Concepts

Sampling

Let your servers request completions from LLMs

Sampling is a powerful MCP feature that allows servers to request LLM completions through the client, enabling sophisticated agentic behaviors while maintaining security and privacy.

This feature of MCP is not yet supported in the Claude Desktop client.

How sampling works

The sampling flow follows these steps:

1. Server sends a `sampling/createMessage` request to the client
2. Client reviews the request and can modify it
3. Client samples from an LLM
4. Client reviews the completion
5. Client returns the result to the server

This human-in-the-loop design ensures users maintain control over what the LLM sees and generates.

Message format

Sampling requests use a standardized message format:

```
{
  messages: [
    {
      role: "user" | "assistant",
      content: {
        type: "text" | "image",

        // For text:
        text?: string,

        // For images:
        data?: string,             // base64 encoded
        mimeType?: string
      }
    }
  ],
  modelPreferences?: {
    hints?: [{
      name?: string                // Suggested model name/family
    }],
    costPriority?: number,         // 0-1, importance of minimizing cost
    speedPriority?: number,        // 0-1, importance of low latency
    intelligencePriority?: number  // 0-1, importance of capabilities
  },
  systemPrompt?: string,
  includeContext?: "none" | "thisServer" | "allServers",
  temperature?: number,
  maxTokens: number,
  stopSequences?: string[],
  metadata?: Record<string, unknown>
}
```

Request parameters

Messages

The `messages` array contains the conversation history to send to the LLM. Each message has:

- `role`: Either “user” or “assistant”
- `content`: The message content, which can be:
  - Text content with a `text` field
  - Image content with `data` (base64) and `mimeType` fields

Model preferences

The `modelPreferences` object allows servers to specify their model selection preferences:

- `hints`: Array of model name suggestions that clients can use to select an appropriate model:
  - `name`: String that can match full or partial model names (e.g. “claude-3”, “sonnet”)
  - Clients may map hints to equivalent models from different providers
  - Multiple hints are evaluated in preference order
- Priority values (0-1 normalized):
  - `costPriority`: Importance of minimizing costs
  - `speedPriority`: Importance of low latency response
  - `intelligencePriority`: Importance of advanced model capabilities

Clients make the final model selection based on these preferences and their available models.

System prompt

An optional `systemPrompt` field allows servers to request a specific system prompt. The client may modify or ignore this.

Context inclusion

The `includeContext` parameter specifies what MCP context to include:

- `"none"` : No additional context
- `"thisServer"` : Include context from the requesting server
- `"allServers"` : Include context from all connected MCP servers

The client controls what context is actually included.

Sampling parameters

Fine-tune the LLM sampling with:

- `temperature` : Controls randomness (0.0 to 1.0)
- `maxTokens` : Maximum tokens to generate
- `stopSequences` : Array of sequences that stop generation
- `metadata` : Additional provider-specific parameters

Response format

The client returns a completion result:

```
{
  model: string,  // Name of the model used
  stopReason?: "endTurn" | "stopSequence" | "maxTokens" | string,
  role: "user" | "assistant",
  content: {
    type: "text" | "image",
    text?: string,
    data?: string,
    mimeType?: string
  }
}
```

Example request

Here's an example of requesting sampling from a client:

```json
{
  "method": "sampling/createMessage",
  "params": {
    "messages": [
      {
        "role": "user",
        "content": {
          "type": "text",
          "text": "What files are in the current directory?"
        }
      }
    ],
    "systemPrompt": "You are a helpful file system assistant.",
    "includeContext": "thisServer",
    "maxTokens": 100
  }
}
```

Best practices

When implementing sampling:

1. Always provide clear, well-structured prompts
2. Handle both text and image content appropriately
3. Set reasonable token limits
4. Include relevant context through `includeContext`
5. Validate responses before using them
6. Handle errors gracefully
7. Consider rate limiting sampling requests
8. Document expected sampling behavior
9. Test with various model parameters

10. Monitor sampling costs

Human in the loop controls

Sampling is designed with human oversight in mind:

For prompts

- Clients should show users the proposed prompt
- Users should be able to modify or reject prompts
- System prompts can be filtered or modified
- Context inclusion is controlled by the client

For completions

- Clients should show users the completion
- Users should be able to modify or reject completions
- Clients can filter or modify completions
- Users control which model is used

Security considerations

When implementing sampling:

- Validate all message content
- Sanitize sensitive information
- Implement appropriate rate limits
- Monitor sampling usage
- Encrypt data in transit
- Handle user data privacy

- Audit sampling requests
- Control cost exposure
- Implement timeouts
- Handle model errors gracefully

Common patterns

Agentic workflows

Sampling enables agentic patterns like:

- Reading and analyzing resources
- Making decisions based on context
- Generating structured data
- Handling multi-step tasks
- Providing interactive assistance

Context management

Best practices for context:

- Request minimal necessary context
- Structure context clearly
- Handle context size limits
- Update context as needed
- Clean up stale context

Error handling

Robust error handling should:

- Catch sampling failures

- Handle timeout errors
- Manage rate limits
- Validate responses
- Provide fallback behaviors
- Log errors appropriately

Limitations

Be aware of these limitations:

- Sampling depends on client capabilities
- Users control sampling behavior
- Context size has limits
- Rate limits may apply
- Costs should be considered
- Model availability varies
- Response times vary
- Not all content types supported

Was this page helpful? Yes No

< Tools Roots >