Model Context Protocol

Advanced Topics for C# Developers

What is MCP?

- Open standard for AI-to-application integration
- Enables secure connections to external data sources and tools
- Standardized protocol for AI assistants to interact with services
- Maintains security boundaries while enabling rich functionality
- Supported by Claude Desktop, IDEs, and custom clients

01
Core Concepts

MCP Architecture

1. **Client**

   AI application that consumes MCP services (Claude Desktop, IDE plugins)

2. **Server**

   Your service that exposes capabilities via the MCP protocol

3. **Transport**

   Communication channel between client and server (STDIO, SSE, HTTP)

4. **Protocol**

   JSON-RPC 2.0 message format for structured communication

The Three Pillars

1. **Tools**

   Executable functions the AI can invoke — file operations, API calls, commands

2. **Prompts**

   Reusable templates for AI interactions — code review, documentation patterns

3. **Resources**

   Structured data access points — configuration, system info, reference data

02
Advanced Features

Sampling

Server requests LLM completions from the connected client, shifting AI costs to client infrastructure.

```
var result = await mcpContext.SendSamplingRequestAsync(
    $"Summarize this content:\n{fileContent}",
    cancellationToken
);
```

✓ Enables sophisticated analysis without server-side AI

✓ Client controls model selection and parameters

✓ Reduces server complexity and costs

Roots

Security boundaries that control file system access through approved directory paths.

```
var approvedRoots = new[] {
    Environment.GetFolderPath(SpecialFolder.MyDocuments),
    Path.GetTempPath()
};

bool isValid = IsPathWithinRoots(requestedPath);
```

✓ Prevents unauthorized access outside designated boundaries

✓ User-configurable permission model

✓ Essential for production security

Progress Notifications

Real-time updates for long-running operations, enabling responsive client UIs.

```
await mcpContext.SendProgressNotification(
    progressToken,
    progress: currentStep,
    total: totalSteps
);
```

✓ Enables client-side progress bars and status indicators

✓ Uses progressToken from client metadata

✓ Recommended for operations exceeding 1 second

Structured Logging

Multi-level diagnostic logging that sends information to connected clients.

```
await mcpContext.SendLogNotification(
    LoggingLevel.Info,
    logger: "FileSearch",
    data: $"Found {matches.Count} files"
);
```

✓ Supports Debug, Info, Warning, and Error levels

✓ Integrates with Microsoft.Extensions.Logging

✓ Enables client-side debugging views

03
Transport Mechanisms

Transport Comparison

Feature	STDIO	SSE	HTTP
Best For	Local / Desktop	Web Applications	REST APIs
Bidirectional	Yes	Server → Client	Request / Response
Sampling Support	Full	Full	Configurable
Scaling Model	Single Process	Stateful Sessions	Horizontal
Setup Complexity	Simple	Moderate	Higher

STDIO Transport

- JSON-RPC 2.0 messages over stdin/stdout
- Client spawns server as a child process
- Full bidirectional communication support
- Ideal for local development and CLI tools
- Simple setup with no network configuration required

04
Message Patterns

MCP Message Types

Request / Response

- Expects a return value
- tools/call — execute tool
- prompts/get — retrieve prompt
- resources/read — read data
- Synchronous pattern

Notifications

- Fire-and-forget messages
- notifications/progress
- notifications/message
- No response expected
- Asynchronous pattern

05
Implementation

Project Structure

- Program.cs — Entry point with dependency injection setup
- McpServerContext.cs — MCP protocol interface bridge
- Tools/ — MCP tool implementations
- Prompts/ — Reusable prompt templates
- Resources/ — Structured data access layer

Demo Tools

Tool	Description	Feature Demonstrated
analyze_large_file	File analysis with updates	Progress Notifications
read_file_secure	Secure file reading	Roots Security
ai_summarize_files	AI-powered summary	Sampling
search_files	Pattern-based search	Structured Logging
batch_process_files	Batch processing	Combined Features

Claude Desktop Configuration

Add your MCP server to the Claude Desktop configuration file.

```
{
  "mcpServers": {
    "csharp-demo": {
      "command": "dotnet",
      "args": ["run", "--project", "path/to/CSharpMcpDemo.csproj"]
    }
  }
}
```

✓ Server starts automatically on demand

✓ Supports multiple server configurations

✓ Can also install as a global .NET tool

06
Best Practices

Security Best Practices

- Always validate paths against approved roots
- Use CancellationToken for graceful operation control
- Implement comprehensive error handling
- Log security-relevant events for auditing
- Never expose sensitive data in tool responses

Performance Best Practices

- Use progress notifications for operations > 1 second
- Implement streaming for large data transfers
- Cache frequently accessed resources
- Use async/await patterns throughout
- Batch related operations when possible

Key Takeaways

- MCP standardizes AI-to-application integration
- Tools, Prompts, and Resources are the core primitives
- Sampling enables server-initiated AI calls
- Roots provide essential security boundaries
- STDIO transport is ideal for local and desktop scenarios
- C# with .NET provides excellent MCP support

Resources

- MCP Specification — modelcontextprotocol.io
- Anthropic MCP Course — anthropic.skilljar.com
- NuGet Package — ModelContextProtocol
- This Demo — GitHub repository

Thank You

Questions?