Resources

ℹ️ **Protocol Revision**: 2024-11-05

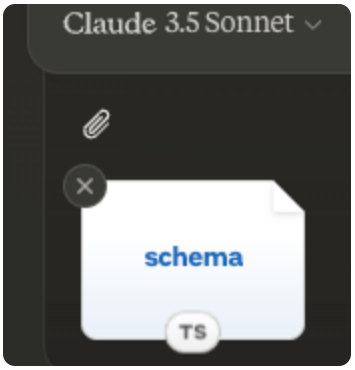
The Model Context Protocol (MCP) provides a standardized way for servers to expose resources to clients. Resources allow servers to share data that provides context to language models, such as files, database schemas, or application-specific information. Each resource is uniquely identified by a URI.

User Interaction Model

Resources in MCP are designed to be **application-driven**, with host applications determining how to incorporate context based on their needs.

For example, applications could:

- Expose resources through UI elements for explicit selection, in a tree or list view
- Allow the user to search through and filter available resources
- Implement automatic context inclusion, based on heuristics or the AI model's selection



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

Capabilities

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

```
{
  "capabilities": {
    "resources": {
      "subscribe": true,
      "listChanged": true
    }
  }
}
```

The capability supports two optional features:

- `subscribe`: whether the client can subscribe to be notified of changes to individual resources.
- `listChanged`: whether the server will emit notifications when the list of available resources changes.

Both `subscribe` and `listChanged` are optional—servers can support neither, either, or both:

```
{
  "capabilities": {
    "resources": {}  // Neither feature supported
  }
}
```

```
{
  "capabilities": {
    "resources": {
      "subscribe": true  // Only subscriptions supported
    }
  }
}
```

```
{
  "capabilities": {
    "resources": {
      "listChanged": true  // Only list change notifications supported
    }
  }
}
```

Protocol Messages

Listing Resources

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

Request:

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

Response:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "resources": [
      {
        "uri": "file:///project/src/main.rs",
        "name": "main.rs",
        "description": "Primary application entry point",
        "mimeType": "text/x-rust"
```

```
        }
      ],
      "nextCursor": "next-page-cursor"
    }
}
```

Reading Resources

To retrieve resource contents, clients send a `resources/read` request:

Request:

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "resources/read",
  "params": {
    "uri": "file:///project/src/main.rs"
  }
}
```

Response:

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "result": {
    "contents": [
      {
        "uri": "file:///project/src/main.rs",
        "mimeType": "text/x-rust",
        "text": "fn main() {\n    println!(\"Hello world!\");\n}"
      }
    ]
  }
}
```

Resource Templates

Resource templates allow servers to expose parameterized resources using URI templates. Arguments may be auto-completed through the completion API.

Request:

```
{
  "jsonrpc": "2.0",
  "id": 3,
  "method": "resources/templates/list"
}
```

Response:

```
{
  "jsonrpc": "2.0",
  "id": 3,
  "result": {
    "resourceTemplates": [
      {
        "uriTemplate": "file:///{path}",
        "name": "Project Files",
        "description": "Access files in the project directory",
        "mimeType": "application/octet-stream"
      }
    ]
  }
}
```

List Changed Notification

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

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

Subscriptions

The protocol supports optional subscriptions to resource changes. Clients can subscribe to specific resources and receive notifications when they change:

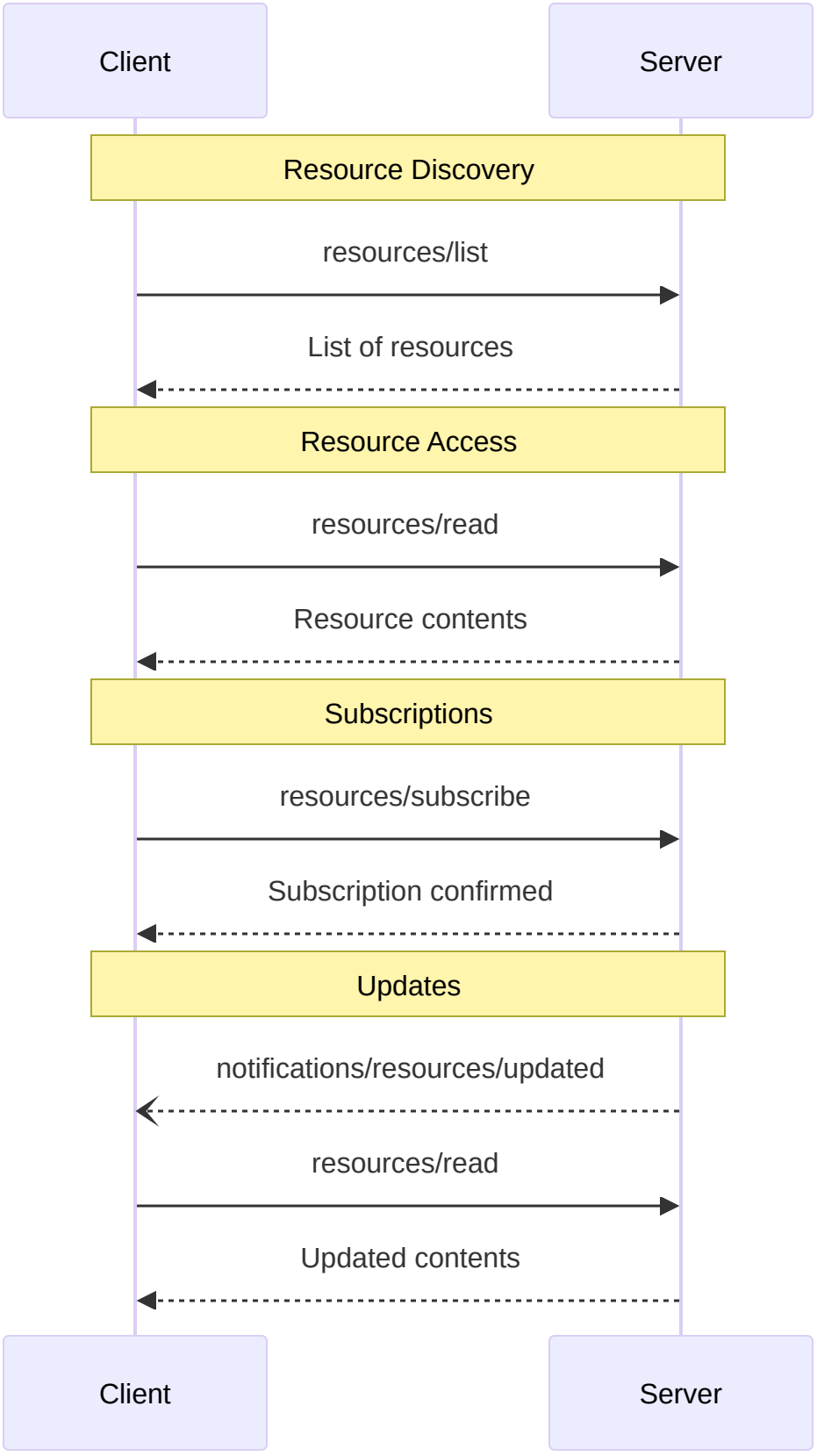
Subscribe Request:

```
{
  "jsonrpc": "2.0",
  "id": 4,
  "method": "resources/subscribe",
  "params": {
    "uri": "file:///project/src/main.rs"
  }
}
```

Update Notification:

```
{
  "jsonrpc": "2.0",
  "method": "notifications/resources/updated",
  "params": {
    "uri": "file:///project/src/main.rs"
  }
}
```

Message Flow


Client
Server
Resource Discovery
resources/list
List of resources
Resource Access
resources/read
Resource contents
Subscriptions
resources/subscribe
Subscription confirmed
Updates
notifications/resources/updated
resources/read
Updated contents
Client
Server

Data Types

Resource

A resource definition includes:

- `uri`: Unique identifier for the resource
- `name`: Human-readable name
- `description`: Optional description
- `mimeType`: Optional MIME type

Resource Contents

Resources can contain either text or binary data:

Text Content

```
{
  "uri": "file:///example.txt",
  "mimeType": "text/plain",
  "text": "Resource content"
}
```

Binary Content

```
{
  "uri": "file:///example.png",
  "mimeType": "image/png",
  "blob": "base64-encoded-data"
}
```

Common URI Schemes

The protocol defines several standard URI schemes. This list not exhaustive—implementations are always free to use additional, custom URI schemes.

https://

Used to represent a resource available on the web.

Servers **SHOULD** use this scheme only when the client is able to fetch and load the resource directly from the web on its own—that is, it doesn't need to read the resource via the MCP server.

For other use cases, servers **SHOULD** prefer to use another URI scheme, or define a custom one, even if the server will itself be downloading resource contents over the internet.

file://

Used to identify resources that behave like a filesystem. However, the resources do not need to map to an actual physical filesystem.

MCP servers **MAY** identify file:// resources with an XDG MIME type, like `inode/directory`, to represent non-regular files (such as directories) that don't otherwise have a standard MIME type.

git://

Git version control integration.

Error Handling

Servers **SHOULD** return standard JSON-RPC errors for common failure cases:

- Resource not found: `-32002`
- Internal errors: `-32603`

Example error:

```
{
  "jsonrpc": "2.0",
  "id": 5,
  "error": {
    "code": -32002,
    "message": "Resource not found",
    "data": {
      "uri": "file:///nonexistent.txt"
    }
  }
}
```

Security Considerations

1. Servers **MUST** validate all resource URIs
2. Access controls **SHOULD** be implemented for sensitive resources
3. Binary data **MUST** be properly encoded
4. Resource permissions **SHOULD** be checked before operations