Memory tool

Copy page

The memory tool enables Claude to store and retrieve information across conversations through a memory file directory. Claude can create, read, update, and delete files that persist between sessions, allowing it to build knowledge over time without keeping everything in the context window.

The memory tool operates client-side—you control where and how the data is stored through your own infrastructure.

> The memory tool is currently in beta. To enable it, use the beta header `context-management-2025-06-27` in your API requests.
>
> Please reach out through our feedback form to share your feedback on this feature.

Use cases

- Maintain project context across multiple agent executions
- Learn from past interactions, decisions, and feedback
- Build knowledge bases over time
- Enable cross-conversation learning where Claude improves at recurring workflows

How it works

When enabled, Claude automatically checks its memory directory before starting tasks. Claude can create, read, update, and delete files in the `/memories` directory to store what it learns while working, then reference those memories in future conversations to handle similar tasks more effectively or pick up where it left off.

Since this is a client-side tool, Claude makes tool calls to perform memory operations, and your application executes those operations locally. This gives you compl
where and how the memory is stored. For security, you should restrict a
operations to the `/memories` directory.

Ask Docs

Claude Docs

When you ask Claude to help with a task, Claude automatically checks its memory directory first. Here's what a typical interaction looks like:

1. User request:

```
"Help me respond to this customer service ticket."
```

2. Claude checks the memory directory:

```
"I'll help you respond to the customer service ticket. Let me check my me y
```

Claude calls the memory tool:

```
{
  "type": "tool_use",
  "id": "toolu_01C4D5E6F7G8H9I0J1K2L3M4",
  "name": "memory",
  "input": {
    "command": "view",
    "path": "/memories"
  }
}
```

3. Your application returns the directory contents:

```
{
  "type": "tool_result",
  "tool_use_id": "toolu_01C4D5E6F7G8H9I0J1K2L3M4",
  "content": "Here're the files and directories up to 2 levels deep in /memori
}
```

4. Claude reads relevant files:

Ask Docs

Claude Docs

```
  "id": "toolu_01D5E6F7G8H9I0J1K2L3M4N5",
  "name": "memory",
  "input": {
    "command": "view",
    "path": "/memories/customer_service_guidelines.xml"
  }
}
```

5. Your application returns the file contents:

```
{
  "type": "tool_result",
  "tool_use_id": "toolu_01D5E6F7G8H9I0J1K2L3M4N5",
  "content": "Here's the content of /memories/customer_service_guidelines.xml
}
```

6. Claude uses the memory to help:

```
"Based on your customer service guidelines, I can help you craft a respor
```

Supported models

The memory tool is available on:

- Claude Sonnet 4.5 (`claude-sonnet-4-5-20250929`)
- Claude Sonnet 4 (`claude-sonnet-4-20250514`)
- Claude Haiku 4.5 (`claude-haiku-4-5-20251001`)
- Claude Opus 4.5 (`claude-opus-4-5-20251101`)
- Claude Opus 4.1 (`claude-opus-4-1-20250805`)
- Claude Opus 4 (`claude-opus-4-20250514`)

Getting started

To use the memory tool:

Ask Docs

1. Include the beta header `context-management-2025-06-27` in your API requests

Claude Docs

To handle memory tool operations in your application, you need to implement handlers for each memory command. Our SDKs provide memory tool helpers that handle the tool interface—you can subclass `BetaAbstractMemoryTool` (Python) or use `betaMemoryTool` (TypeScript) to implement your own memory backend (file-based, database, cloud storage, encrypted files, etc.).

For working examples, see:

- Python: examples/memory/basic.py
- TypeScript: examples/tools-helpers-memory.ts

Basic usage

cURL

```
curl https://api.anthropic.com/v1/messages \
    --header "x-api-key: $ANTHROPIC_API_KEY" \
    --header "anthropic-version: 2023-06-01" \
    --header "content-type: application/json" \
    --header "anthropic-beta: context-management-2025-06-27" \
    --data '{
        "model": "claude-sonnet-4-5",
        "max_tokens": 2048,
        "messages": [
            {
                "role": "user",
                "content": "I'\''m working on a Python web scraper that keeps
            }
        ],
        "tools": [{
            "type": "memory_20250818",
            "name": "memory"
        }]
    }'
```

Tool commands

Ask Docs

with, you can modify your implementation and return strings as needed for your use case.

view

Shows directory contents or file contents with optional line ranges:

```
{
  "command": "view",
  "path": "/memories",
  "view_range": [1, 10]  // Optional: view specific lines
}
```

Return values

For directories: Return a listing that shows files and directories with their sizes:

```
Here're the files and directories up to 2 levels deep in {path}, excludir id
{size}    {path}
{size}    {path}/{filename1}
{size}    {path}/{filename2}
```

- Lists files up to 2 levels deep
- Shows human-readable sizes (e.g., `5.5K`, `1.2M`)
- Excludes hidden items (files starting with `.`) and `node_modules`
- Uses tab character between size and path

For files: Return file contents with a header and line numbers:

```
Here's the content of {path} with line numbers:
{line_numbers}{tab}{content}
```

Line number formatting:

- **Width**: 6 characters, right-aligned with space padding
- **Separator**: Tab character between line number and content
- **Indexing**: 1-indexed (first line is line 1)

Example output:

```
Here's the content of /memories/notes.txt with line numbers:
     1  Hello World
     2  This is line two
    10  Line ten
   100  Line one hundred
```

Error handling

- **File/directory does not exist**: `"The path {path} does not exist. Please provide a valid path."`

create

Create a new file:

```
{
  "command": "create",
  "path": "/memories/notes.txt",
  "file_text": "Meeting notes:\n- Discussed project timeline\n- Next steps def
}
```

Return values

- **Success**: `"File created successfully at: {path}"`

Error handling

- **File already exists**: `"Error: File {path} already exists"`

str_replace

Replace text in a file:

```
  "path": "/memories/preferences.txt",
  "old_str": "Favorite color: blue",
  "new_str": "Favorite color: green"
}
```

Return values

- **Success**: `"The memory file has been edited."` followed by a snippet of the edited file with line numbers

Error handling

- **File does not exist**: `"Error: The path {path} does not exist. Please provide a valid path."`
- **Text not found**: `"No replacement was performed, old_str `\{old_str}` did not appear verbatim in {path}."`
- **Duplicate text**: When `old_str` appears multiple times, return: `"No replacement was performed. Multiple occurrences of old_str `\{old_str}` in lines: {line_numbers}. Please ensure it is unique"`

Directory handling

If the path is a directory, return a "file does not exist" error.

insert

Insert text at a specific line:

```
{
  "command": "insert",
  "path": "/memories/todo.txt",
  "insert_line": 2,
  "insert_text": "- Review memory tool documentation\n"
}
```

Return values

- **Success**: `"The file {path} has been edited."`

- **File does not exist**: `"Error: The path {path} does not exist"`
- **Invalid line number**: `"Error: Invalid `insert_line` parameter: {insert_line}. It should be within the range of lines of the file: [0, {n_lines}]"`

Directory handling

If the path is a directory, return a "file does not exist" error.

delete

Delete a file or directory:

```
{
  "command": "delete",
  "path": "/memories/old_file.txt"
}
```

Return values

- **Success**: `"Successfully deleted {path}"`

Error handling

- **File/directory does not exist**: `"Error: The path {path} does not exist"`

Directory handling

Deletes the directory and all its contents recursively.

rename

Rename or move a file/directory:

```
{
  "command": "rename",
  "old_path": "/memories/draft.txt",
  "new_path": "/memories/final.txt"
}
```

- **Success**: `"Successfully renamed {old_path} to {new_path}"`

Error handling

- **Source does not exist**: `"Error: The path {old_path} does not exist"`
- **Destination already exists**: Return an error (do not overwrite): `"Error: The destination {new_path} already exists"`

Directory handling

Renames the directory.

Prompting guidance

We automatically include this instruction to the system prompt when the memory tool is included:

```
IMPORTANT: ALWAYS VIEW YOUR MEMORY DIRECTORY BEFORE DOING ANYTHING ELSE.
MEMORY PROTOCOL:
1. Use the `view` command of your `memory` tool to check for earlier progress.
2. ... (work on the task) ...
     - As you make progress, record status / progress / thoughts etc in your m
ASSUME INTERRUPTION: Your context window might be reset at any moment, so you
```

If you observe Claude creating cluttered memory files, you can include this instruction:

> *Note: when editing your memory folder, always try to keep its content up-to-date, coherent and organized. You can rename or delete files that are no longer relevant. Do not create new files unless necessary.*

You can also guide what Claude writes to memory, e.g., "Only write down information relevant to <topic> in your memory system."

Security considerations

Here are important security concerns when implementing your memory store:

Sensitive information

information.

File storage size

Consider tracking memory file sizes and preventing files from growing too large. Consider adding a maximum number of characters the memory read command can return, and let Claude paginate through contents.

Memory expiration

Consider clearing out memory files periodically that haven't been accessed in an extended time.

Path traversal protection

> ⚠ Malicious path inputs could attempt to access files outside the `/memories` directory. Your implementation **MUST** validate all paths to prevent directory traversal attacks.

Consider these safeguards:

- Validate that all paths start with `/memories`
- Resolve paths to their canonical form and verify they remain within the memory directory
- Reject paths containing sequences like `../` , `..\\` , or other traversal patterns
- Watch for URL-encoded traversal sequences (`%2e%2e%2f`)
- Use your language's built-in path security utilities (e.g., Python's `pathlib.Path.resolve()` and `relative_to()`)

Error handling

The memory tool uses similar error handling patterns to the text editor tool. See the individual tool command sections above for detailed error messages and behaviors. Common errors include file not found, permission errors, invalid paths, and duplicate text matches.

Using with Context Editing

Ask Docs

combination enables long-running agentic workflows that would otherwise exceed context limits.

How they work together

When context editing is enabled and your conversation approaches the clearing threshold, Claude automatically receives a warning notification. This prompts Claude to preserve any important information from tool results into memory files before those results are cleared from the context window.

After tool results are cleared, Claude can retrieve the stored information from memory files whenever needed, effectively treating memory as an extension of its working context. This allows Claude to:

- Continue complex, multi-step workflows without losing critical information
- Reference past work and decisions even after tool results are removed
- Maintain coherent context across conversations that would exceed typical context limits
- Build up a knowledge base over time while keeping the active context window manageable

Example workflow

Consider a code refactoring project with many file operations:

1. Claude makes numerous edits to files, generating many tool results
2. As the context grows and approaches your threshold, Claude receives a warning
3. Claude summarizes the changes made so far to a memory file (e.g., `/memories/refactoring_progress.xml`)
4. Context editing clears the older tool results automatically
5. Claude continues working, referencing the memory file when it needs to recall what changes were already completed
6. The workflow can continue indefinitely, with Claude managing both active context and persistent memory

Configuration

To use both features together:

Claude Docs

```python
response = client.beta.messages.create(
    model="claude-sonnet-4-5",
    max_tokens=4096,
    messages=[...],
    tools=[
        {
            "type": "memory_20250818",
            "name": "memory"
        },
        # Your other tools
    ],
    betas=["context-management-2025-06-27"],
    context_management={
        "edits": [
            {
                "type": "clear_tool_uses_20250919",
                "trigger": {
                    "type": "input_tokens",
                    "value": 100000
                },
                "keep": {
                    "type": "tool_uses",
                    "value": 3
                }
            }
        ]
    }
)
```

You can also exclude memory tool calls from being cleared to ensure Claude always has access to recent memory operations:

Python

Ask Docs

Claude Docs

```
        {
            "type": "clear_tool_uses_20250919",
            "exclude_tools": ["memory"]
        }
    ]
}
```

Claude Docs

Solutions

AI agents

Code modernization

Coding

Customer support

Education

Financial services

Government

Life sciences

Partners

Amazon Bedrock

Google Cloud's Vertex AI

Learn

Blog

Catalog

Courses

Use cases

Connectors

Customer stories

Engineering at Anthropic

Events

Powered by Claude

Service partners

Startups program

Company

Anthropic

Careers

Economic Futures

Research

News

Responsible Scaling Policy

Security and compliance

Transparency

Ask Docs

Claude Docs

Availability

Status

Support

Discord

Terms and policies

Privacy policy

Responsible disclosure policy

Terms of service: Commercial

Terms of service: Consumer

Usage policy

Ask Docs