

Apple Ecosystem MCP Server

Implementation Plan — Built with Claude Code

Author: Abhinav Agrawal · April 2026 · v1.0

Overview

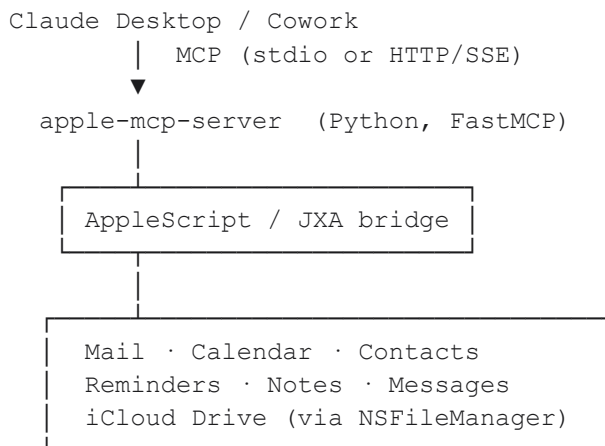
This document is a complete, Claude-executable implementation plan for an Apple Ecosystem MCP (Model Context Protocol) server. The server bridges Claude to native macOS apps — Mail, Calendar, Contacts, Reminders, Notes, Messages, and iCloud Drive — via AppleScript and Python, running locally on the user's Mac. Once complete, the server will be submitted to Anthropic's official Claude Connectors Directory.

Existing open-source attempts (jxnl/python-apple-mcp, pyapple-mcp, peakmojo/applescript-mcp) demonstrate feasibility but lack the polish, security handling, permissions flow, and packaging required for the Anthropic directory. This plan produces a production-quality implementation.

Architecture

High-Level Design

The server runs as a local process on the user's Mac. Claude (running in Claude Code or Cowork) connects to it over stdio or HTTP/SSE. The server translates MCP tool calls into AppleScript or Python/JXA commands that invoke native macOS frameworks.



Technology Choices

Python 3.11+ with FastMCP is the recommended stack. Reasons:

- FastMCP provides a decorator-based tool registration API — minimal boilerplate
- Python subprocess can call osascript (AppleScript) for any macOS app
- Pure Python avoids Swift compilation complexity while still being fast enough for I/O-bound tasks
- Packaging as a standalone binary via PyInstaller makes distribution simple for non-developers

Transport

Support both transports so users can connect from Claude Code (stdio) and from Cowork / remote Claude (HTTP/SSE). FastMCP handles both with a single flag.

- stdio — default for Claude Code CLI (claude mcp add apple-mcp)
- HTTP/SSE — for Cowork and future remote connections (localhost:8765)

Tool Catalog

Each capability below becomes one or more MCP tools. Claude sees these as functions it can call.

Mail

Tool Name	Apple API	Description
<code>mail_search</code>	<code>Mail.app / AppleScript</code>	Search inbox/sent by query, date range, sender
<code>mail_get_thread</code>	<code>Mail.app / AppleScript</code>	Fetch a full email thread by subject or ID
<code>mail_send</code>	<code>Mail.app / AppleScript</code>	Compose and send an email (to, cc, subject, body)
<code>mail_create_draft</code>	<code>Mail.app / AppleScript</code>	Save a draft without sending
<code>mail_list_mailboxes</code>	<code>Mail.app / AppleScript</code>	List all mailboxes/folders
<code>mail_move_message</code>	<code>Mail.app / AppleScript</code>	Move message to a mailbox
<code>mail_flag_message</code>	<code>Mail.app / AppleScript</code>	Flag or unflag a message

Calendar

Tool Name	Apple API	Description
<code>calendar_list_events</code>	<code>EventKit / AppleScript</code>	List events in a date range across calendars
<code>calendar_get_event</code>	<code>EventKit / AppleScript</code>	Get details of a specific event
<code>calendar_create_event</code>	<code>EventKit / AppleScript</code>	Create event with title, time, location, notes, invitees

calendar_update_event	EventKit / AppleScript	Update an existing event
calendar_delete_event	EventKit / AppleScript	Delete an event by ID
calendar_find_free_time	EventKit / AppleScript	Find free slots between two datetimes
calendar_list_calendars	EventKit / AppleScript	List all calendars

Contacts

Tool Name	Apple API	Description
contacts_search	Contacts.app / AppleScript	Search contacts by name, email, phone, company
contacts_get	Contacts.app / AppleScript	Get full contact record by ID
contacts_create	Contacts.app / AppleScript	Create a new contact
contacts_update	Contacts.app / AppleScript	Update fields on an existing contact
contacts_list_groups	Contacts.app / AppleScript	List contact groups

Reminders

Tool Name	Apple API	Description
reminders_list	EventKit / AppleScript	List reminders, optionally filtered by list or status
reminders_create	EventKit / AppleScript	Create a reminder with due date, notes, priority
reminders_complete	EventKit / AppleScript	Mark a reminder as complete
reminders_delete	EventKit / AppleScript	Delete a reminder
reminders_lists	EventKit / AppleScript	List all Reminders lists

Notes

Tool Name	Apple API	Description
notes_search	Notes.app / AppleScript	Full-text search across all notes
notes_get	Notes.app / AppleScript	Get note content by title or ID
notes_create	Notes.app / AppleScript	Create a new note in a specified folder
notes_update	Notes.app / AppleScript	Append or replace note content
notes_delete	Notes.app / AppleScript	Delete a note
notes_folders	Notes.app / AppleScript	List all note folders

Messages

Tool Name	Apple API	Description
messages_send	Messages.app / AppleScript	Send an iMessage or SMS to a contact or number
messages_search	Messages.app / AppleScript	Search message history by contact or keyword

iCloud Drive

Tool Name	Apple API	Description
icloud_list	NSFileManager / Python	List files and folders in iCloud Drive
icloud_read	NSFileManager / Python	Read a text or structured file
icloud_write	NSFileManager / Python	Write or update a file
icloud_move	NSFileManager / Python	Move or rename a file
icloud_delete	NSFileManager / Python	Delete a file (with confirmation)
icloud_search	Spotlight / mdfind	Search iCloud Drive by filename or content

Implementation Phases

Hand each phase to Claude Code in sequence. Each phase builds directly on the previous.

Phase	Name	Duration	Key Deliverables
1	Scaffolding & Core Server	1–2 days	<ul style="list-style-type: none">• Project structure & pyproject.toml• FastMCP server skeleton• AppleScript bridge utility• Permissions check on startup• stdio + HTTP/SSE transport
2	Mail Integration	2–3 days	<ul style="list-style-type: none">• search, get_thread, send tools• create_draft, move, flag tools• Attachment handling• Unit tests with mock osascript
3	Calendar & Reminders	2–3 days	<ul style="list-style-type: none">• Full CRUD for events• find_free_time algorithm• Reminders CRUD• Recurring event support
4	Contacts, Notes, Messages	2 days	<ul style="list-style-type: none">• Contacts search & CRUD• Notes full-text search & CRUD• Messages send & search• Error handling & retries
5	iCloud Drive	1–2 days	<ul style="list-style-type: none">• File listing & reading• Write/move/delete with safety guards• Spotlight search integration

			<ul style="list-style-type: none"> • Binary vs text file detection
6	Packaging & Submission	2-3 days	<ul style="list-style-type: none"> • PyInstaller standalone binary • Homebrew formula • Claude Code one-liner install • Anthropic directory submission

Phase 1: Scaffolding — Claude Prompt

Copy the prompt below verbatim into Claude Code to complete Phase 1. Each subsequent phase has its own prompt at the end of this document.

PHASE 1 PROMPT — paste into Claude Code

Create a new Python project called `apple-mcp` with this structure:

```
apple-mcp/
  pyproject.toml          # FastMCP, pyobjc-framework-EventKit deps
  README.md
  src/
    apple_mcp/
      __init__.py
      server.py           # FastMCP app, stdio + SSE transports
      bridge.py           # run_applescript(script) -> str helper
      permissions.py      # check & request TCC permissions at startup
      tools/
        __init__.py      # registers all tool modules
        mail.py          # placeholder stubs
        calendar.py
        contacts.py
        reminders.py
        notes.py
        messages.py
        icloud.py
      tests/
        conftest.py      # mock_osascript fixture
        test_bridge.py
```

Requirements:

- Use FastMCP (pip install fastmcp)
- bridge.py must call subprocess.run(["osascript", "-e", script]) and raise McpError on non-zero exit
- permissions.py checks Automation, Contacts, and Full Disk Access using tccutil query and prints actionable instructions if missing
- server.py exposes mcp.run(transport="stdio") by default, and HTTP/SSE on port 8765 with --http flag
- Add a hello_apple tool that returns macOS version as a smoke test
- All tools must include a proper docstring — FastMCP uses it as the MCP

tool description shown to Claude

Phase 2: Mail — Claude Prompt

PHASE 2 PROMPT — paste into Claude Code after Phase 1 is complete

Implement the Mail tools in `src/apple_mcp/tools/mail.py`.

Implement these tools using AppleScript via `bridge.run_applescript()`:

```
mail_search(query: str, mailbox: str = "INBOX", limit: int = 20,
            since: str = None) -> list[dict]
    - Returns list of {id, subject, sender, date, preview}
```

```
mail_get_thread(message_id: str) -> dict
    - Returns full message with body, attachments list
```

```
mail_send(to: list[str], subject: str, body: str,
          cc: list[str] = None, reply_to_id: str = None) -> dict
    - Sends via Mail.app, returns {success, message_id}
```

```
mail_create_draft(to: list[str], subject: str, body: str) -> dict
```

```
mail_list_mailboxes() -> list[str]
```

```
mail_move_message(message_id: str, mailbox: str) -> dict
```

```
mail_flag_message(message_id: str, flagged: bool) -> dict
```

AppleScript patterns to follow:

- Use ``tell application "Mail"`` blocks
- Escape all user-supplied strings before interpolation
- Handle ``missing value`` returns gracefully
- Write unit tests in `tests/test_mail.py` using the `mock_osascript` fixture

Phase 3: Calendar & Reminders — Claude Prompt

PHASE 3 PROMPT

Implement calendar and reminders tools.

`calendar.py` tools:

```

calendar_list_calendars() -> list[dict]
calendar_list_events(start: str, end: str, calendar: str = None) ->
list[dict]
calendar_get_event(event_id: str) -> dict
calendar_create_event(title: str, start: str, end: str,
    calendar: str = "Calendar", location: str = None,
    notes: str = None, invitees: list[str] = None) -> dict
calendar_update_event(event_id: str, **kwargs) -> dict
calendar_delete_event(event_id: str) -> dict
calendar_find_free_time(date: str, duration_minutes: int,
    working_hours_start: int = 9,
    working_hours_end: int = 18) -> list[dict]

reminders.py tools:
reminders_lists() -> list[str]
reminders_list(list_name: str = None, completed: bool = False) ->
list[dict]
reminders_create(title: str, list_name: str = "Reminders",
    due: str = None, notes: str = None, priority: int = 0) -> dict
reminders_complete(reminder_id: str) -> dict
reminders_delete(reminder_id: str) -> dict

Use ISO 8601 for all datetimes. Parse with datetime.fromisoformat().
find_free_time: fetch existing events, then return gaps of at least
duration_minutes.

```

Phase 4: Contacts, Notes, Messages — Claude Prompt

PHASE 4 PROMPT

Implement contacts, notes, and messages tools.

contacts.py:

```

contacts_search(query: str, limit: int = 10) -> list[dict]
contacts_get(contact_id: str) -> dict # full record
contacts_create(first: str, last: str = None, email: str = None,
    phone: str = None, company: str = None) -> dict
contacts_update(contact_id: str, **kwargs) -> dict
contacts_list_groups() -> list[str]

```

notes.py:

```

notes_folders() -> list[str]
notes_search(query: str, limit: int = 10) -> list[dict]
notes_get(note_id: str) -> dict
notes_create(title: str, body: str, folder: str = "Notes") -> dict
notes_update(note_id: str, body: str, append: bool = True) -> dict

```

```
notes_delete(note_id: str) -> dict

messages.py:
    messages_send(to: str, body: str, service: str = "iMessage") -> dict
    messages_search(contact: str = None, query: str = None,
        limit: int = 20) -> list[dict]

Security note for messages_send:
    - Validate `to` is a phone number or email before sending
    - Include a dry_run: bool = False param; when True, return what would
      be sent without actually sending
```

Phase 5: iCloud Drive — Claude Prompt

```
# PHASE 5 PROMPT

Implement iCloud Drive tools in icloud.py.

icloud_list(path: str = "/") -> list[dict]
    - Resolve path relative to ~/Library/Mobile
    Documents/com~apple~CloudDocs/
    - Return [{name, path, size, modified, is_dir}]

icloud_read(path: str) -> dict
    - Detect encoding; return {path, content, encoding, size}
    - Refuse to read files > 10 MB; return error with size info

icloud_write(path: str, content: str,
    create_dirs: bool = True) -> dict
    - Write UTF-8 text; create parent dirs if create_dirs=True

icloud_move(src: str, dst: str) -> dict

icloud_delete(path: str, confirm: bool = False) -> dict
    - If confirm=False, return a preview of what would be deleted
    - Only delete when confirm=True

icloud_search(query: str, path: str = "/",
    content_search: bool = False) -> list[dict]
    - Use mdfind for Spotlight; scope to iCloud root
    - content_search=True adds -interpret flag

All paths must be sandboxed to iCloud root — raise McpError if the
resolved absolute path escapes ~/Library/Mobile Documents/.
```

Phase 6: Packaging & Submission — Claude Prompt

```
# PHASE 6 PROMPT
```

```
Package and prepare the server for distribution.
```

1. PyInstaller binary
 - Create build.sh that runs PyInstaller --onefile --name apple-mcp
 - Output: dist/apple-mcp (single executable, no Python required)
2. Homebrew formula (Formula/apple-mcp.rb)
 - url points to GitHub release tarball
 - install block: bin.install "apple-mcp"
 - test block: assert_match "macOS", shell_output("#{bin}/apple-mcp --version")
3. Claude Code install one-liner (add to README):
claude mcp add apple-mcp -- /usr/local/bin/apple-mcp
4. Cowork HTTP install (add to README):
Run: apple-mcp --http
Then in Cowork Settings > MCP Servers: http://localhost:8765/sse
5. Update README with:
 - Full tool catalog table
 - macOS permissions setup instructions (System Settings screenshots guidance)
 - Troubleshooting section for common TCC errors
6. Submission checklist:
 - [] All tools have docstrings used as MCP descriptions
 - [] oauth_flow is not needed (local server, no cloud auth)
 - [] MCP Directory Terms accepted
 - [] Submit at <https://claude.com/partners/mcp>

Permissions & Security Notes

macOS requires explicit user approval (TCC — Transparency, Consent, Control) for each app category. Your MCP server must request and guide the user through these.

Required macOS Permissions

- Automation → Mail, Calendar, Contacts, Reminders, Notes, Messages
- Contacts — separate TCC category
- Full Disk Access — needed for iCloud Drive file operations

- Accessibility — only if using UI scripting (avoid if possible)

Security Invariants (enforce in code)

- All iCloud paths resolved and sandboxed before any file operation
- AppleScript user-string inputs escaped via shlex or explicit quoting
- messages_send has dry_run mode and phone/email validation
- icloud_delete requires confirm=True to actually delete
- No credentials stored — server uses macOS keychain delegation via apps
- Server binds only to 127.0.0.1 in HTTP mode — never 0.0.0.0

Quick Start with Claude Code

To kick off the entire project, open Claude Code and run:

```
claude "Create a new directory called apple-mcp, then implement Phase 1 of the Apple MCP server as described in the plan I'm about to paste."
```

Then paste the Phase 1 prompt from this document. Repeat for each subsequent phase.

Each phase is designed to be independently verifiable — run the tests after each phase before moving on:

```
cd apple-mcp && python -m pytest tests/ -v
```

Anthropic Directory Submission Checklist

- Server passes all unit tests (pytest)
- All tools have clear, accurate MCP docstrings
- README includes privacy policy URL and terms of service URL
- Server has a --version flag returning semver string
- No hardcoded credentials or API keys anywhere in code
- Tested on macOS 14 Sonoma and macOS 15 Sequoia
- PyInstaller binary tested on a clean Mac (no Python installed)
- Homebrew formula installable via brew install --formula
- Submit form completed at <https://claude.com/partners/mcp>

Good luck, Abhinav — this fills a real gap in the Claude ecosystem.