MakerAi 2.5
MCP Server
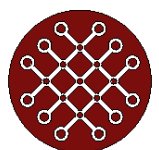
CimaMaker

email: gustavoeenriquez@gmail.com

website: https://makerai.cimamaker.com

Tel.:
+573128441700

Doc. Versión 1.0

MAKERAI
DELPHI SUITE

CONTENT

1. WHAT IS AN MCP (MODEL CONTEXT PROTOCOL)?

The MCP (Model Context Protocol) is an open standard designed so that applications can connect and communicate with artificial intelligence models and their associated tools in a uniform way.

In simple terms, MCP acts as a communication bridge between:

- A client (for example, Claude Desktop or MakerAI Delphi Suite in your Delphi applications).
- One or more MCP servers, which expose functionalities such as file access, databases, external tools, or even specialized agents.

1.1 Objective of MCP

The goal of the protocol is to standardize how messages are sent and received between a client and a server, using a JSON-RPC-based standard format.

This allows any MCP-compatible client to connect to any MCP server, regardless of the implementation language.

1.2 Main Features

- **Standardized communication**: uses JSON-RPC for message exchange.
- **Extensible**: each server can register its own tools.
- **Flexible**: supports different transports (stdio, sockets, web).
- **Interoperable**: a client can connect to MCP servers written in Delphi, Node.js, Python, Go, etc.
- **Multi-server**: the same client can manage multiple MCP connections at the same time.

1.3 Examples of MCP Use

- **Filesystem Server**: an MCP server that allows browsing and manipulating files from the client.
- **Database Server**: an MCP server that exposes queries to a database.
- **Specialized agents**: servers that encapsulate business logic and offer it as accessible tools.

1.4 MCP within MakerAI Delphi Suite

MakerAI includes MCP support so Delphi applications can easily connect to external servers, leveraging:

- Tool discovery (tools/list)
- Function execution (tools/call)
- Direct integration with Delphi components

This makes Delphi a first-class MCP client, capable of integrating with modern AI ecosystems and external services.

1.5 Client-Server Flow in MCP

The basic communication flow between a client (e.g., Claude Desktop or MakerAI Delphi Suite) and an MCP server is as follows:
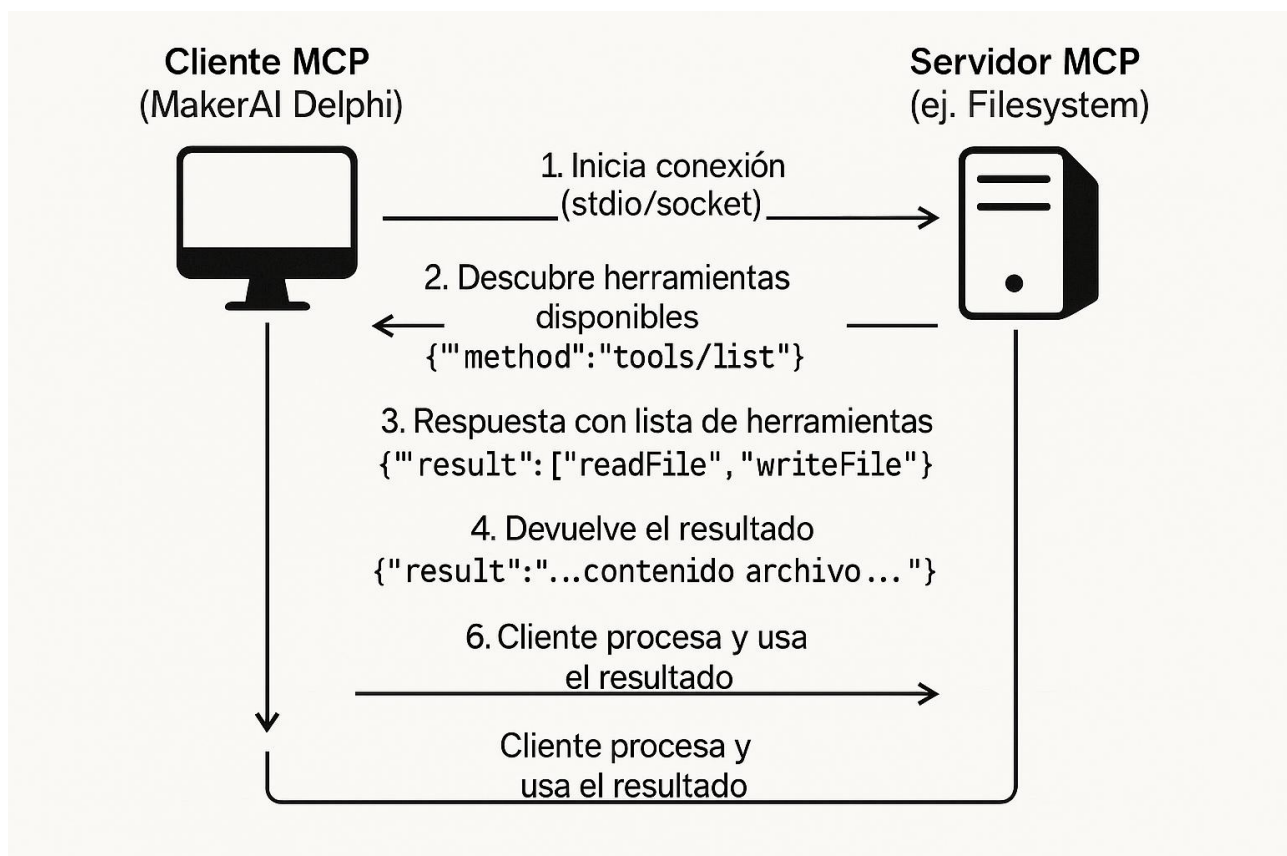
2. SETTING UP AN MCP SERVER IN CLAUDE (CLAUDE DESKTOP)

This section explains how to connect a local MCP server to Claude Desktop, allowing Claude to access functionalities such as file management on your system.

2.1 Prerequisites

- Have Claude Desktop installed (available for Windows; Linux coming soon). Make sure to use the latest version via the application menu ("Check for Updates…").
- Have Node.js installed (LTS version recommended). Verify installation by running in terminal:

```
node --version
```

2.2 What the Filesystem Server Does

The Filesystem Server (MCP server for file systems) allows Claude to perform actions such as:

- Read and list folder contents
- Create, move, or rename files
- Search files by name or content

All these actions are only performed with your explicit consent.

2.3 Configuration Steps

1. **Open developer settings in Claude Desktop**
   - Launch Claude Desktop.
   - Go to *Settings → Developer* tab and click *Edit Config*. This opens (or creates) the claude_desktop_config.json file.
2. **Insert the Filesystem Server configuration**
   In the claude_desktop_config.json file, add something like the following (adjusting paths to your system and user):

```
{
  "mcpServers": {
    "filesystem": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-filesystem",
        "/Users/yourUser/Desktop",
        "/Users/yourUser/Downloads"
      ]
    }
  }
}
```

   - "filesystem" is the friendly name of the server.
   - "command": "npx" tells Claude to use Node.js to run the package.

o "args" contains the server package and allowed directories.

Example for another server (e.g., built with liblab):

```
{
  "mcpServers": {
    "my-local-api": {
      "command": "node",
      "args": [
        "/path/to/your/mcp-server/dist/index.js"
      ],
      "env": {
        "YOUR_API_KEY": "APIvalue"
      }
    }
  }
}
```

3. **Save and restart Claude Desktop**
   - o Save the changes in claude_desktop_config.json.
   - o Fully close Claude Desktop and reopen it. This allows the app to load the configuration and automatically run the MCP server.

4. **Verify connection**
   - o Once reloaded, look for a tools icon (or hammer) in the bottom-right corner of the input box – this indicates Claude detected the MCP server.
   - o From the conversation, you can try commands such as:
     - ▪ "What files are in my Desktop?"
     - ▪ "Save a poem in my Downloads folder."
       Claude will display a list of tools and ask for approval before performing any action.

2.4 Remote or Externally Generated Connections

If you want to connect to a remote MCP server (via HTTP/SSE) or import it from another source:

- **Remote**: use the *Connectors* or *Custom Connector* section in Claude (web or desktop), enter the URL, and follow authentication (OAuth, API key, etc.).
- **Import from Claude Desktop**: you can import existing MCP configs from Claude Desktop to Claude Code with:
  claude mcp add-from-claude-desktop
  claude mcp list
- **Add servers directly from JSON**:
  claude mcp add-json name '{"type":"stdio","command":"...","args":[...]}'

Quick Checklist

Step Action

1   Verify Claude Desktop and Node.js installation

Step Action

2   Open Settings → Developer → Edit Config

3   Insert JSON block with server configuration (filesystem or custom)

4   Save file and restart app

5   Confirm tool icon ⌬ appears and test commands

3. CONNECTING MAKERAI DELPHI SUITE TO MCP SERVERS

1. **Add the TAiFunctions component**

   Place the TAiFunctions component in the main form or wherever MCP integration is required.

   This component centralizes connection and management functions for both MCP clients and standard function calling of LLMs that support it.

2. **Configure MCPClients**

   Inside the MCPClients property, there is a collection editor to manage connections to different MCP servers.

   - Each connection is an independent MCP client.
   - You can add one or more MCP servers depending on your application needs.

3. **Add a new MCP client**

   Press the *Add* button in the collection editor to create a new entry. Each entry has key properties:

   - **TransportType**: protocol of transport:
     - tpStdio → standard input/output communication
     - tpHttp → HTTP communication
     - tpSSE → not yet supported by current MCP servers
     - tpMakerAI → internal transport for functions within the suite
   - **Enabled**: enable or disable a client.
     - True → client is active and loaded at startup
     - False → client is inactive but config is preserved
   - **EnvVars**: environment variables for this MCP server. Example: APIKEY=xxxxxxx
   - **Name**: arbitrary name identifying the MCP client. Recommended to reflect server purpose (e.g., ClaudeFS, PostgresMCP, GitServer).
   - **Params**: contains specific configuration parameters based on TransportType (commands, args, URLs, credentials).

Example Configurations:

- **Filesystem Server in Node.js with npx**

```
Command=npx
Arguments=@
RootDir=C:\Users\genri\AppData\Roaming
PATH=C:\
ApiHeaderName=Authorization
ApiBearerToken=@MCPBearerToken
URL=http://localhost:3001/sse
Login=login
Password=password
OAuthClientId=ClientId
OAuthURL=http://localhost:6274/oauth/callback
OAuthScope=Scope
Timeout=15000
```

- **Filesystem npx en nodejs**

```
Configuración para exponer un sistema de archivos local vía MCP:
Command=npx
Arguments=@modelcontextprotocol/server-filesystem C:\mcp\fs-root C:\mcp\pruebas
Timeout=15000
```

- **Postgres MCP Server**

```
Params:
Command=C:\Users\genri\AppData\Roaming\Python\Python313\Scripts\postgres-mcp.exe
Arguments=--access-mode=unrestricted
Timeout=15000

EnvVar:
DATABASE_URI=postgresql://postgres:masterkey@localhost/lunaidb
```

Configuration and Testing Tools from Delphi with MakerAi MCP Client

Within the configuration of MakerAi Delphi Suite, in the TAiFunctions component under the **MCPClients** property, there is a **Configuration** option. When you click on it, the **MCP Client Editor** window opens, allowing you to configure and test connections both in HTTP and in StdIO.
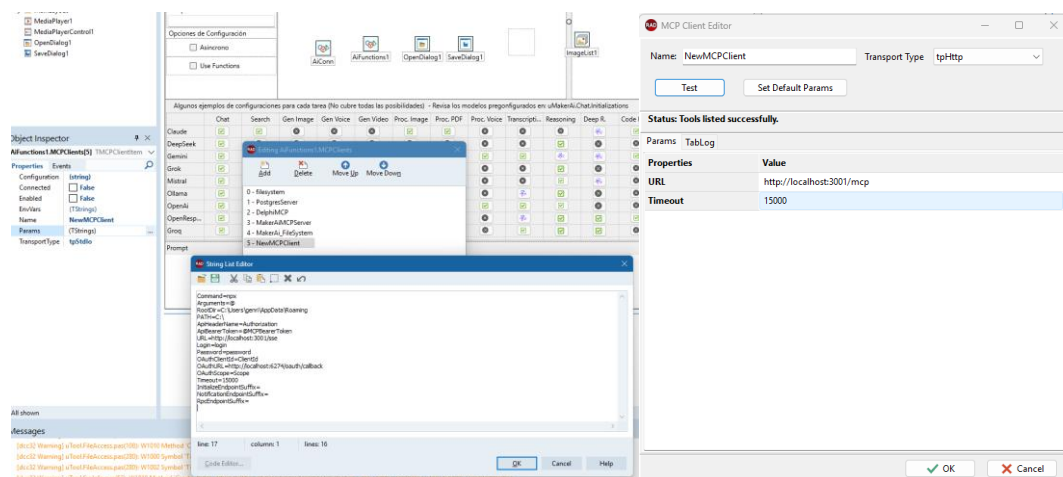
IMPLEMENTING AN MCP SERVER IN DELPHI

This chapter is the core of the developer manual. Here we learn how the demo server is structured, how each part works, and most importantly, how to extend it by creating custom tools.

We will use the AiMCPServerDemo project code as the base.

4.1 Server Project Structure

The demo project is modular and easy to extend. It includes:

- **AiMCPServerDemo.dpr**: main program (console app) responsible for:
  - Parsing command-line arguments (protocol, port, etc.)
  - Creating the server instance (HTTP or Stdio)
  - Registering available tools
  - Controlled server start/stop
- **uTool.FileAccess.pas**: encapsulates tools for file access (list_files, read_file, write_file).
- **uTool.SysInfo.pas**: provides system_info tool to retrieve OS, memory, disk data, etc.

Each tool set is separated in its own unit, a recommended practice for clean and maintainable code.

4.2 Main Program Analysis (AiMCPServerDemo.dpr)

- **Default Config & Argument Parsing**: sets default values (http, port 3000, etc.), then overrides via CLI args or .ini config.
- **Server Instance Creation**:
  - TAiMCPHttpServer: web server listening on HTTP.
  - TAiMCPStdioServer: stdio server (ideal for subprocess integration with apps like Claude VS Code extension).
- **Central Tool Registration**: handled by RegisterAllToolsAndResources.
- **Server Start & Main Loop**:
  - MCPServer.Start → starts listening.
  - Loop keeps console alive while handling requests in background threads.
  - Clean shutdown with Ctrl+C, stopping server gracefully.

4.3 Anatomy of a Tool

Every tool has three core elements (example: TListFilesTool):

1. **Parameter Class** (TListFilesParams) → defines input properties with attributes like [AiMCPSchemaDescription] and [AiMCPOptional].
2. **Tool Class** (TListFilesTool) → inherits TAiMCPToolBase<T>, defines FName, FDescription, and implements ExecuteWithParams.
3. **Registration Procedure** → registers tool with the server using RegisterTool.

4.4 Practical Case: Creating a New Tool get_datetime

- **Step 1**: Create a new unit uTool.DateTime.pas.
- **Step 2**: Implement TGetDateTimeTool with parameters (e.g., return JSON or plain text).
- **Step 3**: Register in AiMCPServerDemo.dpr via RegisterAllToolsAndResources.

```pascal
unit uTool.DateTime;

interface

uses
  System.SysUtils,
  System.Classes,
  System.JSON,
  uMakerAi.MCPServer.Core;

type
  // 1. CLASE DE PARÃ•METROS
  TGetDateTimeParams = class
  private
    FFormatAsJson: Boolean;
  public
    [AiMCPOptional]
    [AiMCPSchemaDescription('Devolver la fecha/hora como un objeto JSON estructurado
(default: true). Si es false, devuelve texto simple.')]
    property FormatAsJson: Boolean read FFormatAsJson write FFormatAsJson;
  end;

  // 2. CLASE DE LA HERRAMIENTA
  TGetDateTimeTool = class(TAiMCPToolBase<TGetDateTimeParams>)
  protected
    function ExecuteWithParams(const AParams: TGetDateTimeParams; const AuthContext:
TAiAuthContext): TJSONObject; override;
  public
    constructor Create;
  end;

// 3. PROCEDIMIENTO DE REGISTRO
procedure RegisterTools(ALogicServer: TAiMCPServer);

implementation

{ TGetDateTimeTool }

procedure RegisterTools(ALogicServer: TAiMCPServer);
begin
  if not Assigned(ALogicServer) then
    Exit;

  // Registramos la herramienta 'get_datetime' para que el servidor sepa de su existencia.
  ALogicServer.RegisterTool('get_datetime',
    function: IAiMCPTool
    begin
      Result := TGetDateTimeTool.Create;
    end);
end;

constructor TGetDateTimeTool.Create;
begin
  inherited Create;
  // El nombre que la IA usarÃ¡ para llamar a la herramienta.
  FName := 'get_datetime';
```

```
  // La descripciÃ³n que la IA leerÃ¡ para entender quÃ© hace la herramienta.
  FDescription := 'Devuelve la fecha y hora actual del servidor. Puede devolverla como texto
o como un objeto JSON.';

  // Establecemos el valor por defecto para los parÃ¡metros.
  TGetDateTimeParams.Create.FFormatAsJson := True;
end;

function TGetDateTimeTool.ExecuteWithParams(const AParams: TGetDateTimeParams; const
AuthContext: TAiAuthContext): TJSONObject;
var
  NowDateTime: TDateTime;
  ResultJson: TJSONObject;
  ResultText: string;
begin
  try
    NowDateTime := Now;

    if AParams.FormatAsJson then
    begin
      // Creamos un objeto JSON con informaciÃ³n detallada
      ResultJson := TJSONObject.Create;
      ResultJson.AddPair('date', FormatDateTime('yyyy-mm-dd', NowDateTime));
      ResultJson.AddPair('time', FormatDateTime('hh:nn:ss', NowDateTime));
      ResultJson.AddPair('timezone', 'Local Server Time'); // Simplificado
      ResultJson.AddPair('iso8601', FormatDateTime('c', NowDateTime));

      // Usamos el builder para enviar el JSON como una cadena de texto.
      // La IA es capaz de interpretar esta cadena como un objeto JSON.
      Result := TAiMCPResponseBuilder.New.AddText(ResultJson.ToJSON).Build;
      ResultJson.Free;
    end
    else
    begin
      // Creamos una cadena de texto simple
      ResultText := Format('La fecha y hora actual del servidor es: %s',
[FormatDateTime('dd/mm/yyyy hh:nn:ss', NowDateTime)]);
      Result := TAiMCPResponseBuilder.New.AddText(ResultText).Build;
    end;

  except
    on E: Exception do
      Result := TAiMCPResponseBuilder.New.AddText('â•Œ Error obteniendo la fecha y hora: ' +
E.Message).Build;
  end;
end;

end.
```

4.5 Step 3: Integrating the New Tool

- Add uTool.DateTime to project uses.

```
uses
  System.SysUtils,
  System.Classes,
  uTool.FileAccess in 'uTool.FileAccess.pas',
  uTool.SysInfo in 'uTool.SysInfo.pas',
  uTool.DateTime in 'uTool.DateTime.pas', // <-- AÃ'ADIR ESTA LÃ•NEA
  uMakerAi.MCPServer.Core in '..\..\Source\MCPServer\uMakerAi.MCPServer.Core.pas',
  UMakerAi.MCPServer.Http in '..\..\Source\MCPServer\UMakerAi.MCPServer.Http.pas',
  UMakerAi.MCPServer.Stdio in '..\..\Source\MCPServer\UMakerAi.MCPServer.Stdio.pas';
```

- Call uTool.DateTime.RegisterTools in RegisterAllToolsAndResources.

```
procedure RegisterAllToolsAndResources(ALogicServer: TAiMCPServer);
begin
  if not Assigned(ALogicServer) then
    Exit;

  WriteLn('Registering tools and resources...');
```

```
  uTool.FileAccess.RegisterTools(ALogicServer);
  uTool.SysInfo.RegisterTools(ALogicServer);
  uTool.DateTime.RegisterTools(ALogicServer); // <-- AÃ'ADIR ESTA LÃ•NEA

  WriteLn('Registration complete.');
end;
```

- Recompile project.

4.6 Compiling and Running

After compiling, when server starts you'll see:

Registering tools and resources...

Registration complete.

Any MCP client (Claude Playground or MakerAI Delphi Suite) connected will now see and use get_datetime.