Specification > Server Features > Tools

Tools

ℹ️ **Protocol Revision**: 2024-11-05

The Model Context Protocol (MCP) allows servers to expose tools that can be invoked by language models. Tools enable models to interact with external systems, such as querying databases, calling APIs, or performing computations. Each tool is uniquely identified by a name and includes metadata describing its schema.

User Interaction Model

Tools in MCP are designed to be **model-controlled**, meaning that the language model can discover and invoke tools automatically based on its contextual understanding and the user's prompts.

However, implementations are free to expose tools through any interface pattern that suits their needs—the protocol itself does not mandate any specific user interaction model.

⚠️ For trust & safety and security, there **SHOULD** always be a human in the loop with the ability to deny tool invocations.

Applications **SHOULD**:

- Provide UI that makes clear which tools are being exposed to the AI model
- Insert clear visual indicators when tools are invoked
- Present confirmation prompts to the user for operations, to ensure a human is in the loop

Capabilities

Servers that support tools **MUST** declare the `tools` capability:

```json
{
  "capabilities": {
    "tools": {
      "listChanged": true
    }
  }
}
```

`listChanged` indicates whether the server will emit notifications when the list of available tools changes.

Protocol Messages

Listing Tools

To discover available tools, clients send a `tools/list` request. This operation supports pagination.

Request:

```json
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/list",
  "params": {
    "cursor": "optional-cursor-value"
  }
}
```

Response:

```json
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "tools": [
      {
        "name": "get_weather",
        "description": "Get current weather information for a location",
        "inputSchema": {
          "type": "object",
          "properties": {
            "location": {
              "type": "string",
              "description": "City name or zip code"
            }
          },
          "required": ["location"]
        }
      }
    ],
    "nextCursor": "next-page-cursor"
  }
}
```

Calling Tools

To invoke a tool, clients send a `tools/call` request:

Request:

```json
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "tools/call",
  "params": {
    "name": "get_weather",
    "arguments": {
      "location": "New York"
    }
  }
}
```

Response:

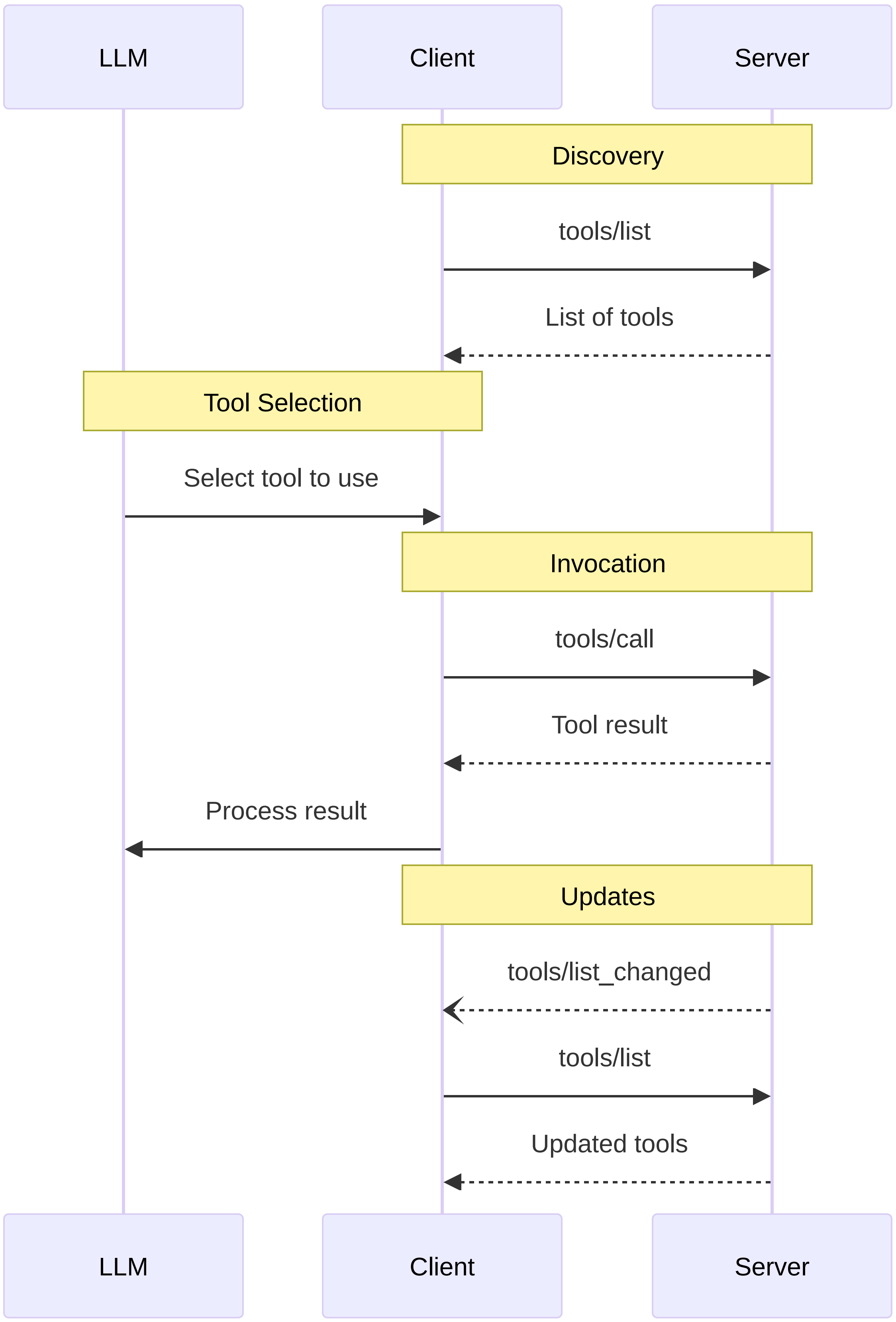
```
{
  "jsonrpc": "2.0",
  "id": 2,
  "result": {
    "content": [{
      "type": "text",
      "text": "Current weather in New York:\nTemperature: 72°F\nConditions: Partly clou
    }],
    "isError": false
  }
}
```

List Changed Notification

When the list of available tools changes, servers that declared the `listChanged` capability **SHOULD** send a notification:

```
{
  "jsonrpc": "2.0",
  "method": "notifications/tools/list_changed"
}
```

Message Flow


LLM
Client
Server
Discovery
tools/list
List of tools
Tool Selection
Select tool to use
Invocation
tools/call
Tool result
Process result
Updates
tools/list_changed
tools/list
Updated tools
LLM
Client
Server

Data Types

Tool

A tool definition includes:

- `name`: Unique identifier for the tool
- `description`: Human-readable description of functionality
- `inputSchema`: JSON Schema defining expected parameters

Tool Result

Tool results can contain multiple content items of different types:

Text Content

```json
{
  "type": "text",
  "text": "Tool result text"
}
```

Image Content

```json
{
  "type": "image",
  "data": "base64-encoded-data",
  "mimeType": "image/png"
}
```

Embedded Resources

Resources **MAY** be embedded, to provide additional context or data, behind a URI that can be subscribed to or fetched again by the client later:

```json
{
  "type": "resource",
  "resource": {
    "uri": "resource://example",
    "mimeType": "text/plain",
    "text": "Resource content"
  }
}
```

Error Handling

Tools use two error reporting mechanisms:

1. **Protocol Errors**: Standard JSON-RPC errors for issues like:
   - Unknown tools
   - Invalid arguments
   - Server errors
2. **Tool Execution Errors**: Reported in tool results with `isError: true`:
   - API failures
   - Invalid input data
   - Business logic errors

Example protocol error:

```
{
  "jsonrpc": "2.0",
  "id": 3,
  "error": {
    "code": -32602,
    "message": "Unknown tool: invalid_tool_name"
  }
}
```

Example tool execution error:

```
{
  "jsonrpc": "2.0",
  "id": 4,
  "result": {
    "content": [{
      "type": "text",
      "text": "Failed to fetch weather data: API rate limit exceeded"
    }],
    "isError": true
  }
}
```

Security Considerations

1. Servers **MUST**:
   - Validate all tool inputs
   - Implement proper access controls
   - Rate limit tool invocations
   - Sanitize tool outputs
2. Clients **SHOULD**:
   - Prompt for user confirmation on sensitive operations
   - Show tool inputs to the user before calling the server, to avoid malicious or accidental data exfiltration
   - Validate tool results before passing to LLM
   - Implement timeouts for tool calls
   - Log tool usage for audit purposes