<!-- page slug: mcp-docs-docs-tools-debugging | order: 8 -->

- Debugging tools overview
- Debugging in Claude Desktop
  - Checking server status
  - Viewing logs
  - Using Chrome DevTools
- Common issues
  - Working directory
  - Environment variables
  - Server initialization
  - Connection problems
- Implementing logging
  - Server-side logging
  - Client-side logging
- Debugging workflow
  - Development cycle
  - Testing changes
- Best practices
  - Logging strategy
  - Security considerations
- Getting help
- Next steps

title: Debugging description: A comprehensive guide to debugging Model Context Protocol (MCP) integrations —

Effective debugging is essential when developing MCP servers or integrating them with applications. This guide covers the debugging tools and approaches available in the MCP ecosystem.

This guide is for macOS. Guides for other platforms are coming soon.

Debugging tools overview

MCP provides several tools for debugging at different levels:

1. **MCP Inspector**
   - Interactive debugging interface
   - Direct server testing
   - See the Inspector guide for details
2. **Claude Desktop Developer Tools**
   - Integration testing
   - Log collection
   - Chrome DevTools integration
3. **Server Logging**
   - Custom logging implementations
   - Error tracking
   - Performance monitoring

Debugging in Claude Desktop

Checking server status

The Claude.app interface provides basic server status information:

1. Click the icon to view:
   - Connected servers
   - Available prompts and resources
2. Click the icon to view:
   - Tools made available to the model

Viewing logs

Review detailed MCP logs from Claude Desktop:

```
# Follow logs in real-time
tail -n 20 -F ~/Library/Logs/Claude/mcp*.log
```

The logs capture: - Server connection events - Configuration issues - Runtime errors - Message exchanges

Using Chrome DevTools

Access Chrome’s developer tools inside Claude Desktop to investigate client-side errors:

1. Create a `developer_settings.json` file with `allowDevTools` set to true:

```
echo '{"allowDevTools": true}' > ~/Library/Application\ Support/Claude/developer_settings.json
```

2. Open DevTools: `Command-Option-Shift-i`

Note: You’ll see two DevTools windows: - Main content window - App title bar window

Use the Console panel to inspect client-side errors.

Use the Network panel to inspect: - Message payloads - Connection timing

Common issues

Working directory

When using MCP servers with Claude Desktop:

- The working directory for servers launched via `claude_desktop_config.json` may be undefined (like `/` on macOS) since Claude Desktop could be started from anywhere
- Always use absolute paths in your configuration and `.env` files to ensure reliable operation
- For testing servers directly via command line, the working directory will be where you run the command

For example in `claude_desktop_config.json`, use:

```
{
  "command": "npx",
  "args": ["-y", "@modelcontextprotocol/server-filesystem", "/Users/username/data"]
}
```

Instead of relative paths like `./data`

Environment variables

MCP servers inherit only a subset of environment variables automatically, like `USER`, `HOME`, and `PATH`.

To override the default variables or provide your own, you can specify an `env` key in `claude_desktop_config.json`:

```
{
  "myserver": {
    "command": "mcp-server-myapp",
    "env": {
      "MYAPP_API_KEY": "some_key",
    }
  }
}
```

Server initialization

Common initialization problems:

1. **Path Issues**
   - Incorrect server executable path
   - Missing required files
   - Permission problems
   - Try using an absolute path for `command`
2. **Configuration Errors**
   - Invalid JSON syntax
   - Missing required fields
   - Type mismatches
3. **Environment Problems**
   - Missing environment variables
   - Incorrect variable values
   - Permission restrictions

Connection problems

When servers fail to connect:

1. Check Claude Desktop logs
2. Verify server process is running
3. Test standalone with Inspector
4. Verify protocol compatibility

Implementing logging

Server-side logging

When building a server that uses the local stdio transport, all messages logged to stderr (standard error) will be captured by the host application (e.g., Claude Desktop) automatically.

Local MCP servers should not log messages to stdout (standard out), as this will interfere with protocol operation.

For all transports, you can also provide logging to the client by sending a log message notification:

```
python     server.request_context.session.send_log_message(       level="info",       data="Server started successfully",     ) typescript     server.sendLoggingMessage({       level: "info",       data: "Server started successfully",     });
```

Important events to log: - Initialization steps - Resource access - Tool execution - Error conditions - Performance metrics

Client-side logging

In client applications:

1. Enable debug logging
2. Monitor network traffic
3. Track message exchanges
4. Record error states

Debugging workflow

Development cycle

1. Initial Development
   - Use Inspector for basic testing
   - Implement core functionality
   - Add logging points
2. Integration Testing
   - Test in Claude Desktop
   - Monitor logs
   - Check error handling

Testing changes

To test changes efficiently:

- **Configuration changes**: Restart Claude Desktop
- **Server code changes**: Use Command-R to reload
- **Quick iteration**: Use Inspector during development

Best practices

Logging strategy

1. **Structured Logging**
   - Use consistent formats
   - Include context
   - Add timestamps
   - Track request IDs
2. **Error Handling**
   - Log stack traces
   - Include error context
   - Track error patterns
   - Monitor recovery
3. **Performance Tracking**
   - Log operation timing

- Monitor resource usage
- Track message sizes
- Measure latency

Security considerations

When debugging:

1. **Sensitive Data**
   - Sanitize logs
   - Protect credentials
   - Mask personal information
2. **Access Control**
   - Verify permissions
   - Check authentication
   - Monitor access patterns

Getting help

When encountering issues:

1. **First Steps**
   - Check server logs
   - Test with Inspector
   - Review configuration
   - Verify environment
2. **Support Channels**
   - GitHub issues
   - GitHub discussions
3. **Providing Information**
   - Log excerpts
   - Configuration files
   - Steps to reproduce
   - Environment details

Next steps

<Card title="MCP Inspector" icon="magnifying-glass" href="/docs/tools/inspector"

> Learn to use the MCP Inspector

□□□ #2
□ □□□ □□□ 2026-10-31 13:08:55 UTC
□ □□□ □□□ 2026-10-31 13:08:56 UTC