prompt_builder.py

https://github.com/etloveaui/multi-agent-workspace/blob/31b0105d66b544a46410212c77899d3c5194a2ca/scripts/prompt_builder.py

6 usage_tracker.py

https://github.com/etloveaui/multi-agent-workspace/blob/31b0105d66b544a46410212c77899d3c5194a2ca/scripts/usage_tracker.py

7 8 HUB.md

https://github.com/etloveaui/multi-agent-workspace/blob/31b0105d66b544a46410212c77899d3c5194a2ca/docs/HUB.md