Text editor tool

Copy page

Claude can use an Anthropic-defined text editor tool to view and modify text files, helping you debug, fix, and improve your code or other text documents. This allows Claude to directly interact with your files, providing hands-on assistance rather than just suggesting changes.

Model compatibility

Model	Tool Version
Claude 4.x models	`text_editor_20250728`
Claude Sonnet 3.7 (deprecated)	`text_editor_20250124`

⚠ The `text_editor_20250728` tool for Claude 4 models does not include the `undo_edit` command. If you require this functionality, you'll need to use Claude Sonnet 3.7 (deprecated).

⚠ Older tool versions are not guaranteed to be backwards-compatible with newer models. Always use the tool version that corresponds to your model version.

When to use the text editor tool

Some examples of when to use the text editor tool are:

- **Code debugging**: Have Claude identify and fix bugs in your code, from syntax errors to logic issues.
- **Code refactoring**: Let Claude improve your code structure, readability, and performance through targeted edits.

Claude Docs

- **Test creation**: Have Claude create unit tests for your code based on its understanding of the implementation.

Use the text editor tool

Claude 4 Claude Sonnet 3.7

Provide the text editor tool (named `str_replace_based_edit_tool`) to Claude using the Messages API.

You can optionally specify a `max_characters` parameter to control truncation when viewing large files.

`max_characters` is only compatible with `text_editor_20250728` and later versions of the text editor tool.

Shell

```
curl https://api.anthropic.com/v1/messages \
  -H "content-type: application/json" \
  -H "x-api-key: $ANTHROPIC_API_KEY" \
  -H "anthropic-version: 2023-06-01" \
  -d '{
    "model": "claude-sonnet-4-5",
    "max_tokens": 1024,
    "tools": [
      {
        "type": "text_editor_20250728",
        "name": "str_replace_based_edit_tool",
        "max_characters": 10000
      }
    ],
    "messages": [
      {
        "role": "user",
        "content": "There'\''s a syntax error in my primes.py file. Can you he
      }
    ]
  }'
```

1 **Provide Claude with the text editor tool and a user prompt**

- Include the text editor tool in your API request
- Provide a user prompt that may require examining or modifying files, such as "Can you fix the syntax error in my code?"

2 **Claude uses the tool to examine files or directories**

- Claude assesses what it needs to look at and uses the `view` command to examine file contents or list directory contents
- The API response will contain a `tool_use` content block with the `view` command

3 **Execute the view command and return results**

- Extract the file or directory path from Claude's tool use request
- Read the file's contents or list the directory contents
- If a `max_characters` parameter was specified in the tool configuration, truncate the file contents to that length
- Return the results to Claude by continuing the conversation with a new `user` message containing a `tool_result` content block

4 **Claude uses the tool to modify files**

- After examining the file or directory, Claude may use a command such as `str_replace` to make changes or `insert` to add text at a specific line number.
- If Claude uses the `str_replace` command, Claude constructs a properly formatted tool use request with the old text and new text to replace it with

5 **Execute the edit and return results**

- Extract the file path, old text, and new text from Claude's tool use request
- Perform the text replacement in the file
- Return the results to Claude

6 **Claude provides its analysis and explanation**

Claude Docs

Text editor tool commands

The text editor tool supports several commands for viewing and modifying files:

view

The `view` command allows Claude to examine the contents of a file or list the contents of a directory. It can read the entire file or a specific range of lines.

Parameters:

- `command` : Must be "view"
- `path` : The path to the file or directory to view
- `view_range` (optional): An array of two integers specifying the start and end line numbers to view. Line numbers are 1-indexed, and -1 for the end line means read to the end of the file. This parameter only applies when viewing files, not directories.

> **Example view commands**

str_replace

The `str_replace` command allows Claude to replace a specific string in a file with a new string. This is used for making precise edits.

Parameters:

- `command` : Must be "str_replace"
- `path` : The path to the file to modify
- `old_str` : The text to replace (must match exactly, including whitespace and indentation)
- `new_str` : The new text to insert in place of the old text

> **Example str_replace command**

create

- `command` : Must be "create"
- `path` : The path where the new file should be created
- `file_text` : The content to write to the new file

> Example create command

insert

The `insert` command allows Claude to insert text at a specific location in a file.

Parameters:

- `command` : Must be "insert"
- `path` : The path to the file to modify
- `insert_line` : The line number after which to insert the text (0 for beginning of file)
- `new_str` : The text to insert

> Example insert command

undo_edit

The `undo_edit` command allows Claude to revert the last edit made to a file.

ⓘ This command is only available in Claude Sonnet 3.7 (deprecated). It is not supported in Claude 4 models using the `text_editor_20250728` .

Parameters:

- `command` : Must be "undo_edit"
- `path` : The path to the file whose last edit should be undone

> Example undo_edit command

Claude Docs

Claude 4 Claude Sonnet 3.7

This example demonstrates how Claude 4 models use the text editor tool to fix a syntax error in a Python file.

First, your application provides Claude with the text editor tool and a prompt to fix a syntax error:

Shell

```
curl https://api.anthropic.com/v1/messages \
  -H "content-type: application/json" \
  -H "x-api-key: $ANTHROPIC_API_KEY" \
  -H "anthropic-version: 2023-06-01" \
  -d '{
    "model": "claude-sonnet-4-5",
    "max_tokens": 1024,
    "tools": [
      {
        "type": "text_editor_20250728",
        "name": "str_replace_based_edit_tool"
      }
    ],
    "messages": [
      {
        "role": "user",
        "content": "There'\''s a syntax error in my primes.py file. Can you he
      }
    ]
  }'
```

Claude will use the text editor tool first to view the file:

Claude Docs

```
    "model": "claude-sonnet-4-5",
    "stop_reason": "tool_use",
    "role": "assistant",
    "content": [
      {
        "type": "text",
        "text": "I'll help you fix the syntax error in your primes.py file. Firs
      },
      {
        "type": "tool_use",
        "id": "toolu_01AbCdEfGhIjKlMnOpQrStU",
        "name": "str_replace_based_edit_tool",
        "input": {
          "command": "view",
          "path": "primes.py"
        }
      }
    ]
  }
```

Your application should then read the file and return its contents to Claude:

Shell

```
    -H "x-api-key: $ANTHROPIC_API_KEY" \
    -H "anthropic-version: 2023-06-01" \
    -d '{
      "model": "claude-sonnet-4-5",
      "max_tokens": 1024,
      "tools": [
        {
          "type": "text_editor_20250728",
          "name": "str_replace_based_edit_tool"
        }
      ],
      "messages": [
        {
          "role": "user",
          "content": "There'\''s a syntax error in my primes.py file. Can you he
        },
        {
              "role": "assistant",
              "content": [
                  {
                      "type": "text",
                      "text": "I'\''ll help you fix the syntax error in your pri
                  },
                  {
                      "type": "tool_use",
                      "id": "toolu_01AbCdEfGhIjKlMnOpQrStU",
                      "name": "str_replace_based_edit_tool",
                      "input": {
                          "command": "view",
                          "path": "primes.py"
                      }
                  }
              ]
          },
          {
              "role": "user",
              "content": [
                  {
                      "type": "tool_result",
                      "tool_use_id": "toolu_01AbCdEfGhIjKlMnOpQrStU",
                      "content": "1: def is_prime(n):\n2:     \"\"\"Check if a n
                  }
              ]
          }
      ]
    }'
```

Line numbers

In the example above, the `view` tool result includes file contents with line numbers prepended to each line (e.g., "1: def is_prime(n):"). Line numbers are not required, but they are essential for successfully using the `view_range` parameter to examine specific sections of files and the `insert_line` parameter to add content at precise locations.

Claude will identify the syntax error and use the `str_replace` command to fix it:

```
{
  "id": "msg_01VwXyZAbCdEfGhIjKlMnO",
  "model": "claude-sonnet-4-5",
  "stop_reason": "tool_use",
  "role": "assistant",
  "content": [
    {
      "type": "text",
      "text": "I found the syntax error in your primes.py file. In the `get_pr
    },
    {
      "type": "tool_use",
      "id": "toolu_01PqRsTuVwXyZAbCdEfGh",
      "name": "str_replace_based_edit_tool",
      "input": {
        "command": "str_replace",
        "path": "primes.py",
        "old_str": "    for num in range(2, limit + 1)",
        "new_str": "    for num in range(2, limit + 1):"
      }
    }
  ]
}
```

Your application should then make the edit and return the result:

Python

Claude Docs

```
    max_tokens=1024,
    tools=[
        {
            "type": "text_editor_20250728",
            "name": "str_replace_based_edit_tool"
        }
    ],
    messages=[
        # Previous messages...
        {
            "role": "assistant",
            "content": [
                {
                    "type": "text",
                    "text": "I found the syntax error in your primes.py file.
                },
                {
                    "type": "tool_use",
                    "id": "toolu_01PqRsTuVwXyZAbCdEfGh",
                    "name": "str_replace_based_edit_tool",
                    "input": {
                        "command": "str_replace",
                        "path": "primes.py",
                        "old_str": "    for num in range(2, limit + 1)",
                        "new_str": "    for num in range(2, limit + 1):"
                    }
                }
            ]
        },
        {
            "role": "user",
            "content": [
                {
                    "type": "tool_result",
                    "tool_use_id": "toolu_01PqRsTuVwXyZAbCdEfGh",
                    "content": "Successfully replaced text at exactly one loca
                }
            ]
        }
    ]
)
```

Finally, Claude will provide a complete explanation of the fix:

```
    "model": "claude-sonnet-4-5",
    "stop_reason": "end_turn",
    "role": "assistant",
    "content": [
      {
        "type": "text",
        "text": "I've fixed the syntax error in your primes.py file. The issue w
      }
    ]
  }
```

Implement the text editor tool

The text editor tool is implemented as a schema-less tool. When using this tool, you don't need to provide an input schema as with other tools; the schema is built into Claude's model and can't be modified.

The tool type depends on the model version:

- **Claude 4**: `type: "text_editor_20250728"`
- **Claude Sonnet 3.7**: `type: "text_editor_20250124"`

1 **Initialize your editor implementation**

Create helper functions to handle file operations like reading, writing, and modifying files. Consider implementing backup functionality to recover from mistakes.

2 **Handle editor tool calls**

Create a function that processes tool calls from Claude based on the command type:

Claude Docs

```
    command = input_params.get('command', '')
    file_path = input_params.get('path', '')

    if command == 'view':
        # Read and return file contents
        pass
    elif command == 'str_replace':
        # Replace text in file
        pass
    elif command == 'create':
        # Create new file
        pass
    elif command == 'insert':
        # Insert text at location
        pass
    elif command == 'undo_edit':
        # Check if it's a Claude 4 model
        if 'str_replace_based_edit_tool' in model_version:
            return {"error": "undo_edit command is not supported in Clau
        # Restore from backup for Claude 3.7
        pass
```

3 **Implement security measures**

Add validation and security checks:

- Validate file paths to prevent directory traversal
- Create backups before making changes
- Handle errors gracefully
- Implement permissions checks

4 **Process Claude's responses**

Extract and handle tool calls from Claude's responses:

```
    if content.type == "tool_use":
        # Execute the tool based on command
        result = handle_editor_tool(content)

        # Return result to Claude
        tool_result = {
            "type": "tool_result",
            "tool_use_id": content.id,
            "content": result
        }
```

⚠ When implementing the text editor tool, keep in mind:

1. **Security**: The tool has access to your local filesystem, so implement proper security measures.
2. **Backup**: Always create backups before allowing edits to important files.
3. **Validation**: Validate all inputs to prevent unintended changes.
4. **Unique matching**: Make sure replacements match exactly one location to avoid unintended edits.

Handle errors

When using the text editor tool, various errors may occur. Here is guidance on how to handle them:

> **File not found**

> **Multiple matches for replacement**

> **No matches for replacement**

> **Permission errors**

Follow implementation best practices

> **Be explicit about file paths**

> **Create backups before editing**

> **Handle unique text replacement carefully**

> **Verify changes**

Pricing and token usage

The text editor tool uses the same pricing structure as other tools used with Claude. It follows the standard input and output token pricing based on the Claude model you're using.

In addition to the base tokens, the following additional input tokens are needed for the text editor tool:

Tool	Additional input tokens
`text_editor_20250429` (Claude 4.x)	700 tokens
`text_editor_20250124` (Claude Sonnet 3.7 (deprecated))	700 tokens

For more detailed information about tool pricing, see Tool use pricing.

Integrate the text editor tool with other tools

The text editor tool can be used alongside other Claude tools. When combining tools, ensure you:

- Match the tool version with the model you're using
- Account for the additional token usage for all tools included in your request

Change log

Claude Docs

July 28, 2025	`text_editor_20250728`	issues and adds an optional `max_characters` parameter. It is otherwise identical to `text_editor_20250429` .
April 29, 2025	`text_editor_20250429`	Release of the text editor Tool for Claude 4. This version removes the `undo_edit` command but maintains all other capabilities. The tool name has been updated to reflect its str_replace-based architecture.
March 13, 2025	`text_editor_20250124`	Introduction of standalone text editor Tool documentation. This version is optimized for Claude Sonnet 3.7 but has identical capabilities to the previous version.
October 22, 2024	`text_editor_20241022`	Initial release of the text editor Tool with Claude Sonnet 3.5 (retired). Provides capabilities for viewing, creating, and editing files through the `view` , `create` , `str_replace` , `insert` , and `undo_edit` commands.

Next steps

Here are some ideas for how to use the text editor tool in more convenient and powerful ways:

- **Integrate with your development workflow**: Build the text editor tool into your development tools or IDE
- **Create a code review system**: Have Claude review your code and make improvements
- **Build a debugging assistant**: Create a system where Claude can help you diagnose and fix issues in your code
- **Implement file format conversion**: Let Claude help you convert files from one format to another
- **Automate documentation**: Set up workflows for Claude to automatically document your code

As you build applications with the text editor tool, we're excited to see how you leverage Claude's capabilities to enhance your development workflow and productivity.

Tool use overview

Learn how to implement tool workflows for use with Claude.

Claude Docs

Bash tool

Execute shell commands with Claude.

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

Help and security

Claude Docs

Support

Discord

Terms and policies

Privacy policy

Responsible disclosure policy

Terms of service: Commercial

Terms of service: Consumer

Usage policy