<epam>

MCP: Deep dive

JUNE 2025

DIALX COMMUNITY
POWERED BY <epam>

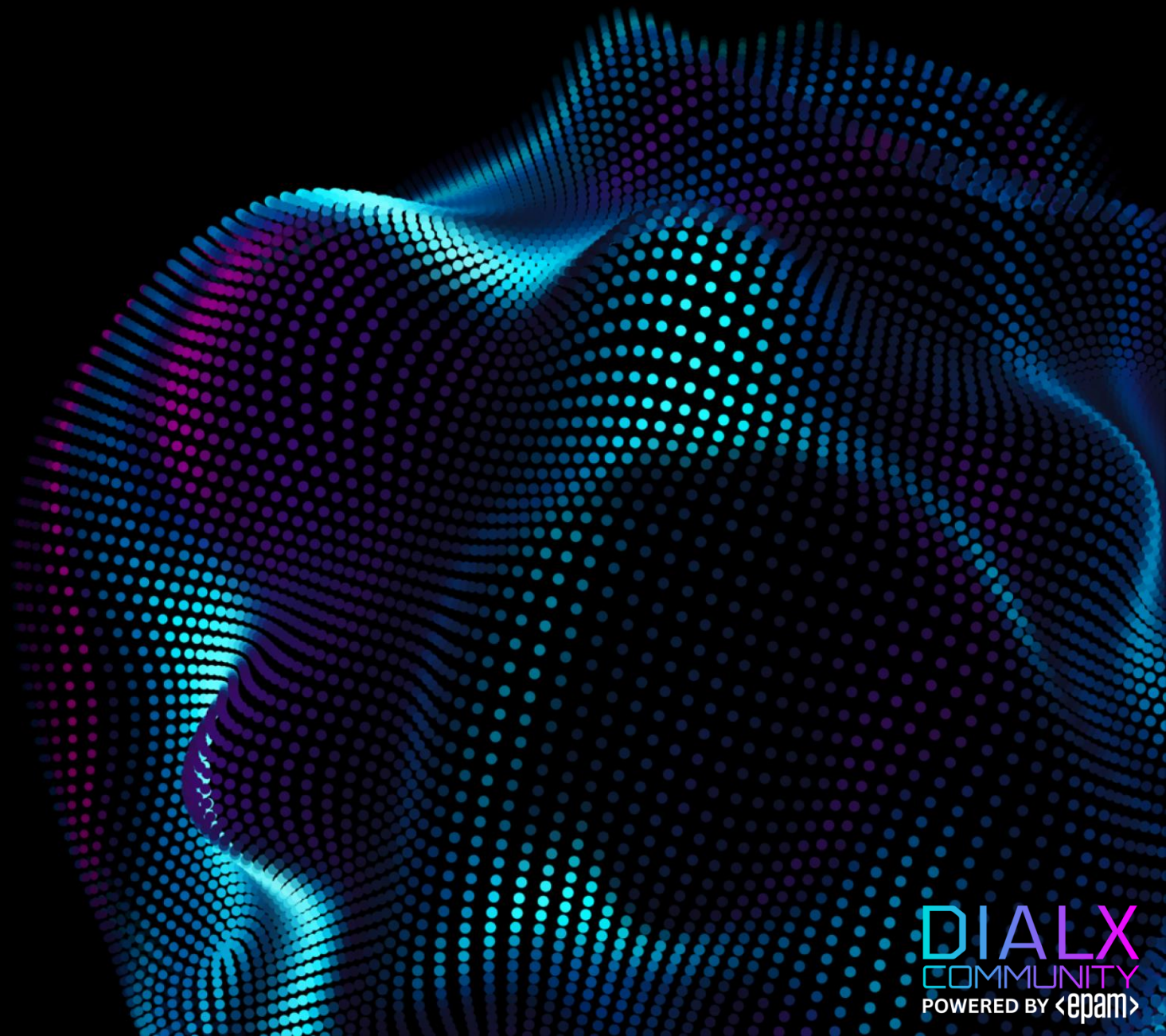
Before we start:

- Raise your hand and ask questions if you have any
- It is better to ask questions when you have
- Also, type them in chat
- We will need DIAL API key for this session

Agenda:

- **Presentation:**
  - **About workflow**
  - **JSON-RPC protocol overview**
  - **Request/response flow**
- **Workshop:**
  - **Provide Web Search tool JSON Schema**
  - **Finish MCPServer creation**
  - **Finish endpoint for MCP Server flow**
  - **Run MCP Server**
  - **Test it with Postman**
  - **Test it with MCPClient**
  - **Write Custom MCP Client**

Workflow



Workflow, Key details:

Base Protocol	• JSON-RPC (Remote Procedure Call) message format • Exchange and negotiate capabilities • Share implementation details
Tools	Functions for the AI model to execute
Resources	Context and data, for the user or the AI model to use (not covered in this practice)
Prompts	Templated messages and workflows for users (not covered in this practice)
Auth	The Model Context Protocol provides authorization capabilities at the transport level, enabling MCP clients to make requests to restricted MCP servers on behalf of resource owners. (not covered in this practice)

https://modelcontextprotocol.io/specification/2025-06-18/basic

https://modelcontextprotocol.io/specification/2025-06-18

Workflow:

1. Initialization

The initialization phase MUST be the first interaction between client and server. During this phase, the client and server:

- **Establish protocol version compatibility**
- **Exchange and negotiate capabilities**
- **Share implementation details**

2. Notification

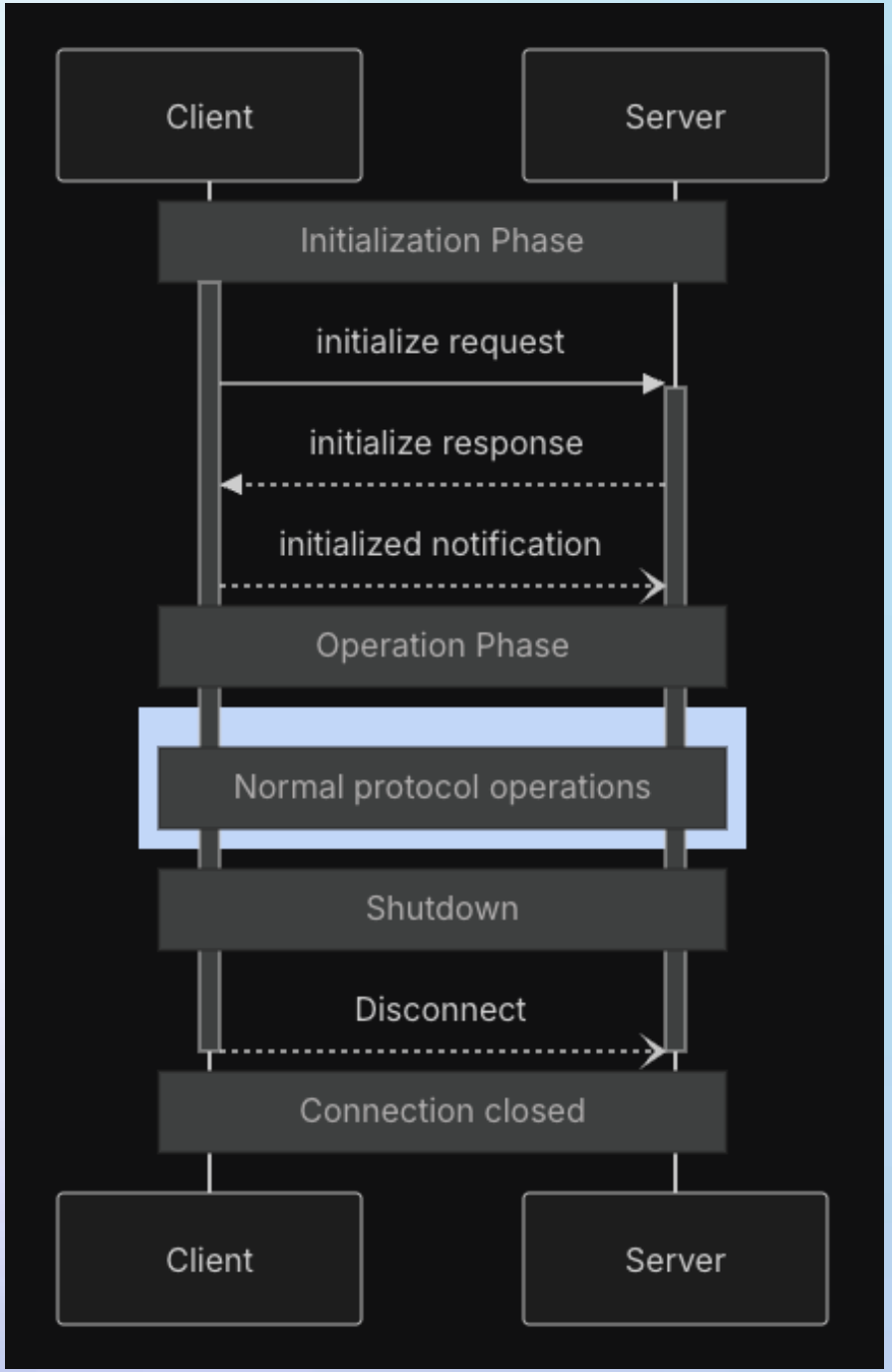
After successful initialization, the client MUST send an initialized notification to indicate it is ready to begin normal operations:

3. Operation

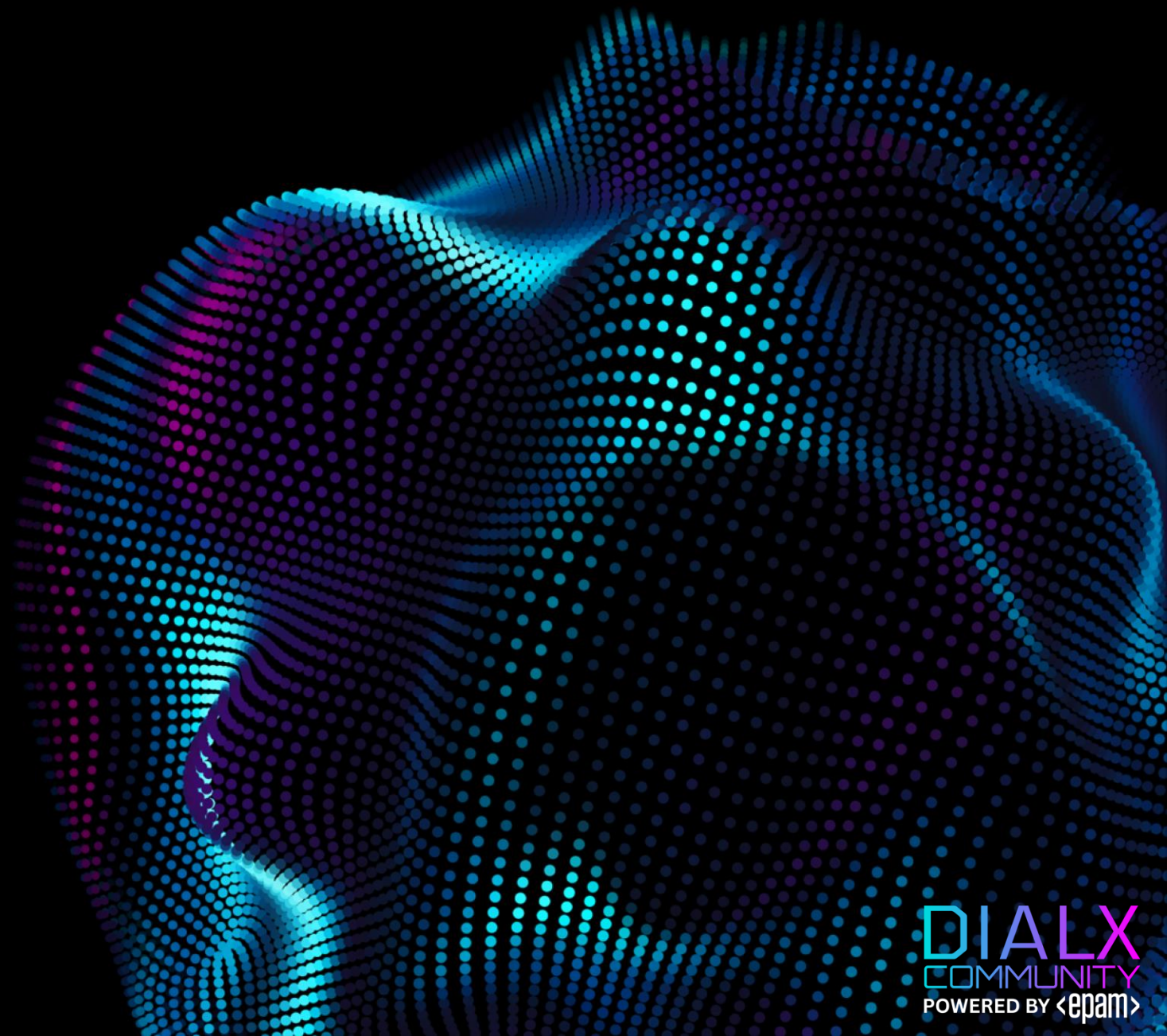
During the operation phase, the client and server exchange messages according to the negotiated capabilities.

4. Shutdown

During the shutdown phase, one side (usually the client) cleanly terminates the protocol connection. No specific shutdown messages are defined—instead, the underlying transport mechanism should be used to signal connection termination.



JSON-RPC protocol



What is JSON-RPC protocol

JSON-RPC is a stateless, light-weight remote procedure call (RPC) protocol, that uses JSON (RFC 4627) as data format.

Reason?

- **Remote method calls:** Allows clients to call methods on remote servers
- **API communication:** Enables communication between different applications/services
- **Cross-platform integration:** Works across different programming languages and platforms

Request

```
{
 "jsonrpc": "2.0",
 "method": "methodName",
 "params": [parameters] or {parameters},
 "id": 1
}
```

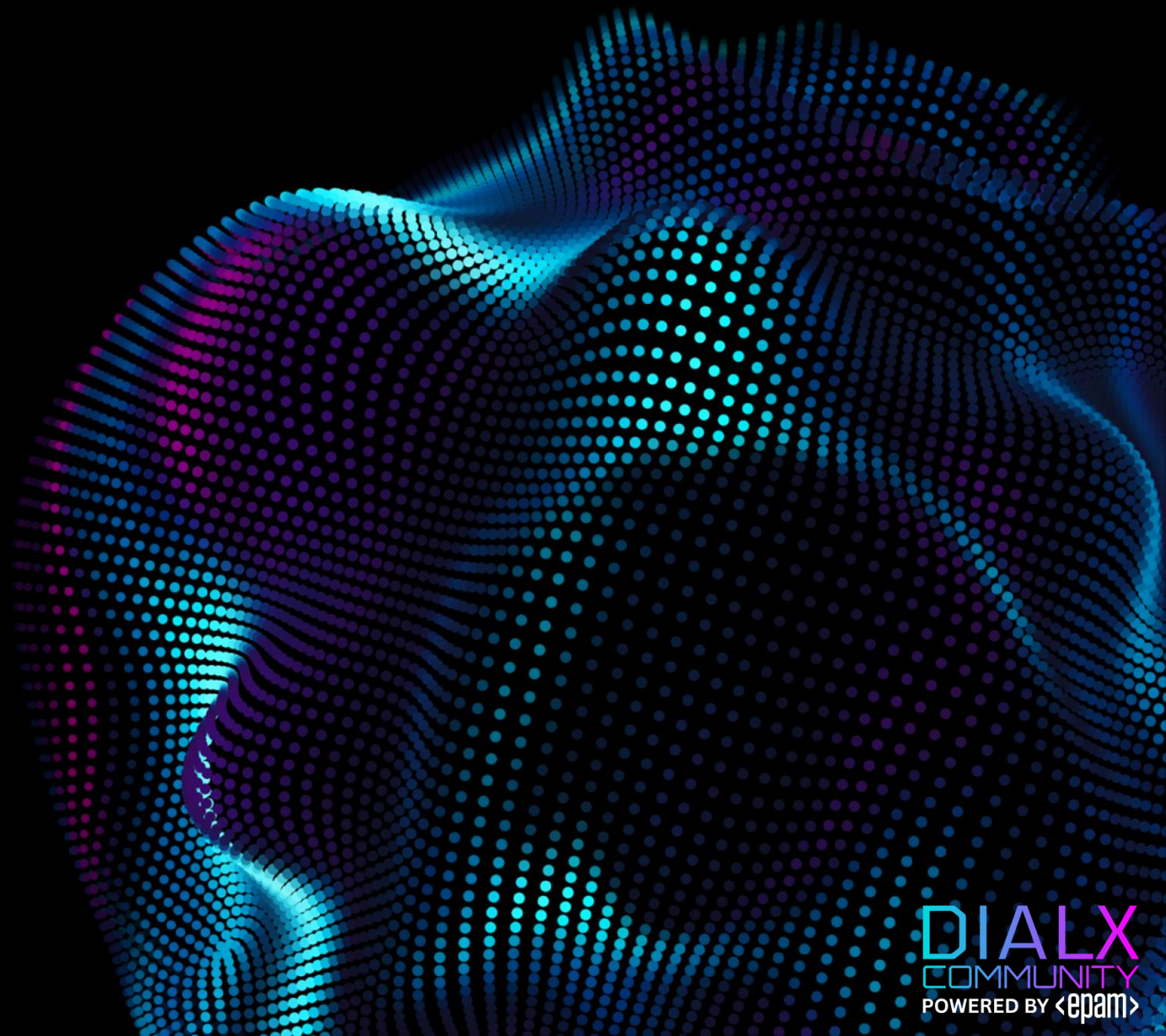
Response

```
{
 "jsonrpc ": "2.0",
 "id": 1,
 "result": {
   [key: string]: unknown
 }
}
```

Error

```
{
 "jsonrpc": "2.0",
 "id": 1,
 "error": {
   "code": -32600,
   "message": "invalid request”,
   "data": "additional info”,
 }
}
```

Request/ response flow



Initialize:

What happens during initialization:
The initialization phase is the first interaction between client and server, establishing protocol version and capabilities

Request Required Parameters:

- **header:** Accept with json and event-stream
- **protocolVersion:** Client MUST send protocol version it supports (latest preferred)
- **capabilities:** Declares what features client supports
- **clientInfo:** Name and version identification

Response Required Parameters:

- **header:** Mcp-Session-Id with session ID
- **protocolVersion:** Client MUST send protocol version it supports (latest preferred)
- **capabilities:** Declares what features server supports
- **serverInfo:** Name and version identification

Accept: application/json, text/event-stream

```
{
"jsonrpc": "2.0",
 "id": 1,
 "method": "initialize",
 "params": {
  "protocolVersion": "2024-11-05",
  "capabilities": {
   // client capabilities
  },
  "clientInfo": {
   "name": "client-name",
   "version": "1.0.0"
}
```

Mcp-Session-Id: session_id

```
{
 "jsonrpc": "2.0",
 "id": 1,
 "result": {
  "protocolVersion": "2024-11-05",
  "capabilities": {
   // server capabilities
  },
  "serverInfo": {
   "name": "server-name",
   "version": "1.0.0"
}
```

Notification:

Purpose:
This handshake requires one last message from the client to the server acknowledging the server initialization message

It's a mandatory acknowledgment that completes the 3-step initialization handshake and enables all subsequent MCP operations.

Request

```
Accept: application/json, text/event-stream
Mcp-Session-Id: Session ID
{
  "jsonrpc": "2.0",
  "method": "notifications/initialized"
}
```

Response

```
Mcp-Session-Id: Session ID

Status: 204 Created
```

Discovery:

Clients requests what capabilities (Tools, Resources, Prompts) the server offers. The Server responds with a list and descriptions

Pay attention that *tools/list* returns JSON Schema with anthropic tools configuration. For DIAL you need to map it properly.

With this request a Client will get list with tools, prompts, or resources.
Today we will practice only with *tools*.

```
Accept: application/json, text/event-stream
Mcp-Session-Id: Session ID

{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "[tools|prompts|resources]/list",
  "params": {}
}
```

```
Mcp-Session-Id: Session ID
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "tools": [
      {
        "name": "tool_name",
        "description": "Tool description",
        "inputSchema": {
          "type": "object",
          "properties": {
            "param1": {
              "type": "string",
              "description": "Parameter description"
            }
          },
          "required": ["param1"]
        }
```

Tools/call:

If the LLM determines it needs to use a Tool, the Host directs the Client to send an invocation request to the appropriate Server

- **What happens:**

- **Tool invocation:** The Client invokes capabilities based on the Host's needs
- **Execution:** The Server receives the request, executes the underlying logic (calls the API), and gets the result
- **Response:** The Server sends the result back to the Client

Tools/call:

Request

```
{
 "jsonrpc": "2.0",
 "id": 1,
 "method": "tools/call",
 "params": {
  "name": "tool_name",
  "arguments": {
   "param1": "value1",
   "param2": "value2"
  }
 }
}
```

Success response

```
{
 "jsonrpc": "2.0",
 "id": 1,
 "result": {
  "content": [
   {
    "type": "text",
    "text": " Result"
   }
  ],
  "isError": false
 }
}
```

Error response

```
{
 "jsonrpc": "2.0",
 "id": 1,
 "result": {
  "content": [
   {
    "type": "text",
    "text": "Error message
describing what went
wrong"
   }
  ],
  "isError": true
 }
}
```

Error response

```
{
 "jsonrpc": "2.0",
 "id": 1,
 "error": {
  "code": -32602,
  "message": "Invalid parameters",
  "data": {
   "details": "Missing required
argument 'param1'"
  }
 }
}
```

DIALX COMMUNITY
POWERED BY epam

Join us:

Subscribe to WeAreCommuntiy

https://wearecommunity.io/communities/dial

Keep in touch with our latest updates. Here you find webinars, workshops and articles about DIALX features and products.

Subscribe to YouTube

https://www.youtube.com/@TeamDIALX

Here we publish videos about our newest products and features.

Join our Discord community

https://discord.gg/jvTCQv4E4q

✨AI DIALX Community✨ is the place where you can find help with your questions about DIALX, direct communication with DIALX team and contributors.


DIALX
COMMUNITY
POWERED BY epam
Thank you!