

# MCP SERVER

(MODEL CONTEXT PROTOCOL)

TOOLS/LIST

[ tools/call ]

tools/call

```
{  
  params...  
}...
```

resources/

prompts/

## MakerAi 2.5 MCP Server

**CimaMaker**

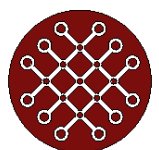
Correo electrónico: [gustavoeenriquez@gmail.com](mailto:gustavoeenriquez@gmail.com)

Sitio web: <https://makerai.cimamaker.com>

Tel.:

+573128441700

Doc. Versión 1.0



**MAKERAI**  
DELPHI SUITE

# CONTENIDO

1. ¿Qué es un MCP (Model Context Protocol)? .....	3
1.1. Objetivo del MCP .....	3
1.2. Características principales .....	3
1.3. Ejemplos de uso de MCP .....	3
1.4. MCP dentro de MakerAI Delphi Suite.....	3
1.5. Flujo Cliente-Servidor en MCP .....	4
2. Configuración de un servidor MCP en Claude (Claude Desktop) .....	5
2.1. Requisitos previos.....	5
2.2. Qué hace el <i>Filesystem Server</i> .....	5
2.3. Pasos para configurar .....	5
2.4. Conexiones remotas o generadas por herramientas externas .....	6
3. Conexión de MakerAI Delphi Suite a servidores MCP .....	8
1. Adicionar el componente TAIFunctions .....	8
3. Adicionar un nuevo cliente MCP .....	8
4. Implementación de un Servidor MCP en Delphi .....	11
4.1. Estructura del Proyecto Servidor.....	11
4.2. Análisis del Programa Principal (AiMCPServerDemo.dpr) .....	11
4.3. Anatomía de una Herramienta (Tool) .....	12
4.4. Caso Práctico: Creando una Nueva Herramienta get_datetime.....	13
4.5. Paso 3: Integrar la Nueva Herramienta en el Servidor .....	15

# 1. ¿QUÉ ES UN MCP (MODEL CONTEXT PROTOCOL)?

El MCP (Model Context Protocol) es un estándar abierto diseñado para que las aplicaciones puedan **conectarse y comunicarse con modelos de inteligencia artificial y sus herramientas asociadas** de una forma uniforme.

En términos simples, MCP actúa como un **punto de comunicación** entre:

- Un **cliente** (por ejemplo, Cloud Desktop o MakerAI Delphi Suite en tus aplicaciones Delphi).
- Uno o varios **servidores MCP**, que exponen funcionalidades como acceso a archivos, bases de datos, herramientas externas o incluso agentes especializados.

## 1.1. Objetivo del MCP

El objetivo del protocolo es **normalizar cómo se envían y reciben mensajes** entre un cliente y un servidor, usando un formato estándar basado en **JSON-RPC**.

Esto permite que cualquier cliente compatible con MCP pueda conectarse a cualquier servidor MCP, sin importar en qué lenguaje esté implementado.

## 1.2. Características principales

- **Comunicación estandarizada:** usa JSON-RPC para intercambio de mensajes.
- **Extensible:** permite que cada servidor registre sus propias herramientas (tools).
- **Flexible:** soporta diferentes transportes (stdio, sockets, web).
- **Interoperable:** un cliente puede conectarse a servidores MCP escritos en Delphi, Node.js, Python, Go, etc.
- **Multi-servidor:** un mismo cliente puede gestionar varias conexiones MCP al mismo tiempo.

## 1.3. Ejemplos de uso de MCP

- **Filesystem Server:** un servidor MCP que permite explorar y manipular archivos desde el cliente.
- **Database Server:** un servidor MCP que expone consultas a una base de datos.
- **Agentes especializados:** servidores que encapsulan lógica de negocio y la ofrecen como herramientas accesibles.

## 1.4. MCP dentro de MakerAI Delphi Suite

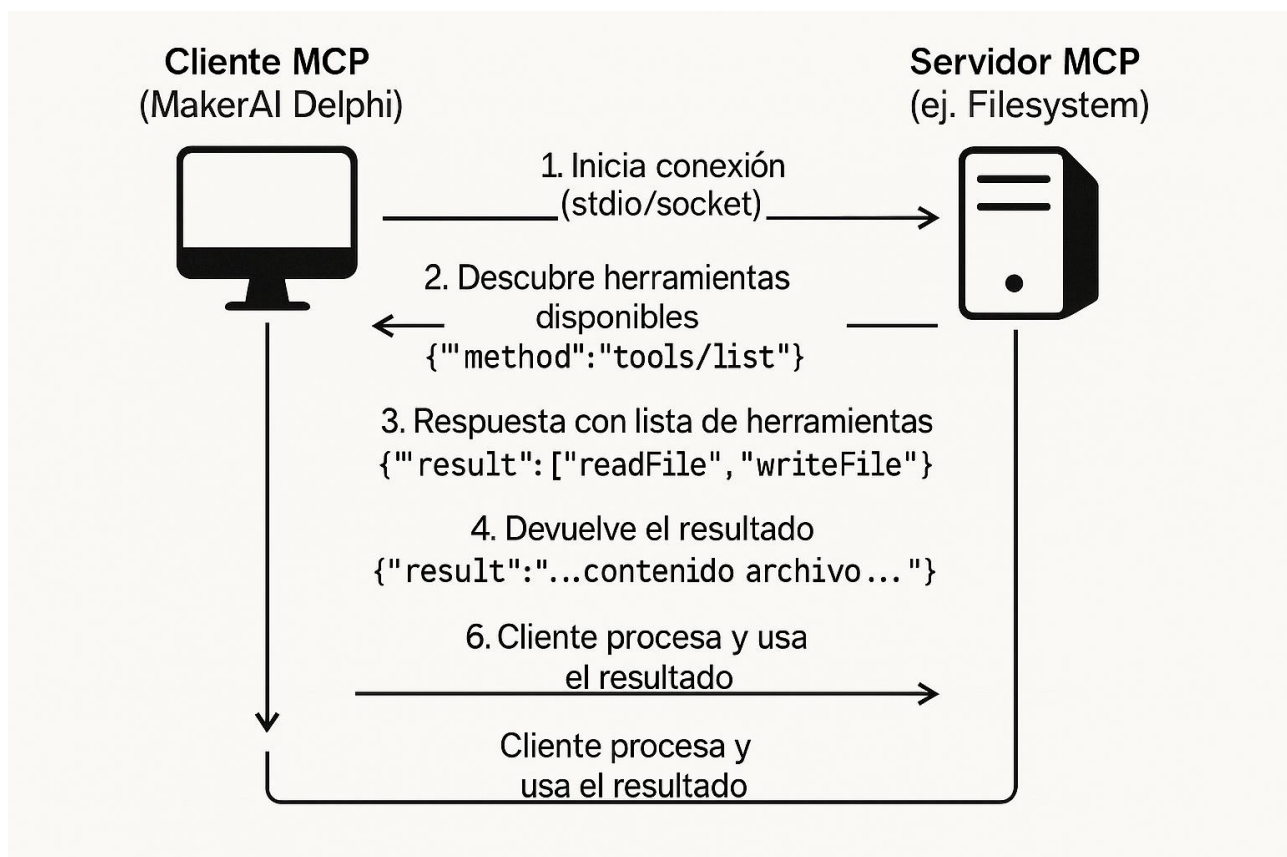
MakerAI incluye soporte MCP para que **las aplicaciones Delphi puedan conectarse fácilmente a servidores externos**, aprovechando:

- Descubrimiento de herramientas (tools/list).
- Ejecución de funciones (tools/call).
- Integración directa con componentes Delphi.

Esto convierte a Delphi en un **cliente MCP de primera clase**, capaz de integrarse con ecosistemas modernos de IA y servicios externos.

## 1.5. Flujo Cliente–Servidor en MCP

El flujo básico de comunicación entre un **cliente** (ej. Claude Desktop o MakerAI Delphi Suite) y un **servidor MCP** es el siguiente:



## 2. CONFIGURACIÓN DE UN SERVIDOR MCP EN CLAUDE (CLAUDE DESKTOP)

Esta sección te explica cómo conectar un servidor MCP local a Claude Desktop, permitiendo que Claude acceda a funcionalidades como manejo de archivos en tu sistema.

### 2.1. Requisitos previos

- Tener **Claude Desktop** instalado (disponible para Windows; próximamente en Linux). Asegúrate de usar la versión más reciente desde el menú de la aplicación ("Check for Updates...").
- Tener instalado **Node.js** (se recomienda usar la versión LTS). Verifica la instalación ejecutando en terminal:

```
node --version
```

### 2.2. Qué hace el *Filesystem Server*

El Filesystem Server (servidor MCP para sistema de archivos) permite a Claude realizar acciones como:

- Leer y listar contenidos de carpetas.
- Crear, mover o renombrar archivos.
- Buscar archivos por nombre o contenido.

Todas estas acciones solo se realizan con tu **expreso consentimiento**.

### 2.3. Pasos para configurar

#### 1. Abrir configuración de desarrollador en Claude Desktop

- Lanza Claude Desktop.
- Ve a **Settings** → pestaña **Developer**, y haz clic en **Edit Config**. Esto abrirá (o creará) el archivo `claude_desktop_config.json`.

#### 2. Insertar la configuración del servidor Filesystem

En el archivo `claude_desktop_config.json`, agrega algo como lo siguiente (ajustando rutas según tu sistema y usuario):

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

- "filesystem" es el nombre amigable del servidor.
- "command": "npx" le indica a Claude que use Node.js para ejecutar el paquete.
- "args" contiene el paquete del servidor y los directorios permitidos.

Si estás usando otro servidor (ej. uno creado con liblab), el bloque puede verse así:

```
{
  "mcpServers": {
    "my-local-api": {
      "command": "node",
      "args": [
        "/ruta/a/tu/servidor-mcp/dist/index.js"
      ],
      "env": {
        "YOUR_API_KEY": "valorAPI"
      }
    }
  }
}
```

### 3. Guardar y reiniciar Claude Desktop

- Guarda los cambios en `claude_desktop_config.json`.
- Cierra completamente Claude Desktop y vuelve a abrirlo. Esto permite que cargue la configuración y ejecute automáticamente el servidor MCP.

### 4. Verificación de conexión

- Una vez recargada la aplicación, busca un icono de herramientas (o martillo) en la esquina inferior derecha del campo de entrada de texto – es un indicador de que Claude detectó el servidor MCP.
- Desde la conversación, puedes probar comandos como:
  - “¿Qué archivos hay en mi Desktop?”
  - “Guarda un poema en mi carpeta Descargas.”

Claude te mostrará una lista de herramientas y te pedirá aprobación antes de realizar alguna acción.

## 2.4. Conexiones remotas o generadas por herramientas externas

Si deseas conectar un servidor MCP remoto (a través de HTTP/SSE) o importarlo desde otra fuente:

- **Remoto:** usa la sección de *Connectors* o *Custom Connector* en la interfaz de Claude (web o escritorio), ingresa la URL y sigue el flujo de autenticación (OAuth, API key, etc.).
- **Importación desde Claude Desktop:** puedes importar configuraciones MCP ya existentes desde Claude Desktop a Claude Code con:
  - `claude mcp add-from-claude-desktop`


y luego listar los servidores importados con:

`claude mcp list`

- También puedes añadir servidores directamente desde JSON usando el comando:
  - `claude mcp add-json nombre '{"type":"stdio","command":"...","args":[...]}'`
- 

## Resumen: Checklist Rápido

### Paso Acción

- 1 Verificar instalación de **Claude Desktop** y **Node.js**
- 2 Abrir Settings → Developer → Edit Config
- 3 Insertar bloque JSON con configuración del servidor (filesystem o personalizada)
- 4 Guardar el archivo y reiniciar la app
- 5 Confirmar la aparición del ícono de herramientas  y probar comandos

## 3. CONEXIÓN DE MAKERAI DELPHI SUITE A SERVIDORES MCP

### 1. Adicionar el componente TAIFunctions

En el formulario principal o en el formulario donde se requiera la integración MCP, se coloca el componente **TAIFunctions**.

Este componente centraliza las funciones de conexión y manejo tanto de clientes MCP como de function calling estándar de los modelos LLM que lo soporten.

### 2. Configuración de MCPClients

Dentro de la propiedad **MCPClients** se encuentra un **editor de colecciones**, desde el cual es posible administrar las distintas conexiones a servidores MCP.

- Cada conexión es un **cliente MCP** independiente.
- Se puede añadir uno o varios servidores MCP, dependiendo de las necesidades de la aplicación.

### 3. Adicionar un nuevo cliente MCP

Presionar el botón **Add** en el editor de colecciones para crear una nueva entrada.

Cada entrada tendrá las siguientes propiedades clave:

#### 3.1. TransportType

Define el protocolo de transporte de la información:

- tpStdio → comunicación estándar por entrada/salida.
- tpHttp → comunicación HTTP.
- tpSSE → *no soportado aún por los servidores MCP actuales*.
- tpMakerAI → transporte interno para funciones en la suite.

#### 3.2. Enabled

Permite habilitar o deshabilitar un cliente.

- True → el cliente queda activo y será cargado al iniciar la aplicación.
- False → el cliente no se utiliza, pero se conserva su configuración.

#### 3.3. EnvVars

Variables de entorno exclusivas para este servidor MCP.

Ejemplo:

```
APIKEY=xxxxxxx
```



### 3.4. Name

Nombre arbitrario que identifica al cliente MCP.

Se recomienda que el nombre refleje el propósito del servidor, por ejemplo:

- ClaudeFS
- PostgresMCP
- GitServer

### 3.5. Params

Contiene los parámetros de configuración específicos según el TransportType.

Aquí se definen comandos, argumentos, URL de servicios y credenciales de acceso.

#### 3.5.1. Ejemplos de configuración

Este es un listado de todas las configuraciones disponibles

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

#### 3.5.2 Servidor Filesystem npx en nodejs

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

#### 3.5.3. Servidor Postgres MCP

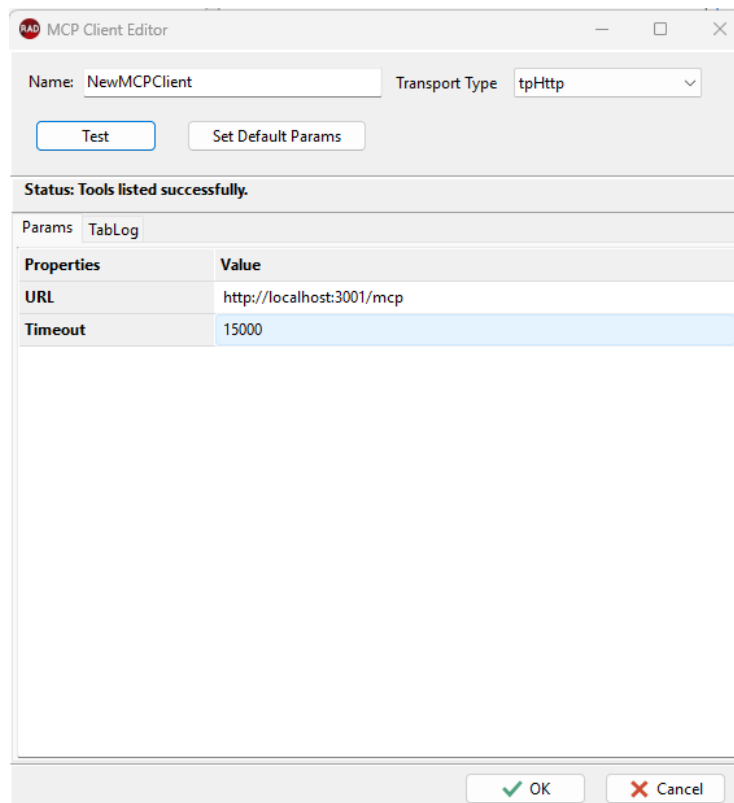
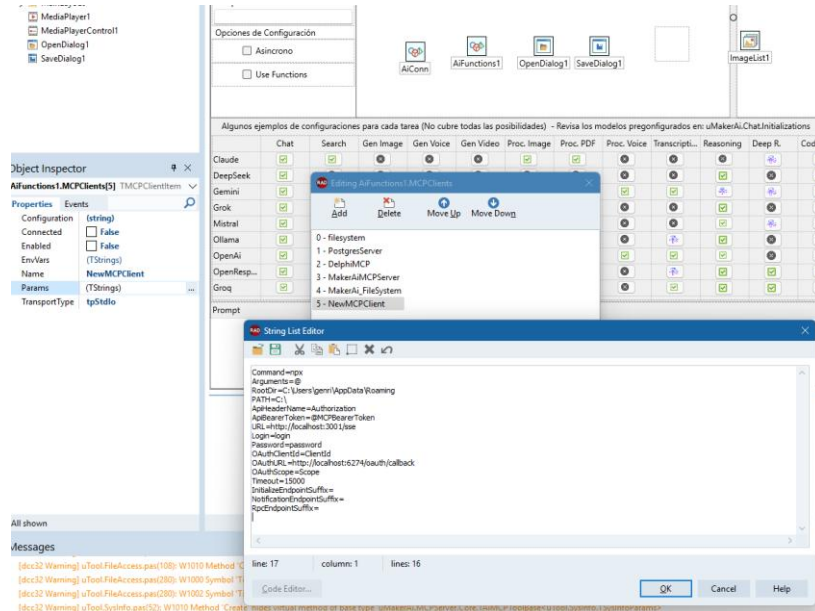
Ejemplo de conexión hacia PostgreSQL como servidor MCP:

```
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
```

### 3.5.4. Herramientas de configuración y pruebas desde Delphi con MakerAi MCP Client

Dentro de la configuración de MakerAi Delphi Suite en el componente TAIFunctions en la propiedad MCPClients se encuentra la propiedad "Configuration", al presionar en ella se abre la ventana de MCP Client Editor, el cual permite configurar y probar las conexiones tanto en HTTP como en StdIO



## 4. IMPLEMENTACIÓN DE UN SERVIDOR MCP EN DELPHI

Este capítulo es el corazón del manual para desarrolladores. Aquí aprenderemos cómo está estructurado el servidor de demostración, cómo funciona cada una de sus partes y, lo más importante, cómo extenderlo creando nuestras propias herramientas personalizadas. Usaremos el código del proyecto AiMCPServerDemo como base para nuestro aprendizaje.

### 4.1. Estructura del Proyecto Servidor

El proyecto de demostración está diseñado para ser modular y fácil de extender. Consta de los siguientes ficheros clave:

- **AiMCPServerDemo.dpr**: El programa principal. Es una aplicación de consola responsable de:
  - Interpretar los parámetros de la línea de comandos (protocolo, puerto, etc.).
  - Crear la instancia del servidor (HTTP o Stdio).
  - Registrar todas las herramientas disponibles.
  - Iniciar y detener el servidor de forma controlada.
- **uTool.FileAccess.pas**: Una unidad que encapsula todas las herramientas relacionadas con el acceso a ficheros (list\_files, read\_file, write\_file). Contiene la lógica para interactuar con el sistema de archivos de forma segura.
- **uTool.SysInfo.pas**: Una unidad que provee la herramienta system\_info, capaz de obtener datos sobre el sistema operativo, memoria, discos, etc.

Este enfoque de separar cada conjunto de herramientas en su propia unidad (unit) es una buena práctica que recomendamos seguir para mantener el código organizado y mantenible.

### 4.2. Análisis del Programa Principal (AiMCPServerDemo.dpr)

El fichero .dpr orquesta todo el funcionamiento del servidor. Vamos a desglosar sus secciones más importantes:

1. **Configuración por Defecto y Parseo de Argumentos**: Al inicio, el programa establece valores por defecto (protocolo http, puerto 3000, etc.). Luego, recorre los parámetros de la línea de comandos para sobrescribir estos valores, permitiendo una configuración dinámica al ejecutar el servidor. También incluye la capacidad de cargar la configuración desde un fichero .ini si no se pasan argumentos.
2. **Creación de la Instancia del Servidor**: Basado en el parámetro --protocol, el código decide qué tipo de servidor instanciar:

- `TAiMCPHttpServer`: Crea un servidor web que escucha peticiones HTTP. Es el más común para interactuar con aplicaciones web o clientes remotos.
- `TAiMCPStdioServer`: Crea un servidor que se comunica a través de la entrada/salida estándar (consola). Es ideal para ser invocado como un subproceso por otras aplicaciones, como la extensión de VS Code para Claude.

### 3. Registro Centralizado de Herramientas: El

procedimiento `RegisterAllToolsAndResources` es el punto de encuentro de todas las herramientas. Llama a los procedimientos de registro de cada unidad de herramientas (ej. `uTool.FileAccess.RegisterTools(ALogicServer)`). Esta centralización facilita la adición o eliminación de funcionalidades completas con solo añadir o quitar una línea de código.

### 4. Inicio del Servidor y Bucle Principal:

- `MCPServer.Start`: Pone en marcha el servidor para que empiece a escuchar peticiones.
- `while True do Sleep(1000)`: Mantiene la aplicación de consola viva indefinidamente. El trabajo real de atender las peticiones ocurre en hilos segundo plano gestionados por el componente servidor.
- `except on E: EControlC do`: Este bloque captura la pulsación de Ctrl+C para permitir un apagado limpio y ordenado del servidor, llamando a `MCPServer.Stop` en la sección `finally`.

## 4.3. Anatomía de una Herramienta (Tool)

Toda herramienta en el ecosistema MCP de MakerAI se compone de tres elementos fundamentales. Analizaremos `TListFilesTool` de la unidad `uTool.FileAccess` como ejemplo:

### 1. La Clase de Parámetros (`TListFilesParams`):

- Es una clase simple que define las propiedades que la herramienta aceptará como entrada.
- Cada property se corresponde con un parámetro que la IA puede proporcionar.
- Se utilizan atributos para describir cada parámetro a la IA:
  - `[AiMCPSchemaDescription('...')]`: Proporciona una descripción clara de lo que hace el parámetro. Esto es **esencial** para que el modelo de lenguaje sepa cómo usarlo.
  - `[AiMCPOptional]`: Indica que el parámetro no es obligatorio.

### 2. La Clase de la Herramienta (`TListFilesTool`):

- Debe heredar de `TAiMCPToolBase<T>`, donde `T` es la clase de parámetros que hemos definido (ej. `TAiMCPToolBase<TListFilesParams>`).
- **Constructor:** En el constructor, es vital establecer `FName` (el nombre con el que la IA llamará a la herramienta, ej: 'list\_files') y `FDescription` (una descripción clara y concisa de lo que hace la herramienta).
- **Método `ExecuteWithParams`:** Este es el núcleo de la herramienta. Recibe un objeto con los parámetros ya deserializados (`AParams`) y el contexto de autenticación (`AuthContext`). Aquí se implementa toda la lógica.
- **Retorno:** El método debe devolver un `TJSONObject`. Para ello, se utiliza el `TAiMCPResponseBuilder`, que facilita la creación de la estructura de respuesta JSON que el protocolo MCP espera. Por ejemplo: `TAiMCPResponseBuilder.New.AddText('Resultado...').Build;`

### 3. El Procedimiento de Registro (`RegisterTools`):

- Es una función simple, generalmente al final de la unidad, que se encarga de registrar las herramientas en la instancia del servidor.
- Utiliza `ALogicServer.RegisterTool('nombre_herramienta', function: IAI MCPTool begin Result := TMiClaseTool.Create; end);`. Esto le dice al servidor que cuando la IA quiera usar 'nombre\_herramienta', debe crear una nueva instancia de `TMiClaseTool`.

## 4.4. Caso Práctico: Creando una Nueva Herramienta `get_datetime`

Ahora, pongamos en práctica lo aprendido. Crearemos una herramienta simple pero muy útil que devuelve la fecha y hora actual del servidor.

### 4.4.1. Objetivo

Crear una herramienta llamada `get_datetime` que pueda devolver la fecha y hora actual en diferentes formatos (JSON estructurado o texto simple).

### 4.4.2. Paso 1: Crear una Nueva Unidad

En nuestro proyecto de Delphi, crearemos una nueva unidad. La guardaremos con el nombre `uTool.DateTime.pas`. Esta unidad contendrá todo el código de nuestra nueva herramienta.

### 4.4.3. Paso 2: Escribir el Código de la Herramienta

Pega el siguiente código en el fichero `uTool.DateTime.pas`:

```
unit uTool.DateTime;

interface

uses
  System.SysUtils,
```

```

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

type
// 1. CLASE DE PARÁMETROS
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 usará; para llamar a la herramienta.
    FName := 'get_datetime';
    // La descripción que la IA leerá; para entender qué 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 parámetros.
    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 información 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€œError obteniendo la fecha y hora: ' +
E.Message).Build;
    end;
end;

end.

```

## 4.5. Paso 3: Integrar la Nueva Herramienta en el Servidor

Ahora que nuestra herramienta está creada, necesitamos que el servidor principal la conozca.

1. **Añadir la Unidad al Proyecto:** Abre el fichero AiMCPServerDemo.dpr.
2. **Declarar la Unidad en uses:** Busca la sección uses al principio del fichero y añade uTool.DateTime a la lista.

```

uses
    System.SysUtils,
    System.Classes,
    uTool.FileAccess in 'uTool.FileAccess.pas',
    uTool.SysInfo in 'uTool.SysInfo.pas',
    uTool.DateTime in 'uTool.DateTime.pas', // <-- AÃ“DIR 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';

```

3. **Llamar al Procedimiento de Registro:** Ve al procedimiento RegisterAllToolsAndResources y añade la llamada a la función de registro de nuestra nueva unidad.

```

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“DIR ESTA LÃ“NEA

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

## 4.6. Compilando y Ejecutando el Servidor Actualizado

¡Eso es todo! Ahora puedes compilar y ejecutar tu proyecto AiMCPServerDemo.exe. Cuando el servidor se inicie, verás en la consola el mensaje "Registering tools and resources...". Internamente, ya habrá cargado y registrado nuestra nueva herramienta get\_datetime.

Cualquier cliente MCP (como el Playground de Claude o MakerAI Delphi Suite) que se conecte a este servidor ahora podrá ver y utilizar la herramienta `get_datetime` para consultar la hora del sistema donde se ejecuta el servidor.