

ANTHROPIC

WELCOME TO THE COURSE

Introducing MCP

MCP clients

Hands-on with MCP servers

Project setup

Defining tools with MCP

The server inspector

Course satisfaction survey

Connecting with MCP clients

Implementing a client

Defining resources

Accessing resources

Defining prompts

Prompts in the client

Assessment and wrap up

Final assessment on MCP

MCP review

Anthropic Academy

Courses

Defining prompts

AI Open in Claude

Our Server

MCP Client

MCP Server

Tools

Resources

Prompts

Outside Service

MCP Server

Tools

Resources

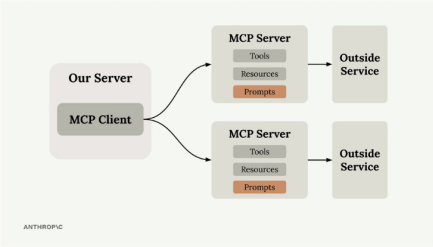
Prompts

Outside Service

Previous

Next

Prompts in MCP servers let you define pre-built, high-quality instructions that clients can use instead of writing their own prompts from scratch. Think of them as carefully crafted templates that give better results than what users might come up with on their own.



ANTHROPIC

Next

Next

Here's the key insight: users can already ask Claude to do most tasks directly. For example, a user could type "reformat the report.pdf in markdown" and get decent results. But they'll get much better results if you provide a thoroughly tested, specialized prompt that handles edge cases and follows best practices.

As the MCP server author, you can spend time crafting, testing, and evaluating prompts that work consistently across different scenarios. Users benefit from this expertise without having to become prompt engineering experts themselves.

If we left this process up to a user, here's what they'd write:

```
Convert report.pdf to markdown
```

Yes, it'd work, but the user might get a better result with some strong prompt engineering

User might have more luck if they use our thoroughly-eval'd prompt instead!

```
@mcp.prompt(
    name="format",
    description="Rewrites the contents of the document in Markdown format."
)
def format_document(
    doc_id: str = Field(description="Id of the document to format")
) -> list[base.Message]:
    prompt = f"""
    Your goal is to reformat a document to be written with markdown syntax.
    """
    return [
        base.Message(
            role="assistant",
            content=prompt,
        )
    ]
```

ANTHROPIC

Next

Next

Building a `format` Command

Let's implement a practical example: a `format` command that converts documents to markdown. Users will type `/format doc_id` and get back a professionally formatted markdown version of their document.

The workflow looks like this:

- User types `/` to see available commands
- They select `format` and specify a document ID
- Claude uses your pre-built prompt to read and reformat the document
- The result is clean markdown with proper headers, lists, and formatting

Defining Prompts

Prompts use a similar decorator pattern to tools and resources:

```
@mcp.prompt(
    name="format",
    description="Rewrites the contents of the document in Markdown format."
)
def format_document(
    doc_id: str = Field(description="Id of the document to format")
) -> list[base.Message]:
    prompt = f"""
    Your goal is to reformat a document to be written with markdown syntax.
    """
    return [
        base.Message(
            role="assistant",
            content=prompt,
        )
    ]
```

THE END OF THE COURSE. YOU HAVE NOW COMPLETED ALL THE

<document_id>

```
{doc_id}
</document_id>

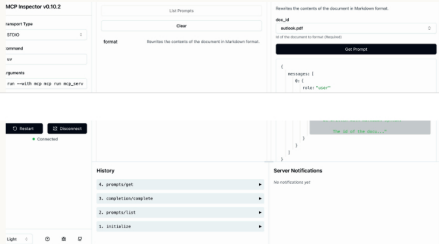
Add in headers, bullet points, tables, etc as necessary. Feel free to add in st
Use the 'edit_document' tool to edit the document. After the document has been
***

return [
  base.UserMessage(prompt)
]
```

The function returns a list of messages that get sent directly to Claude. You can include multiple user and assistant messages to create more complex conversation flows.

Testing Your Prompts

Use the MCP Inspector to test your prompts before deploying them:



The inspector shows you exactly what messages will be sent to Claude, including how variables get interpolated into your prompt template. This lets you verify the prompt looks correct before users start relying on it.

Key Benefits

- Consistency - Users get reliable results every time
- Expertise - You can encode domain knowledge into prompts
- Reusability - Multiple client applications can use the same prompts
- Maintenance - Update prompts in one place to improve all clients

Prompts work best when they're specialized for your MCP server's domain. A document management server might have prompts for formatting, summarizing, or analyzing documents. A data analysis server might have prompts for generating reports or visualizations.

The goal is to provide prompts that are so well-crafted and tested that users prefer them over writing their own instructions from scratch.