

# SindByte MCP Server

Die einfache KI-Schaltzentrale für Menschen, die schnell  
Ergebnisse wollen.



Version: 2026-02-11 | Produktname: SindByte MCP Server (ehem. MCP Server 01)

## Impressum:

© Theo Gottwald

Herrenstr.11

76706 Dettenheim

Tel.07247-9851112

<http://smart-ai-robot.com>

[Info@smart-package.com](mailto:Info@smart-package.com)

# SindByte's Features in Kurz

## Kernfunktionen

1. **Lokaler MCP-Server** – läuft auf dem eigenen PC, keine Cloud-Abhängigkeit
2. **KI-Modell-Anbindung** – unterstützt lokale Modelle (LM Studio) und Cloud-Modelle (OpenAI)
3. **HTTP-API** – eigene Skripte können per POST auf `/mcp` zugreifen
4. **Integrierter Config-Editor** – einfache Verwaltung von Einstellungen und API-Keys

## Datei- & Textoperationen

1. **Dateien lesen/schreiben** – vollständige Dateiverwaltung (CRUD-Operationen)
2. **Textbearbeitung** – Suchen, Ersetzen, Regex-Operationen, Zeilenauswahl
3. **Memory/Storage** – temporäre und permanente Datenspeicherung mit Key-Value-System
4. **Clipboard-Integration** – direkter Zugriff auf die Windows-Zwischenablage

## Automatisierung & Workflows

1. **IQ-Workflows** – erweiterte Reasoning-Funktionen (Ensemble, Chain-of-Thought, Self-Critique, Tree-of-Thought) uvm.
2. **Timer-Tools** – zeitgesteuerte Aufgaben und verzögerte Ausführung
3. **System-Tools** – Prozessverwaltung, Registry-Zugriff, Umgebungsvariablen, Services steuern
4. **Batch/PowerShell-Ausführung** – direkte Integration von Skripten

## Computer-Use (GUI-Automatisierung)

1. **Screenshots** – Bildschirmaufnahmen für Vision-Models
2. **Fenster-Lokalisierung** – automatisches Finden und Positionieren von Fenstern
3. **Maus- & Tastatur-Simulation** – UI-Automatisierung durch simulierte Eingaben

## Web-Integration

1. **Web-Tools** – HTTP-Requests, Header-Auslesen, Text-Extraktion aus Webseiten
2. **Bild-Workflows** – Bildgenerierung und -verarbeitung über KI-Modelle

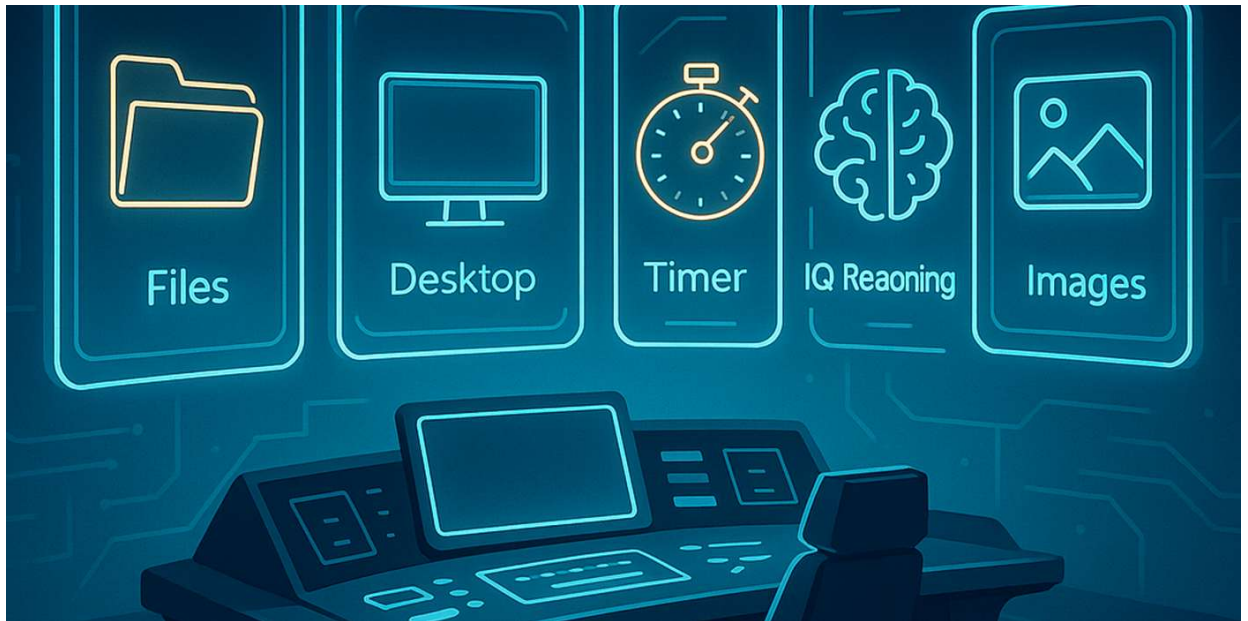
## Architektur & Sicherheit

1. **Kategorien-Modus** – nur benötigte Tool-Kategorien aktivieren (spart Tokens)
2. **Whitelisting** – präzise Kontrolle über erlaubte Tools und Befehle
3. **Job-Queue** – asynchrone Verarbeitung mit Status-Tracking
4. **SSE-Notifications** – Echtzeit-Statusupdates an Clients

\*Liste ist nicht komplett da die Software ständig erweitert wird.

# 1) Was ist SindByte?

SindByte ist ein lokaler MCP-Server. Er verbindet KI-Modelle mit echten Werkzeugen auf deinem PC: Dateien lesen/schreiben, Screenshots machen, Text bearbeiten, Timer ausführen, IQ-Workflows starten und vieles mehr.



Kurz gesagt: Deine KI bekommt Hände und Augen auf dem Desktop.

# 2) Wofür ist das gut?

- Webseiten schneller bauen (inkl. Bild- und Text-Workflow).
- Admin-Aufgaben automatisieren (Dateien, Logs, Prozesse).
- KI-Workflows planen: zeitgesteuert oder sofort.
- Lokale Modelle (LM Studio) und Cloud-Modelle (OpenAI) nutzen.
- Computer-Use: Fenster finden, Screenshots, Mouse/Keyboard-Simulation.

### 3) Installation in 5 Minuten



#### Schnellstart

1. Server an einen Ort deiner Wahl entpacken und eine Verknüpfung auf dem Desktop erstellen.
2. **`SindByte.exe`** mit der Verknüpfung starten.
3. LM-Studio sollte installiert sein.
4. Klicke auf den Button "Installieren". Dadurch trägt sich der Server bei LM-Studio ein.
5. Der Server kann nun von Modellen, die Tools verwenden können verwendet werden.

#### Achtung:

Wenn Modelle die Tools nicht verwenden können liegt es in der Regel an ungültigen Prompt-Vorlagen (Jinja). Bei Problemen teste das Modell einfach mit dem bei LM-Studio beiliegenden "js-code Sandbox".

Funktioniert diese geht in der Regel der SindByte Server auch.

#### Optional:

Starte die **`MCP\_ConfigEditor.exe`** die dem Program beiliegt. Dort kannst du Tools deaktivieren. Und Einstellungen vornehmen.

Es empfiehlt sich nur die Tools aktiviert zu lassen die du benötigst.

Dadurch sparst du Kontextspeicher im Model.

## 4) Alles Lokal – oder doch nicht?

**Fast alles** funktioniert mit SindByte local also ohne OpenAI-Key.

Es gibt aber ein paar Ausnahmen – diese must du nicht nutzen – diese funktionieren derzeit nur mit einem OpenAI Key.

### **Das geht nur mit OpenAI Key:**

1. Transcribe im IQTools-Fenster
2. IQTools-Bildgenerierung und
3. IQTools-Bild-Editierungen
4. Die GUI-Elementerkennung wird wohl mit den derzeitigen lokalen Modellen nicht ausreichend gut funktionieren. Deswegen hilft es hier OpenAI verwenden zu können.
5. Du kannst für die IQTools OpenAI verwenden, dadurch könnte dein lokales Model in Notfall "OpenAI zu Rate ziehen".

### **Das geht auch ohne OpenAI-Key:**

1. Eigentlich geht alles bis auf die genannten Dinge oben.
2. IQTools funktionieren grundsätzlich auch ohne OpenAI-Key. Diese benutzen dann einfach die lokalen Modelle in LM-Studio. Man kann bis zu zwei Modelle laden.

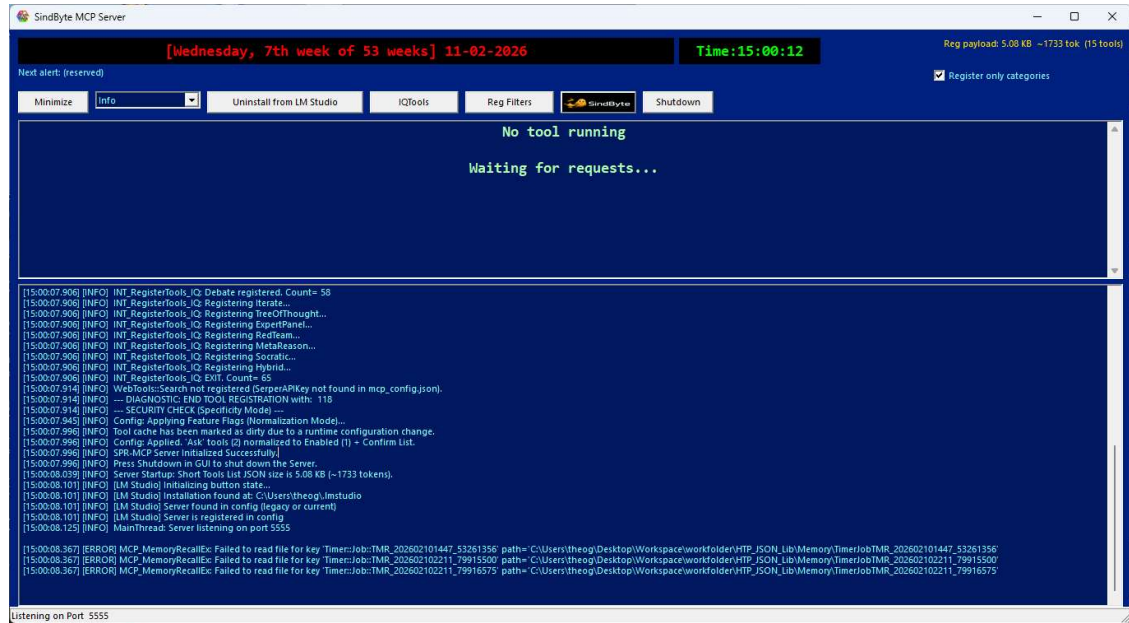
### **Bei OpenAI-Nutzung:**

1. API-Key im Config-Editor eintragen und speichern.
2. Client verbinden (z. B. LM Studio, Codex, VS Code, Roo Code, andere MCP-Clients).
3. Test: ``spr_server_assistant`` oder ``DataTools::Echo`` aufrufen.

## **Tipp für Einsteiger**

Im "Normal-Modus" starten, dann später auf "Kategorie-Modus" umstellen, und schauen ob dein Modell damit klar kommt. Das ist gut, wenn du Tokens sparen willst.

### 3) Wie sieht SindByte aus?



# SindByte's Features in Kurz


## 1) FileTools – Dateiverwaltung


Die FileTools sind dein direkter Zugang zum Dateisystem. Sie ersetzen nicht den Explorer, aber sie machen KI-gesteuerte Dateioperationen möglich – von automatisierten Backups bis zur Verarbeitung von Logdateien. Alle Tools arbeiten mit Unicode-Pfaden und beherrschen lange Dateinamen.

### **\*\*1. GetInfo\*\***

Prüft Metadaten einer Datei (Größe, Erstellungsdatum, Attribute).

☒ **Stärke:** Sehr schnell, kein Inhaltsladen nötig. Gut für Datei-Existenz-Checks vor aufwendigen Operationen.


 **Schwäche:** Gibt keine Inhalts-Hashs (MD5/SHA) zurück – für Integritätsprüfungen nicht ausreichend.

 **Tipp:** Vor jedem ReadFile erst GetInfo aufrufen, um Dateigröße zu prüfen (Schutz gegen Memory-Overflow bei riesigen Dateien).

### **\*\*2. GetLines\*\***

Liest beliebige Zeilenbereiche aus einer Datei (z.B. Zeile 100-150).

☒ **Stärke:** Speichereffizient bei großen Dateien – liest nicht die ganze Datei in den RAM.


 **Schwäche:** Bei UTF-8-Dateien mit variabler Byte-Länge pro Zeichen kann die Positionierung langsam werden.

 **Tipp:** Für Logfile-Analyse perfekt – lies immer nur das "Fenster" rund um den Timestamp.

### **\*\*3. FindText\*\***

Sucht Text in einer Datei (einfache Strings, keine Regex).

☒ **Stärke:** Schnell für erste Orientierung, case-sensitive/insensitive wählbar.

 **Schwäche:** Keine Regex-Unterstützung (dafür DataTools::RegexSearch verwenden).

💡 Tipp: Gut für Konfigurationsdateien (INI, CONF), wo du exakte Schlüsselwörter suchst.

#### **\*\*4. GetDirectoryListing\*\***

Listet Verzeichnisinhalte auf (Dateien + Ordner).

☑ Stärke: Unterstützt Filtermuster (\*.txt, report\_\*.log). Rekursiv-Modus verfügbar.

⚠ Schwäche: Bei Netzwerk-Laufwerken mit vielen Dateien (>10.000) kann es zu Timeouts kommen.

💡 Tipp: Immer mit Filter arbeiten, nie "alles auf einmal" bei großen Verzeichnissen.

#### **\*\*5. ReadFile\*\***

Liest eine komplette Datei in den Memory oder gibt sie als Text zurück.

☑ Stärke: Direkte Weiterleitung an MemoryTools oder KI-Modelle. Unterstützt UTF-8, UTF-16, ANSI.

⚠ Schwäche: Bei Dateien >100 MB sollte man zu Streaming/Chunking wechseln (sonst OutOfMemory).

💡 Tipp: Für Quellcode-Dateien ideal – komplette .py/.js/.bas-Dateien direkt an das KI-Modell schicken.

#### **\*\*6. WriteFile\*\***

Schreibt Text oder Memory-Inhalt in eine Datei (überschreibt oder neu).

☑ Stärke: Atomare Schreiboperation (temporäre Datei + Umbenennen) – keine halben Dateien bei Absturz.

⚠ Schwäche: Kein automatisches Backup der alten Version (muss manuell gemacht werden).

💡 Tipp: Vor WriteFile immer Copy als Backup machen, wenn es sich um wichtige Configs handelt.

#### **\*\*7. AppendToFile\*\***



Hängt Text an eine bestehende Datei an.

☑ Stärke: Perfekt für Logfiles und stetig wachsende Daten. Thread-sicher bei lokalem Dateisystem.

⚠ Schwäche: Bei Netzwerk-Shares kann es zu Konflikten kommen, wenn mehrere Prozesse gleichzeitig appenden.

💡 Tipp: Für "Write-Ahead-Logging" verwenden – erst in Datei schreiben, dann verarbeiten.

## **\*\*8. Delete\*\***

Löscht Dateien (mit Papierkorb-Option oder permanent).

☑ Stärke: Kann Wildcards verwenden (\*.tmp, old\_\*.bak).

⚠ Schwäche: Kein "Undo" nach permanentem Löschen (Vorsicht!).

💡 Tipp: Immer erst GetDirectoryListing mit dem Pattern machen, um zu sehen WAS gelöscht würde, dann Delete.

## **\*\*9. Move\*\***

Verschiebt Dateien (inkl. Umbenennen).

☑ Stärke: Funktioniert auch über Laufwerksgrenzen hinweg (Copy+Delete bei Bedarf).

⚠ Schwäche: Bei Unterbrechung (Netzwerkabbriss) kann die Datei "verloren" gehen (nur teilweise kopiert).

💡 Tipp: Für Archivierung ideal – verschiebe verarbeitete Dateien in "done/"-Ordner.

## **\*\*10. Copy\*\***

Kopiert Dateien (mit Überschreib-Option).

☑ Stärke: Behält Zeitstempel bei (optional), unterstützt Progress-Callbacks.

⚠ Schwäche: Keine integrierte Prüfsummen-Verifikation nach dem Kopieren.

💡 Tipp: Vor wichtigen Änderungen immer Copy auf .bak, dann arbeite am Original.

### **\*\*11. CreateDirectory\*\***

Erstellt Verzeichnisse (auch verschachtelt "a/b/c").

☑ Stärke: "Recursive"-Modus erstellt die ganze Kette auf einmal.

⚠ Schwäche: Keine Berechtigungs-Setzung (ACL) möglich – nur Standard-Rechte.

💡 Tipp: Vor WriteFile immer CreateDirectory für den Zielpfad aufrufen (idempotent).

### **\*\*12. SearchInFiles\*\***

Durchsucht mehrere Dateien nach Text (Regex-fähig).

☑ Stärke: Kombination aus GetDirectoryListing + FindText – very powerful für Code-Suche.

⚠ Schwäche: Kann bei binären Dateien (EXE, DLL) "Müll" finden – nur Textdateien empfohlen.

💡 Tipp: Mit \*.cs, \*.java, \*.py einschränken – dann hast du eine Code-Search-Engine.

### **\*\*13. FileExists\*\***

Schneller Existenz-Check (nur Boolean-Return).

☑ Stärke: Minimaler Overhead, ideal für "if exists then..." Logik in Skripten.

⚠ Schwäche: Unterscheidet nicht zwischen Datei und Verzeichnis (dafür GetInfo verwenden).

💡 Tipp: In Roboter-Skripten vor jeder Datei-Operation als Guard-Clause verwenden.

### **\*\*14. Zip\*\***

Komprimiert Dateien/Ordner zu einem ZIP-Archiv.

☑ Stärke: Standard-ZIP-Format, kompatibel mit Windows-Explorer. Passwort-Schutz möglich.


⚠ Schwäche: Kein 7z/RAR-Support. Bei sehr großen Dateien (>4GB) kann es zu Kompatibilitätsproblemen kommen.

💡 Tipp: Für Backups und Übertragungen ideal – zippe Logs vor dem Versand per Mail.

### **\*\*15. Unzip\*\***

Entpackt ZIP-Archive.

☒ **Stärke:** Unterstützt Unterverzeichnisse und behält Ordnerstruktur bei.

 **Schwäche:** Keine Überprüfung auf "Zip-Slip" (Path-Traversal) in alten Versionen – nur vertrauenswürdige Archive verwenden.

 **Tipp:** Immer erst GetInfo auf die ZIP-Datei, dann Unzip mit Zielverzeichnis-Check.

---


## **## 2) DataTools – Textverarbeitung und Datentransformation**

DataTools sind das "Schweizer Taschenmesser" für Textoperationen. Während FileTools mit Dateien arbeiten, operieren DataTools auf Strings und Datenstrukturen. Sie sind essenziell für ETL-Prozesse (Extract, Transform, Load), Log-Analysen und Datenbereinigung. Alle Tools beherrschen Unicode und können mit großen Textmengen (MB-Bereich) umgehen.

### **\*\*16. Echo\*\***

Gibt den Eingabetext unverändert zurück (Ping-Test).

☒ **Stärke:** Perfekt für Verbindungstests und Latenz-Messungen. Verbraucht minimale Ressourcen.

 **Schwäche:** Keine echte Funktionalität – nur für Diagnosezwecke.

 **Tipp:** Immer als erstes Tool nach Client-Verbindung aufrufen ("Hello World" für MCP).

### **\*\*17. GetInfo\*\***

Analysiert Text-Metadaten (Zeichenanzahl, Zeilen, Wörter, Encoding-Erkennung).

☒ **Stärke:** Schnelle Vorbestimmung der Datengröße vor aufwendigen Operationen. Erkennt BOM-Header.

⚠️ Schwäche: Encoding-Erkennung ist heuristisch – bei gemischten Encodings kann sie falsch liegen.

💡 Tipp: Vor Regex-Operationen aufrufen, um zu prüfen ob der Text überhaupt verarbeitbar ist (< 10 MB empfohlen).

#### **\*\*18. GetLines\*\***

Extrahiert Zeilenbereiche aus einem Text (nicht aus Dateien – dafür FileTools::GetLines).

☑️ Stärke: Arbeitet auf Memory-Ebene – sehr schnell für zwischengespeicherte Daten.

⚠️ Schwäche: Bei fehlenden Zeilenumbrüchen (eine riesige Zeile) kann es zu Memory-Problemen kommen.

💡 Tipp: Gut für "Pagination" von Ergebnissen – zeige nur Zeile 1-50, dann 51-100 usw.

#### **\*\*19. FindText\*\***

Einfache Textsuche in Strings (Case-sensitive/insensitive).

☑️ Stärke: Schneller als Regex für einfache Literalsuchen ("foo" finden).

⚠️ Schwäche: Keine Wildcards, kein Pattern-Matching – nur exakte Teilstrings.

💡 Tipp: Für Konfigurationsparsing nutzen, wo du genaue Schlüssel wie "Server=" suchst.

#### **\*\*20. SortList\*\***

Sortiert Listen (alphabetisch, numerisch, nach Länge).

☑️ Stärke: Unterstützt natürliche Sortierung (file2.txt kommt vor file10.txt).

⚠️ Schwäche: Bei sehr großen Listen (>100.000 Einträge) wird es speicherintensiv.

💡 Tipp: Mit "unique"-Option direkt Duplikate entfernen lassen – spart extra Aufruf.

#### **\*\*21. Replace\*\***

Ersetzt Text (einfache String-Ersetzung).

☑ **Stärke:** Schnell und vorhersagbar. Unterstützt mehrere Ersetzungen in einem Durchlauf.

⚠ **Schwäche:** Keine Regex-Ersetzung (dafür `RegexReplace` verwenden).

💡 **Tipp:** Für Templating nutzen – `{{USERNAME}}` durch tatsächlichen Namen ersetzen.

## **\*\*22. Split\*\***

Teilt Text anhand eines Delimiters auf (z.B. CSV, Tab-getrennt).

☑ **Stärke:** Flexibler als `Parse$` – gibt Arrays zurück. Unterstützt Multi-Char-Delimiter.

⚠ **Schwäche:** Bei verschachtelten Delimitern (CSV mit Kommas in Anführungszeichen) lieber `CsvToJson` nutzen.

💡 **Tipp:** Mit `max_splits`-Parameter begrenzen – nicht immer alles splitten, nur die ersten 5 Spalten.

## **\*\*23. Join\*\***

Fügt Array-Elemente zu einem String zusammen (Gegenstück zu `Split`).

☑ **Stärke:** Flexibles Trennzeichen (nicht nur Komma). Fügt auch mit Leerzeichen, Newlines, etc.

⚠ **Schwäche:** Bei `nil`/`NULL`-Werten im Array kann das Ergebnis unerwartet ausfallen.

💡 **Tipp:** Perfekt für SQL-IN-Klauseln erstellen: `JOIN(array, ", ") → 'a', 'b', 'c'`.

## **\*\*24. RegexSearch\*\***

Sucht mit regulären Ausdrücken (Pattern-Matching).

☑ **Stärke:** Vollständige Regex-Engine mit Capturing Groups. Gibt strukturierte Ergebnisse zurück.

⚠ **Schwäche:** Komplexe Regex mit Backtracking kann bei "catastrophic backtracking" hängen (Timeouts beachten!).

💡 **Tipp:** Immer mit Timeout-Parameter arbeiten. Patterns vorher auf [Regex101.com](https://www.regex101.com) testen.

## **\*\*25. RegexReplace\*\***

Ersetzt mit regulären Ausdrücken.

☑ **Stärke:** Capture-Gruppen können im Replacement verwendet werden (\$1, \$2).

⚠ **Schwäche:** Bei globalem Replace (alle Vorkommen) kann es bei großen Texten langsamer werden.

💡 **Tipp:** Für Data-Cleansing unverzichtbar – entferne alle nicht-druckbaren Zeichen: `[^\x20-\x7E]`.

## **\*\*26. NormalizeWhitespace\*\***

Bereinigt Whitespace (mehrere Leerzeichen → eines, Tabs → Spaces).

☑ **Stärke:** Wichtig für konsistente Textverarbeitung. Entfernt auch trailing spaces.

⚠ **Schwäche:** Kann intendierte Formatierung (z.B. in Code) zerstören.

💡 **Tipp:** Vor Tokenisierung für NLP/KI-Modelle anwenden – reduziert Token-Anzahl.

## **\*\*27. NormalizeNewlines\*\***

Konvertiert Zeilenumbrüche (CRLF → LF, oder umgekehrt, oder gemischt → einheitlich).

☑ **Stärke:** Behebt "Mixed line ending" Probleme zwischen Windows (CRLF) und Unix (LF).

⚠ **Schwäche:** Bei Binärdateien, die zufällig `\r\n` enthalten, kann es zu Datenkorruption kommen.

💡 **Tipp:** Vor Diff/Vergleichsoperationen immer normalisieren – sonst sind alle Zeilen "unterschiedlich".

## **\*\*28. Chunk\*\***

Teilt großen Text in gleich große Blöcke auf.

☑ **Stärke:** Essenziell für KI-Modelle mit Token-Limits (Context-Window). Überlappungsbereich einstellbar.

⚠ **Schwäche:** Kann Satzgrenzen ignorieren – Text wird "hart" geteilt (mid-word möglich).

💡 Tipp: Mit overlap=100 arbeiten, damit Kontext am Blockende nicht verloren geht.

#### **\*\*29. JsonSelect\*\***

Extrahiert Werte aus JSON via Pfad (Dot-Notation).

☑ Stärke: Einfacher als vollständiges Parsen. Unterstützt Arrays [0], Wildcards [\*].

⚠ Schwäche: Bei verschachtelten Arrays mit unterschiedlichen Strukturen kann es Fehlermeldungen geben.

💡 Tipp: Für schnelle Extraktion: `jsonSelect("data.users[0].email")` → direkter Wert.

#### **\*\*30. CsvToJson\*\***

Konvertiert CSV-Tabellen nach JSON-Array.

☑ Stärke: Berücksichtigt Anführungszeichen, Escaping, verschachtelte Kommas.

⚠ Schwäche: Bei fehlenden Header-Zeilen muss man Index-basiert arbeiten (weniger lesbar).

💡 Tipp: Mit `"header_row=true"` aufrufen, um aus Spalten 1,2,3 → named Properties zu machen.

#### **\*\*31. JsonToCsv\*\***

Konvertiert JSON-Array zurück zu CSV.

☑ Stärke: Flacht verschachtelte JSONs automatisch (dot-notation in Header).

⚠ Schwäche: Bei sehr tiefen Verschachtelungen werden die Spaltennamen extrem lang.

💡 Tipp: Gut für Excel-Export – JSON-API-Ergebnisse direkt in CSV für Business-User wandeln.

#### **\*\*32. ParseKeyValue\*\***

Parsed Key=Value Paare (z.B. Config-Dateien, INI-Style).

☑ Stärke: Toleriert Whitespace um das Gleichheitszeichen. Unterstützt Kommentare (#, ;).

⚠ Schwäche: Keine Unterstützung für Sections ([SectionName] wie in INI).

💡 Tipp: Für .env-Dateien und einfache Configs perfekt – gibt Dictionary/Map zurück.

### **\*\*33. TableToJson\*\***

Konvertiert Markdown-Tabellen oder ASCII-Tabellen nach JSON.

☑ Stärke: Erkennt automatisch Spaltenausrichtung (|:--| vs |--:|).

⚠ Schwäche: Bei verschmolzenen Zellen (colspan) oder komplexen Markdown-Tabellen Fehleranfällig.

💡 Tipp: Gut für API-Dokumentationen – kopiere Tabelle aus README, wandle in strukturiertes JSON um.

### **\*\*34. LoopCalc (Alias für MathTools::LoopCalc)\*\***

Führt Berechnungen über Zahlenreihen aus (z.B. Summe von 1 bis 100).

☑ Stärke: Serverseitige Berechnung spart Roundtrips zum KI-Modell. Unterstützt Formeln mit #loop Variable.

⚠ Schwäche: Nur numerische Operationen – keine String-Verarbeitung in der Schleife.

💡 Tipp: Für Batch-Berechnungen nutzen: "Berechne Quadratzahlen für 1..1000" → ein Aufruf, kein Chat.

---

## **## 3) MemoryTools – Speicherverwaltung und Datenaustausch**

MemoryTools bilden das "Kurzzeitgedächtnis" des MCP-Servers. Im Gegensatz zu FileTools, die auf die Festplatte schreiben, arbeiten MemoryTools im RAM oder in einem schnellen Temp-Store. Sie sind ideal für Datenübergaben zwischen verschiedenen KI-Workflows, für Caching von Zwischenergebnissen und für Situationen, wo du etwas speichern möchtest, ohne den



Datenträger zu belasten. Alle Memory-Tools unterstützen sowohl temporären Speicher (RAM, bei Serverneustart weg) als auch permanenten Speicher (lokal, überlebt Neustarts).

### **\*\*35. Store\*\***

Speichert einen Wert unter einem Schlüssel (temporär oder permanent).

☑ **Stärke:** Extrem schnell (RAM-Zugriff im Temp-Modus). Unterstützt TTL (Time-To-Live) für auto-expire.

⚠ **Schwäche:** Permanenter Speicher nutzt Dateien – bei sehr häufigen Schreibzugriffen Verschleiß auf SSDs.

💡 **Tipp:** Für Session-Daten immer TTL setzen (z.B. 3600 Sekunden), damit alter Müll nicht aufhäuft.

### **\*\*36. Recall\*\***

Ruft einen gespeicherten Wert ab (als WSTRING).

☑ **Stärke:** Automatische Cache-Erkennung – permanente Werte werden beim ersten Zugriff in RAM gecacht.

⚠ **Schwäche:** Bei nicht-existenten Keys kommt leerer String zurück – unterscheide zwischen "leer" und "nicht gefunden" mit RecallEx.

💡 **Tipp:** In Roboterskripten: Recall("lastResult") als Standard-Pattern für Ergebnisweitergabe.

### **\*\*37. Delete\*\***

Löscht einen oder mehrere Keys (Wildcards möglich).

☑ **Stärke:** Pattern-Löschung (z.B. "temp\_\*" löscht alle Temp-Daten auf einmal).

⚠ **Schwäche:** Kein "Undo" – gelöschte permanente Daten sind weg (kein Papierkorb).

💡 **Tipp:** Vor Delete immer ListKeys mit dem Pattern aufrufen, um zu sehen WAS gelöscht wird.

### **\*\*38. ListKeys\*\***

Listet alle gespeicherten Keys auf (mit Filteroption).

☑ Stärke: Unterscheidet zwischen TEMP und PERMANENT. Unterstützt Pattern-Matching.

⚠ Schwäche: Bei sehr vielen Keys (>10.000) kann die Liste groß werden – immer mit Filter arbeiten.

💡 Tipp: Mit "persistenceFilter=2" nur temporäre Keys anzeigen – gut für Aufräum-Routinen.

#### **\*\*39. FromFile\*\***

Lädt eine Datei direkt in den Memory (ohne Umweg über Client).

☑ Stärke: Schneller als FileTools::ReadFile + MemoryStore – Serverseitige Optimierung.

⚠ Schwäche: Datei muss auf dem Server-Rechner vorhanden sein (kein Remote-Zugriff).

💡 Tipp: Für "Upload"-Simulation: Client schickt nur Pfad, Server lädt direkt – spart Bandbreite.

#### **\*\*40. ToFile\*\***

Speichert Memory-Inhalt direkt in eine Datei.

☑ Stärke: Atomare Operation – entweder komplett geschrieben oder gar nicht (keine halben Dateien).

⚠ Schwäche: Überschreibt existierende Dateien ohne Warnung (außer "overwrite=false" ist gesetzt).

💡 Tipp: Perfekt für "Export"-Funktionen – KI-generierte Reports direkt als .txt/.html speichern.

#### **\*\*41. ToClipboard\*\***

Kopiert Memory-Inhalt in die Windows-Zwischenablage.

☑ Stärke: Unicode-kompatibel – behält Umlaute und Sonderzeichen bei.

⚠ Schwäche: Funktioniert nur auf Windows mit GUI-Session (nicht auf headless Servern).

💡 Tipp: In Kombination mit Screenshots nutzen – "Kopiere Ergebnis in Zwischenablage zum Einfügen".

#### **\*\*42. FromClipboard\*\***

Liest aus der Windows-Zwischenablage in den Memory.

☑ **Stärke:** Ermöglicht "Copy-Paste"-Workflows zwischen beliebigen Apps und der KI.

⚠ **Schwäche:** Liest nur Text – bei Bildern in der Zwischenablage kommt leerer String zurück.

💡 **Tipp:** Als "Quick Input" nutzen – User kopiert Text in beliebiger App, KI verarbeitet ihn sofort.

#### **\*\*43. GetLines\*\***

Extrahiert Zeilen aus einem Memory-Wert (ähnlich DataTools::GetLines, aber für Memory).

☑ **Stärke:** Kein erneutes Speichern nötig – arbeitet direkt auf dem Memory-Eintrag.

⚠ **Schwäche:** Bei sehr langen Zeilen (>10.000 Zeichen) kann die Performance leiden.

💡 **Tipp:** Gut für Log-Analysen im Memory – lies nur Zeile 1000-1100 aus einem großen Log.

#### **\*\*44. JSON\_Escape\*\***

Escapet einen String für sichere JSON-Nutzung (Anführungszeichen, Backslashes, etc.).

☑ **Stärke:** Verhindert JSON-Injections und Syntaxfehler bei dynamischer JSON-Erzeugung.

⚠ **Schwäche:** Macht den String länger – bei sehr großen Texten (>1MB) Performance-Impact.

💡 **Tipp:** IMMER vor dem Einbetten von User-Input in JSON aufrufen – Sicherheit zuerst.

#### **\*\*45. JSON\_Unescape\*\***

Entfernt JSON-Escaping (umgekehrte Operation zu JSON\_Escape).

☑ **Stärke:** Korrekte Handhabung von \uXXXX Unicode-Escapes.

⚠ **Schwäche:** Bei fehlerhaftem Input (unvollständiges Escaping) kann das Ergebnis unerwartet sein.

💡 Tipp: Nach JSON\_ExtractString-Aufrufen meist nicht nötig – die JSON-Library entescaped automatisch.

#### **\*\*46. Format\*\***

Löscht ALLE Memory-Einträge (Factory Reset für den Memory).

☑ Stärke: Schnelle Aufräumaktion – alles auf einmal weg. Unterscheidet TEMP/PERMANENT.

⚠ Schwäche: Keine Undo-Möglichkeit – "Format permanent" löscht unwiderruflich.

💡 Tipp: In Produktivumgebungen mit Vorsicht nutzen – erst ListKeys prüfen, dann Format.

---

#### **## 4) MathTools – Mathematische Berechnungen und Analysen**

MathTools bieten serverseitige mathematische Funktionen, die ohne Roundtrip zum KI-Modell ausgeführt werden. Das spart Zeit und Token, besonders bei repetitiven Berechnungen oder wenn numerische Präzision wichtig ist. Alle Tools arbeiten mit DOUBLE-Präzision (ca. 15 signifikante Stellen) und beherrschen sowohl einfache Formeln als auch komplexe numerische Methoden.

#### **\*\*47. ResolveFormula\*\***

Löst mathematische Formeln auf (z.B. "2+3\*4" oder "sin(pi/2)").

☑ Stärke: Umfangreiche Funktionsbibliothek (sin, cos, log, sqrt, etc.). Variablen-Substitution möglich.

⚠ Schwäche: Keine symbolische Mathematik (kein "x^2" auflösen nach x).

💡 Tipp: Für Konfigurationsberechnungen nutzen – "preis=netto\*(1+mwst/100)" direkt auswerten.

#### **\*\*48. LoopCalc\*\***

Führt Berechnungen über Zahlenreihen aus (Summen, Produkte, etc.).

☑ Stärke: Sehr effizient für "Berechne Summe von 1 bis 100000" – kein KI-Request nötig.

⚠ Schwäche: Nur numerische Operationen – keine String-Verarbeitung in der Schleife.

💡 Tipp: Mit benutzerdefinierten Variablen (#value, #loop) arbeiten für komplexe Aggregationsformeln.

#### **\*\*49. LoopStart\*\***

Initialisiert einen Schleifen-Zähler für kumulative Berechnungen.

☑ Stärke: Zustandsbasiert – mehrere parallele Schleifen möglich (unterschiedliche IDs).

⚠ Schwäche: Bei Serverneustart geht der Zustand verloren (nur temporärer Speicher).

💡 Tipp: In Kombination mit LoopStep für "Fortschrittsberechnungen" nutzen – "Berechne nächste 100 Iterationen".

#### **\*\*50. LoopStep\*\***

Führt einen Schleifenschritt aus (Inkrement + Berechnung).

☑ Stärke: Atomare Operation – Thread-safe für parallele Berechnungen.

⚠ Schwäche: Keine automatische Konvergenzprüfung – Endlos-Schleifen möglich wenn Schrittweite = 0.

💡 Tipp: Immer mit Abbruchbedingung (max\_iterations) aufrufen, um Endlosschleifen zu vermeiden.

#### **\*\*51. SmartSequence\*\***

Erkennt Muster in Zahlenreihen und prognostiziert nächste Werte.

☑ Stärke: Erkennt arithmetische, geometrische und rekursive Folgen. Nützlich für "Was kommt als Nächstes?".

⚠ Schwäche: Bei zufälligen oder nicht-mathematischen Sequenzen falsche Vorhersagen.

💡 Tipp: Gut für IQ-Tests oder Prüfnummern-Berechnungen – erkenne das Schema hinter 2, 6, 12, 20...

#### **\*\*52. RootFinder\*\***

Findet Nullstellen von Funktionen (Newton-Verfahren, Bisektion).

☑ Stärke: Automatische Verfahrenswahl basierend auf Funktionsverhalten. Konvergenz-Garantie für stetige Funktionen.

⚠ Schwäche: Bei Funktionen ohne Nullstelle oder mit Sprungstellen kann es fehlschlagen.

💡 Tipp: Für "Bei welchem Preis ist der Gewinn = 0?" – Break-Even-Analysen direkt berechnen.

#### **\*\*53. Optimizer\*\***

Findet Minima oder Maxima einer Funktion (Numerische Optimierung).

☑ Stärke: Unterstützt Einschränkungen (Constraints) – z.B. "x muss zwischen 0 und 100 liegen".

⚠ Schwäche: Kann in lokalen Minima stecken bleiben bei komplexen Funktionen mit vielen Peaks.

💡 Tipp: Für Ressourcen-Optimierung nutzen – "Wieviele Arbeiter minimieren die Kosten bei gegebener Nachfrage?"

#### **\*\*54. Integrate\*\***

Berechnet bestimmte Integrale (Fläche unter Kurve).

☑ Stärke: Adaptive Simpson-Quadratur – hohe Genauigkeit auch bei komplexen Kurven.

⚠ Schwäche: Bei unstetigen Funktionen (Sprüngen) oder Singularitäten (Division durch 0) kann es fehlschlagen.

💡 Tipp: Für "Wie viel Strom wurde zwischen 14:00 und 16:00 verbraucht?" – Fläche unter der Lastkurve.

#### **\*\*55. MonteCarlo\*\***

Führt Monte-Carlo-Simulationen durch (wahrscheinlichkeitsbasierte Risikoanalyse).

☑ **Stärke:** Modelliert Unsicherheiten – perfekt für Risikoanalysen und Prognosen mit Konfidenzintervallen.

⚠ **Schwäche:** Ergebnisse sind stochastisch (leicht variabel bei jedem Lauf). Viele Iterationen nötig für Genauigkeit.

💡 **Tipp:** Für "Wie wahrscheinlich ist ein Budgetüberschreitung?" – Simuliere 10.000 Szenarien mit verteilten Eingaben.

---

## ## 5) WebTools – Internetzugriff und Webscraping

WebTools ermöglichen dem MCP-Server direkte HTTP-Kommunikation und Web-Inhaltsverarbeitung. Sie sind essenziell für Recherche-Aufgaben, API-Integrationen und das Anreichern von KI-Kontext mit aktuellen Web-Daten. Alle Tools basieren auf WinHTTP und unterstützen HTTPS, Redirects sowie verschiedene Encodings (UTF-8, ISO-8859-1, etc.).

### \*\*56. FetchText\*\*

Ruft eine Webseite oder API ab und gibt den Text zurück.

☑ **Stärke:** Automatische HTML-zu-Text-Konvertierung optional. Entfernt Script-Tags und Navigation.

⚠ **Schwäche:** Kein JavaScript-Rendering – dynamische SPAs (React, Vue) liefern nur den initialen HTML-Skelett.

💡 **Tipp:** Für statische Seiten und APIs ideal. Bei JS-lastigen Seiten zuerst prüfen ob eine API-Alternative existiert.

### \*\*57. ExtractText\*\*

Extrahiert lesbaren Text aus HTML (ohne HTTP-Request – rein lokale Verarbeitung).

☑ Stärke: Sehr schnell – arbeitet auf bereits geladenem HTML. Entfernt Tags, behält Struktur bei.

⚠ Schwäche: Tabellen werden oft linearisiert – relationale Datenstruktur geht verloren.

💡 Tipp: Nach FetchText aufrufen: "FetchText roh → ExtractText bereinigt → KI analysiert".

#### **\*\*58. FetchHeaders\*\***

Ruft nur HTTP-Header ab (HEAD-Request), nicht den Body.

☑ Stärke: Extrem schnell – prüfe Content-Type, Last-Modified oder Rate-Limits ohne Daten zu laden.

⚠ Schwäche: Nicht alle Server unterstützen HEAD-Requests korrekt (manchmal wird 405 zurückgegeben).

💡 Tipp: Vor großen Downloads: "FetchHeaders → Content-Length prüfen → nur bei Bedarf FetchText".

#### **\*\*59. Search\*\***

Führt Websuchen durch (anonymisiert über konfigurierbare Suchanbieter).

☑ Stärke: Liefert strukturierte Ergebnisse (Titel, URL, Snippet). Keine Browser-Notwendigkeit.

⚠ Schwäche: Rate-Limits bei Suchmaschinen beachten – bei zu vielen Requests kann die IP temporär blockiert werden.

💡 Tipp: Mit "site:example.com" Suchoperator nutzen für gezielte Domain-Suche statt allgemeiner Suche.

#### **\*\*60. FetchMany\*\***

Ruft mehrere URLs parallel ab (Batch-HTTP).

☑ Stärke: Deutlich schneller als serielle Aufrufe – parallele Connection-Pool-Nutzung.

⚠ Schwäche: Bei zu vielen parallelen Requests (>10) kann der Zielservice 429 (Too Many Requests) senden.



💡 Tipp: Für "Lese die letzten 5 Blog-Posts" ideal – ein Aufruf, alle Inhalte gleichzeitig geladen.

---

## ## 6) SystemTools – Systemverwaltung und Administration

SystemTools bieten direkten Zugriff auf das Windows-Betriebssystem – von Prozessverwaltung über Registry-Operationen bis hin zur Dienststeuerung. Diese Tools sind besonders mächtig und sollten deshalb mit Vorsicht eingesetzt werden. Sie erfordern oft erhöhte Rechte (Administrator) und können das System verändern. Ideal für IT-Administratoren, DevOps-Automatisierung und Systemüberwachung.

### \*\*61. GetEnvironmentVariable\*\*

Liest Umgebungsvariablen aus (z.B. PATH, USERNAME, TEMP).

☒ **Stärke:** Zugriff auf sowohl User- als auch System-Variablen. Auflösung von dynamischen Variablen (%VAR%).

⚠️ **Schwäche:** Bei nicht-existenten Variablen leerer String – unterscheide zwischen "leer" und "nicht gesetzt".

💡 Tipp: Für portable Skripte nutzen – "%APPDATA%\MyApp" auflösen statt feste Pfade zu verwenden.

### \*\*62. ExecuteCommand\*\*

Führt beliebige Systembefehle aus (CMD-Wrapper).

☒ **Stärke:** Volle Shell-Leistung – alle internen Befehle (dir, echo, etc.) verfügbar.

⚠️ **Schwäche:** Sicherheitsrisiko bei unsaniertem Input (Command Injection möglich). Immer Parameter validieren!

💡 Tipp: Mit Timeout arbeiten – "cmd /c ping 8.8.8.8" sollte nicht ewig hängen können.

### **\*\*63. RunPowerShell\*\***

Führt PowerShell-Skripte aus (volle .NET-Leistung).

☒ **Stärke:** Deutlich mächtiger als CMD – Zugriff auf WMI, .NET, COM-Objekte.

⚠ **Schwäche:** Execution Policy kann blockieren. Skripte müssen ggf. signiert oder Policy angepasst werden.

💡 **Tipp:** Für komplexe Admin-Aufgaben bevorzugen – "Get-Process | Where-Object {\$\_.CPU -gt 100}"

### **\*\*64. RunBatchFile\*\***

Führt .bat oder .cmd Dateien aus.

☒ **Stärke:** Kompatibilität mit bestehenden Legacy-Skripten. Keine Modifikation nötig.

⚠ **Schwäche:** Fehlerlevel (ERRORLEVEL) muss explizit geprüft werden – automatisches Exception-Handling gibt es nicht.

💡 **Tipp:** Mit "echo off" und expliziten Exit-Codes arbeiten, um Erfolg/Misserfolg zu signalisieren.

### **\*\*65. ListProcesses\*\***

Listet laufende Prozesse mit Details (PID, Speicher, CPU-Zeit).

☒ **Stärke:** Echtzeit-Informationen. Filter nach Name oder Speicherverbrauch möglich.

⚠ **Schwäche:** Bei sehr schnell startenden/beendenden Prozessen kann die Liste inkonsistent sein (Snapshot-Problem).

💡 **Tipp:** Für "Warum ist mein PC langsam?" – Sortiere nach WorkingSet (Speicher) oder CPU-Zeit.

### **\*\*66. ListServices\*\***

Listet Windows-Dienste auf (Status: Running/Stopped, Starttyp).

☒ **Stärke:** Zeigt sowohl DisplayName als auch ServiceName (kurz/lang) für bessere Identifikation.

⚠ Schwäche: Bei beschädigter Service-Datenbank können Fehler auftreten (selten).

💡 Tipp: Mit Filter "Status=Stopped && StartType=Auto" – finde Dienste, die hätten laufen sollen.

#### **\*\*67. StartService\*\***

Startet einen Windows-Dienst (z.B. SQL Server, Apache).

☒ Stärke: Wartet auf Bestätigung (Status Change Pending → Running). Timeout konfigurierbar.

⚠ Schwäche: Startet keine abhängigen Dienste automatisch (im Gegensatz zu `sc start` mit Dependencies).

💡 Tipp: Immer `ListServices` vorher aufrufen, um sicherzustellen dass der Dienst existiert und gestoppt ist.

#### **\*\*68. StopService\*\***

Stoppt einen laufenden Windows-Dienst.

☒ Stärke: Graceful Shutdown – wartet auf ordnungsgemäße Beendigung.

⚠ Schwäche: Bei hängenden Diensten kann der Timeout überschritten werden – dann harter Kill nötig.

💡 Tipp: Für Updates/Wartung: "StopService → Warte auf Stopped → Starte Setup → StartService".

#### **\*\*69. RestartService\*\***

Kombination aus Stoppen und Starten (Recycling).

☒ Stärke: Atomare Operation – wenn Stoppen fehlschlägt, wird nicht versucht zu starten.

⚠ Schwäche: Kein "Force-Restart" – bei nicht reagierenden Diensten muss erst `StopService` mit Force versucht werden.


💡 Tipp: Für "Habe ich es schon ausprobiert?" – klassischer IT-Crowd-Fix für speicherleckende Dienste.

### **\*\*70. GetServiceDetails\*\***

Detaillierte Informationen zu einem Dienst (Pfad, Beschreibung, Account).

☒ **Stärke:** Zeigt den tatsächlichen ausführbaren Pfad – wichtig für Malware-Analyse.


 **Schwäche:** Bei Diensten mit Leerzeichen im Pfad kann die Parsing-Logik komplex werden.


 **Tipp:** Für Sicherheitsaudits – prüfe ob Dienste aus ungewöhnlichen Pfaden (nicht C:\Windows) laufen.

### **\*\*71. IdentifySvchostService\*\***

Findet heraus, WELCHER Dienst in einem svchost.exe-Prozess läuft.

☒ **Stärke:** Löst das "svchost.exe frisst 100% CPU"-Rätsel auf. Zeigt alle Services in der Gruppe.


 **Schwäche:** Bei Windows-Versionen mit geschützten Prozessen (Secure Process) kann der Zugriff verweigert werden.


 **Tipp:** Immer mit ListProcesses kombinieren – "Hohe CPU bei PID 1234 → IdentifySvchostService → Täter gefunden".

### **\*\*72. IdentifyConhostParent\*\***

Findet das Elternprogramm einer conhost.exe (Konsolenfenster).

☒ **Stärke:** Unterscheidet legitime Konsolen (CMD, PowerShell) von möglichen Malware-Konsolen.

 **Schwäche:** Bei sehr schnell beendeten Prozessen kann der Parent bereits weg sein (Race Condition).

 **Tipp:** Für Verdachtsfälle – "Warum öffnet sich ein schwarzes Fenster?" – nachvollziehen wer es startet.

### **\*\*73. ListRegistrySubKeys\*\***

Listet Unterschlüssel eines Registry-Zweigs auf.

☒ **Stärke:** Unterstützt sowohl HKLM (Machine) als auch HKCU (User). Rekursiv-Option verfügbar.

⚠ Schwäche: Bei sehr großen Keys (z.B. Classes) kann die Liste extrem lang werden – Performance-Impact.

💡 Tipp: Für Software-Inventarisierung – "HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall" durchlaufen.

#### **\*\*74. ListRegistryValues\*\***

Listet die Werte (Name, Typ, Daten) eines Registry-Keys auf.

☑ Stärke: Zeigt Datentypen an (REG\_SZ, REG\_DWORD, REG\_BINARY) – wichtig für korrekte Interpretation.

⚠ Schwäche: Bei REG\_MULTI\_SZ (String-Arrays) kann die Darstellung komplex werden.

💡 Tipp: Für Config-Backups – exportiere wichtige Registry-Zweige vor Systemänderungen.

#### **\*\*75. CreateRegistryKey\*\***

Erstellt einen neuen Registry-Schlüssel.

☑ Stärke: Erstellt fehlende Parent-Keys automatisch (rekursive Erstellung).

⚠ Schwäche: Keine Berechtigungsprüfung im Voraus – bei fehlenden Rechten kommt es erst zur Laufzeit zum Fehler.

💡 Tipp: Für Software-Installationen – "HKCU\Software\MyCompany\MyApp" mit Defaults anlegen.

#### **\*\*76. DeleteRegistryValue\*\***

Löscht einen einzelnen Wert aus einem Registry-Key.

☑ Stärke: Sicherer als DeleteRegistryKey – nur der Wert wird entfernt, nicht der ganze Zweig.


⚠ Schwäche: Bei nicht-existenten Werten wird kein Fehler geworfen (idempotent, aber evtl. unerwartet).


💡 Tipp: Für "Uninstall"-Routinen – entferne nur die eigenen Werte, lass den Key für andere Apps bestehen.

### **\*\*77. DeleteRegistryKey\*\***

Löscht einen kompletten Registry-Zweig (rekursiv).

☒ **Stärke:** Bereinigt alle Subkeys und Values auf einmal.


 **Schwäche:** **\*\*GEFÄHRLICH\*\*** – kein Papierkorb, keine Undo-Funktion. Kann System unbrauchbar machen!


 **Tipp:** IMMER vorher ListRegistrySubKeys aufrufen und Ergebnis bestätigen lassen. Nie auf HKLM\System\CurrentControlSet anwenden!

### **\*\*78. WriteRegistryValue\*\***

Schreibt einen Wert in die Registry (DWORD, String, Binary).

☒ **Stärke:** Typ-Sicherheit – konvertiert automatisch (z.B. "1" → REG\_DWORD 0x00000001).


 **Schwäche:** Bei REG\_BINARY muss das Format exakt stimmen (Hex-String), sonst Datenkorruption.


 **Tipp:** Für Konfigurationsänderungen – "Aktiviere Feature X" durch Setzen eines DWORD-Werts auf 1.

### **\*\*79. ConfirmOperation\*\***

Zeigt einen Bestätigungsdialog (JA/NEIN) für kritische Operationen.

☒ **Stärke:** Human-in-the-Loop für gefährliche Aktionen – verhindert unbeabsichtigte Systemänderungen.

 **Schwäche:** Erfordert interaktiven Desktop – bei headless/Service-Modus funktioniert es nicht.

 **Tipp:** Vor DeleteRegistryKey, StopService oder RunPowerShell mit Admin-Rechten immer ConfirmOperation einbauen.

### **\*\*80. GetInstallPath\*\***

Ermittelt Installationspfade bekannter Software (Registry-Scan).

☒ **Stärke:** Findet Software unabhängig vom Installationsort (Program Files, AppData, etc.).

⚠ Schwäche: Bei portablen/portable Apps (ohne Registry-Eintrag) wird nichts gefunden.

💡 Tipp: Für "Finde Python.exe" oder "Wo ist Git installiert?" – vermeidet hartcodierte Pfade.

---

### ## 7) CUTools – Computer-Use und GUI-Automatisierung

CUTools (Computer Use Tools) ermöglichen die Steuerung der grafischen Benutzeroberfläche von Windows. Sie sind das "Auge und die Hand" der KI auf dem Desktop – Screenshots erstellen, Fenster finden, Maus und Tastatur simulieren. Diese Tools sind essenziell für RPA (Robotic Process Automation), visuelle Regressionstests und Aufgaben, die traditionell nur ein Mensch am Bildschirm erledigen konnte.

#### \*\*81. Screenshot\*\*

Erstellt Bildschirmaufnahmen (ganzer Bildschirm, Fenster oder Bereich).

☒ Stärke: Unterstützt verschiedene Modi (Fullscreen, aktives Fenster, freier Bereich).

Kompatibel mit Vision-Modellen.

⚠ Schwäche: Bei mehreren Monitoren muss der Ziel-Monitor angegeben werden – sonst wird nur der Primärmonitor erfasst.

💡 Tipp: Immer mit "parse\_mode=vision" für KI-Analyse nutzen – "Was siehst du auf dem Screenshot?" vor der KI senden.

#### \*\*82. GetFormats\*\*

Ermittelt verfügbare Bildformate für Screenshots (PNG, JPEG, BMP).

☒ Stärke: Zeigt qualitätsrelevante Details (Kompression, Transparenz-Unterstützung).

⚠ Schwäche: Keine direkte Bildbearbeitung – nur Metainformationen.

💡 Tipp: Vor automatisierter Screenshot-Erzeugung prüfen – PNG für Text (scharf), JPEG für Fotos (klein).

### **\*\*83. MouseSim\*\***

Simuliert Mausclicks und Mausbewegungen.

☑ **Stärke:** Unterstützt Linksklick, Rechtsklick, Doppelklick, Drag-&-Drop. Koordinaten oder relativen Offset.

⚠ **Schwäche:** Bei DPI-Skalierung (125%, 150%) müssen Koordinaten umgerechnet werden – sonst "daneben" klicken.

💡 **Tipp:** Immer mit Screenshot kombinieren – "Mache Screenshot → KI findet Button → MouseSim klickt Koordinaten".

### **\*\*84. KeySim\*\***

Simuliert Tastatureingaben (Text schreiben, Shortcuts).

☑ **Stärke:** Sendet Unicode – Umlaute und Sonderzeichen korrekt. Unterstützt Modifier (Strg+C, Alt+Tab).

⚠ **Schwäche:** Bei sicherheitssensiblen Anwendungen (Banking, Passwortfelder) kann die Eingabe blockiert werden.

💡 **Tipp:** Für "Formular ausfüllen" nutzen – Tab zum nächsten Feld, dann Text eingeben, wiederholen.

### **\*\*85. GDIPlus\_Image\*\***

Bildverarbeitung (Resize, Crop, Format-Konvertierung).

☑ **Stärke:** Hochwertige Skalierung (bicubic). Wasserzeichen-Overlay möglich.

⚠ **Schwäche:** Keine fortgeschrittene Bildanalyse (kein OCR, kein Gesichtserkennung) – dafür Vision-Modelle nutzen.

💡 **Tipp:** Vor Upload zu KI-Modellen nutzen – "Resize auf 1024x1024" spart Tokens und Bandbreite.

Verstanden – TaskTools sind SLL-intern. Ich mache mit **\*\*ClipboardTools\*\*** weiter:



---


## ## 8) ClipboardTools – Zwischenablage-Verwaltung


ClipboardTools bieten direkten Zugriff auf die Windows-Zwischenablage. Sie ermöglichen Datenaustausch zwischen dem MCP-Server und beliebigen Windows-Anwendungen ohne Dateisystem- oder Netzwerk-Overhead. Ideal für "Copy-Paste-Workflows", bei denen der Benutzer Inhalte in einer Anwendung kopiert und die KI sie sofort verarbeitet.

### \*\*86. Read\*\*

Liest Text aus der Windows-Zwischenablage.

☒ **Stärke:** Unicode-kompatibel – behält Umlaute, Emojis und Sonderzeichen bei. Schneller als temporäre Dateien.

 **Schwäche:** Liest nur Text – wenn Bilder oder formatierter Text (RTF) in der Zwischenablage sind, wird nur der Plain-Text extrahiert.


 **Tipp:** Als "Quick Input" nutzen – User kopiert Code-Fehler aus IDE, KI analysiert sofort mit Read.

### \*\*87. Write\*\*

Schreibt Text in die Windows-Zwischenablage.

☒ **Stärke:** Überschreibt vorherigen Inhalt sauber. Andere Anwendungen können sofort darauf zugreifen.

 **Schwäche:** Keine Formatierung (nur Plain-Text) – kein RTF, kein HTML mit Styling.

 **Tipp:** Für "Ergebnis kopieren" nutzen – KI generiert SQL-Query, Write legt sie in Zwischenablage, User fügt in Management Studio ein.

### \*\*88. Clear\*\*

Leert die Zwischenablage (sichereres Löschen).

☑ **Stärke:** Entfernt sensible Daten sofort aus dem Speicher. Wichtig für Passwörter oder Tokens.

⚠ **Schwäche:** Kann nicht verhindern, dass Zwischenablagen-Manager (Drittanbieter) bereits historisiert haben.

💡 **Tipp:** Nach Verarbeitung sensibler Daten immer Clear aufrufen – "Read → Process → Clear" Pattern.

#### **\*\*89. GetFormats\*\***

Zeigt verfügbare Formate in der Zwischenablage an (Text, Unicode, Bitmap, etc.).

☑ **Stärke:** Erkennt, WAS in der Zwischenablage liegt, bevor man liest. Vermeidet Fehler bei falschem Format.

⚠ **Schwäche:** Zeigt nur an – konvertiert nicht automatisch. Bei "Bild in Zwischenablage" muss man selbst entscheiden, was zu tun ist.

💡 **Tipp:** Vor Read prüfen – "if CF\_UNICODETEXT available → Read, else → Fehlermeldung 'Bitte Text kopieren'".

---

#### **## 9) TimerTools – Zeitgesteuerte Aufgaben**

TimerTools ermöglichen verzögerte und wiederkehrende Ausführungen. Sie sind das "Wecker"-System des MCP-Servers – nützlich für Erinnerungen, zeitgesteuerte Berichte, verzögerte Aktionen und periodische Wartungsaufgaben. Alle Timer sind persistiert (überleben kurze Server-Neustarts) und können asynchron überwacht werden.

#### **\*\*90. SetTimer\*\***

Erstellt einen neuen Timer (einmalig oder wiederholend).

☑ **Stärke:** Flexible Zeitangaben (absolute Uhrzeit, relative Verzögerung, Cron-ähnliche Muster).

⚠ Schwäche: Bei System-Schlaf/Hibernate werden Timer unterbrochen – kein Echtzeit-Garant bei Laptops.

💡 Tipp: Für "Erinnere mich in 30 Minuten" oder "Starte Backup jeden Tag um 02:00" nutzen.

#### **\*\*91. ListTimers\*\***

Zeigt alle aktiven Timer mit Status (wartend, überfällig, ausgeführt).

☒ Stärke: Übersicht über Due-Time, Intervall und zugehörige Aktion.

⚠ Schwäche: Bei sehr vielen Timern (>100) wird die Liste unübersichtlich – Filter nach Status empfohlen.

💡 Tipp: Im Dashboard anzeigen – "3 Timer aktiv, nächster läuft in 5 Minuten".

#### **\*\*92. CancelTimer\*\***

Bricht einen geplanten Timer ab (Löschung).

☒ Stärke: Sofortige Entfernung – Timer wird nicht mehr ausgeführt, auch wenn er kurz bevorsteht.

⚠ Schwäche: Bereits gestartete Timer können nicht mehr gestoppt werden (nur geplante).

💡 Tipp: Für "Doch nicht"-Szenarien – User plant Backup, bricht aber kurz vorher ab.

#### **\*\*93. CheckTimer\*\***

Prüft Status eines Timers (verbleibende Zeit, Ausführungszähler).

☒ Stärke: Polling-möglich – Client kann regelmäßig nachfragen ohne SSE/WebSocket.

⚠ Schwäche: Keine Push-Benachrichtigung – Client muss aktiv abfragen (außer SSE wird separat genutzt).

💡 Tipp: Für "Wie lange noch?" Anzeigen – verbleibende Zeit bis zur nächsten Ausführung anzeigen.

---

## ## 10) TextTools – Erweiterte Textverarbeitung

TextTools ergänzen die DataTools um spezialisierte Operationen für komplexe Texttransformationen. Der Fokus liegt auf strukturierten Änderungen an Dokumenten – insbesondere das Anwenden von Diffs/Patches. Diese Tools sind essenziell für Code-Reviews, automatisierte Refactorings und Versionierungs-Workflows, bei denen präzise Textmanipulationen nötig sind.

### \*\*94. ApplyDiff\*\*

Wendet ein Unified-Diff-Format auf einen Text an (Patch-Operation).

☑ **Stärke:** Präzise Änderungen – nur die markierten Zeilen werden modifiziert, der Rest bleibt unverändert. Unterstützt Kontext-Zeilen für robustes Matching.

⚠ **Schwäche:** Bei stark abweichendem Ausgangstext (Datei wurde zwischenzeitlich geändert) schlagen die Kontext-Matches fehl.

💡 **Tipp:** Immer mit "strict\_mode=false" arbeiten, wenn möglich – toleriert kleine Abweichungen in den Kontextzeilen.

### \*\*95. ApplyUnifiedDiff\*\*

Spezialisierte Variante für Git-kompatible Unified-Diffs mit erweiterter Metadatenverarbeitung.

☑ **Stärke:** Erkennt Git-Diff-Header (Index-Zeilen, Mode-Changes) und ignoriert sie automatisch. Unterstützt mehrere Hunks (Änderungsböcke) in einem Patch.

⚠ **Schwäche:** Bei Binärdateien oder speziellen Git-Attributen (LFS) nicht anwendbar – nur für Textdateien.

💡 **Tipp:** Für "KI schlägt Code-Änderung vor" Workflow: KI generiert Unified-Diff → ApplyUnifiedDiff wendet an → Git zeigt Diff zur Review.

---

## ## 11) IQTools – Intelligenz-Qualitäts-Tools (KI-Reasoning)

IQTools sind die "Denkwerkzeuge" für KI-Modelle. Sie implementieren verschiedene Reasoning-Strategien, um die Qualität von KI-Antworten zu verbessern – von einfachen Selbstkontrollen bis zu komplexen Multi-Agent-Debatten. Alle IQTools können mit verschiedenen Modellen kombiniert werden (lokale LM Studio + Cloud-OpenAI) und unterstützen asynchrone Verarbeitung mit Fortschrittsbenachrichtigungen.

### \*\*96. ChainOfThought\*\*

Zerlegt komplexe Probleme in Zwischenschritte ("Lass mich das Schritt für Schritt durchgehen...").

☑ **Stärke:** Deutliche Verbesserung bei mathematischen und logischen Problemen. Transparente Nachvollziehbarkeit.

⚠ **Schwäche:** Erhöht Token-Verbrauch um 30-50%. Bei sehr einfachen Fragen ("Wie spät ist es?") Overkill.

💡 **Tipp:** Für "Warum ist das so?"-Fragen aktivieren – User sieht DENKPROZESS, nicht nur Ergebnis.

### \*\*97. SelfCritique\*\*

Lässt das Modell seine eigene Antwort überprüfen und verbessern.

☑ **Stärke:** Fängt logische Fehler und Widersprüche auf. Kostengünstig (nur ein zusätzlicher Durchlauf).

⚠ **Schwäche:** Das Modell kann seine eigenen "Blind Spots" nicht erkennen – fundamentale Fehler bleiben bestehen.

💡 **Tipp:** Für wichtige E-Mails oder Dokumente vor dem Versand – "Schreiben → SelfCritique → Finalisieren".

### \*\*98. MultiVote\*\*

Führt dieselbe Anfrage mehrfach aus und wählt die häufigste/meiste Antwort (Konsens).

☑ Stärke: Reduziert Halluzinationen und Zufälligkeiten. Gut bei Fakten-Abfragen (Namen, Daten).

⚠ Schwäche: Teuer – 3-5x Token-Verbrauch. Bei kreativen Aufgaben ("Schreibe ein Gedicht") kontraproduktiv.

💡 Tipp: Mit "vote\_count=3" und günstigem Modell starten – nur für Faktenprüfung, nicht für Kreativität.

#### **\*\*99. Validate\*\***

Prüft eine Antwort gegen externe Kriterien oder Faktenbasen.

☑ Stärke: Kann mit Memory-Tools kombiniert werden – "Prüfe gegen unsere Dokumentation".

⚠ Schwäche: Benötigt zuverlässige Validierungsquelle – bei veralteten Daten falsch positives "Valid".

💡 Tipp: Für "Entspricht das unseren Style Guidelines?" – Antwort wird gegen interne Regeln geprüft.

#### **\*\*100. CrossResolve\*\***

Vergleicht Antworten verschiedener Modelle und löst Widersprüche auf.

☑ Stärke: Kombiniert Stärken verschiedener Modelle (z.B. GPT-4 für Fakten + lokales Modell für Kontext).

⚠ Schwäche: Erfordert mindestens 2 Modelle – Setup-Aufwand höher. Konfliktlösung nicht immer eindeutig.

💡 Tipp: Bei widersprüchlichen Recherche-Ergebnissen – "Modell A sagt X, Modell B sagt Y → CrossResolve entscheidet".

#### **\*\*101. BetterPrompt\*\***

Optimiert den User-Prompt für bessere Ergebnisse (Prompt-Engineering-Assistent).

☑ Stärke: Formuliert vage Anfragen präziser um. Fügt Kontext und Beispiele hinzu.

⚠ Schwäche: Kann den ursprünglichen Intent verfälschen, wenn der Prompt zu komplex ist.

💡 Tipp: Für Einsteiger ideal – "Ich will was mit Code" wird zu "Schreibe eine Python-Funktion mit..."

#### **\*\*102. GetResult\*\***

Holt Ergebnisse asynchroner IQ-Operationen ab (Polling für lange Operationen).

☑ Stärke: Nicht-blockierend – Client kann andere Aufgaben erledigen während IQTool arbeitet.

⚠ Schwäche: Erfordert Job-ID-Management – Client muss sich merken, welche Anfrage zu welchem Ergebnis gehört.

💡 Tipp: Immer mit Timeout arbeiten – "Warte max 60 Sekunden, dann Abbrechen".

#### **\*\*103. GetModel\*\***

Zeigt Informationen über aktuell verwendete Modelle (Name, Version, Kontext-Fenster).

☑ Stärke: Transparenz – User sieht, WELCHE KI antwortet. Wichtig für Debugging.

⚠ Schwäche: Zeigt nur konfigurierte Modelle – keine Live-Verfügbarkeitsprüfung (Server offline = Fehler erst bei Request).

💡 Tipp: Beim Start der Session aufrufen – "Verbunden mit: LM Studio (Llama-3-8B)" anzeigen.

#### **\*\*104. SelectModel\*\***

Wählt das beste Modell für eine bestimmte Aufgabe (Routing).

☑ Stärke: Spart Kosten – simple Aufgaben an lokales Modell, komplexe an Cloud.

⚠ Schwäche: Heuristik-basiert – Auswahl nicht immer optimal (Edge Cases).

💡 Tipp: Mit Kategorien arbeiten – "Code-Task → lokales Modell, Kreativ-Task → GPT-4".

#### **\*\*105. Ensemble\*\***

Kombiniert mehrere Modelle zu einem "Komitee" – gewichtetes Voting.

☑ Stärke: Höchste Qualität bei kritischen Entscheidungen. Redundanz bei Modell-Ausfällen.

⚠ Schwäche: Sehr teuer – 3-5 Modelle parallel laufen lassen. Latenz höher.

💡 Tipp: Nur für finale Entscheidungen – "Soll ich diesen Vertrag unterschreiben?" → Ensemble gibt Sicherheit.

#### **\*\*106. Debate\*\***

Zwei Modelle diskutieren pro/contra eine These (adversariales Reasoning).

☑ Stärke: Enthüllt Schwächen in Argumentation. Gut für komplexe ethische/professionelle Dilemmas.

⚠ Schwäche: Kann in Endlosschleifen geraten ("Doch!" "Nein!") – Iterationslimit zwingend nötig.

💡 Tipp: Für Pro-Contra-Listen nutzen – "Soll ich Microservices oder Monolith wählen?" → beide Seiten beleuchten.

#### **\*\*107. Iterate\*\***

Verbessert eine Antwort iterativ über mehrere Durchläufe.

☑ Stärke: Jede Iteration poliert das Ergebnis weiter – "schlecht → okay → gut → exzellent".

⚠ Schwäche: Abnehmender Grenznutzen – nach 3-4 Iterationen kaum noch Verbesserung, aber steigende Kosten.

💡 Tipp: Mit "max\_iterations=3" begrenzen – dann menschliches Review, nicht endlose Iteration.

#### **\*\*108. TreeOfThought\*\***

Exploriert multiple Lösungswege parallel (breitensuche im Gedankenraum).

☑ Stärke: Findet kreative Lösungen, die direkte Antworten übersehen. Gut für Rätsel/Planung.

⚠ Schwäche: Exponentielle Komplexität – bei vielen Ästen schnell unübersichtlich und teuer.

💡 Tipp: Mit "branching\_factor=2" starten (nur 2 Optionen pro Ebene) – nicht zu breit werden lassen.



### **\*\*109. ExpertPanel\*\***

Simuliert verschiedene Expertenrollen (CTO, Anwalt, Marketing).

☑ **Stärke:** Mehrperspektivische Betrachtung – technisch, rechtlich, wirtschaftlich gleichzeitig.

⚠ **Schwäche:** "Experten" sind immer noch dasselbe Modell – keine echte Spezialisierung, nur Rolle.

💡 **Tipp:** Für Team-Entscheidungen – "Was sagt der Security-Experte dazu?" simulieren.

### **\*\*110. RedTeam\*\***

Lässt ein Modell die Antwort eines anderen angreifen (sicherheitskritische Prüfung).

☑ **Stärke:** Findet Schwachstellen, Sicherheitslücken und logische Fehler. Wichtig für Code-Reviews.

⚠ **Schwäche:** Kann überkritisch werden – nicht jeder gefundene "Fehler" ist wirklich relevant.

💡 **Tipp:** Für Security-Code – "Schreibe Auth-Modul → RedTeam versucht es zu knacken → Fixe Schwachstellen".

### **\*\*111. MetaReason\*\***

Analysiert DEN reasoning-Prozess selbst ("Wie bin ich zu dieser Schlussfolgerung gekommen?").

☑ **Stärke:** Höchste Meta-Ebene – erkennt systematische Denkfehler im Modell.

⚠ **Schwäche:** Sehr abstrakt, schwer zu kontrollieren. Kann in philosophische Abschweifungen geraten.

💡 **Tipp:** Für Debugging von IQ-Tools selbst – "Warum hat MultiVote hier falsch entschieden?"

### **\*\*112. Socratic\*\***

Führt durch Fragen zum eigenen Verständnis (maieutische Methode).

☑ **Stärke:** Lernfördernd – User kommt selbst zur Erkenntnis statt Antwort zu bekommen.

⚠ **Schwäche:** Frustrierend, wenn User direkte Antwort will ("Stell keine Gegenfragen, sag es mir!").

💡 Tipp: Für Coaching/Education – "Erkläre mir OOP" → durch Fragen zum Aha-Effekt führen.

### **\*\*113. Hybrid\*\***

Kombiniert mehrere IQ-Strategien dynamisch basierend auf Aufgabentyp.

☒ Stärke: "Best of all worlds" – automatische Auswahl zwischen ChainOfThought, SelfCritique, etc.

⚠️ Schwäche: Black-Box – nicht transparent, warum gerade DIESER Mix gewählt wurde.

💡 Tipp: Als Standard-Einstellung für unerfahrene User – "Ich weiß nicht welches IQTool → Hybrid entscheidet".

# Zusammenfassung der Tool-Anzahl:

---

- FileTools: 15
- DataTools: 19
- MemoryTools: 12
- MathTools: 9
- WebTools: 5
- SystemTools: 20
- CUTools: 5
- ClipboardTools: 4
- TimerTools: 4
- TextTools: 2
- IQTools: 18

**\*\*Gesamt: 113 Tools\*\*** (ohne TaskTools, die nur in der SLL-Version verfügbar sind)

## 8) Checkliste vor Verteilung

- Server startet ohne Fehlermeldung.
- Config-Editor lädt/speichert korrekt.
- Wichtige Tools (Echo, ReadFile, Screenshot, IQ Prompt) erfolgreich getestet.
- Optional: OpenAI-Key gesetzt und Test-Request erfolgreich.
- Dokumentation + FAQ aktuell.
- 

MCP Config Editor - mcp\_config.json

SerperAPIKey

OpenAIAPIKey

☒ local\_only

☐ OpenAI image tool enabled

OpenAI imageModel

log\_level  port  session\_timeout\_sec  IQTimeout  tool\_registration\_mode

tool\_registration\_categories (CSV: Category=0/1/2, ...)

ClipboardTools=2, CUTools=2, DataTools=2, FileTools=2, IQTools=2, MathTools=2, MemoryTools=2, SystemTools=2, TaskTools=0, TextTools=2, TimerTools=2

featureFlags (+4 = essential). DoubleClick/Space toggles 0/1/2; use Toggle Essential for +4.

State	Tool	Group	Essential	Known	Description
<input checked="" type="checkbox"/>	Enabled	apply_diff_edit	No	Yes	Apply a diff to text (edit/patch helper).
<input checked="" type="checkbox"/>	Enabled	spr_server_assistant	No	Yes	SPR server assistant (internal helper).
<input checked="" type="checkbox"/>	Disabled	task_check_result	No	Yes	Internal task result checker (legacy).
<input checked="" type="checkbox"/>	Enabled	ClipboardTools::Read	Yes	Yes	Read text from the Windows clipboard.
<input checked="" type="checkbox"/>	AskUser	ClipboardTools::Write	No	Yes	Write text to the Windows clipboard.
<input checked="" type="checkbox"/>	Enabled	CUTools::ListWindows	Yes	Yes	List visible windows with title/class/...
<input checked="" type="checkbox"/>	Enabled	CUTools::LocateAppWindow	Yes	Yes	Locate and set current top window.
<input checked="" type="checkbox"/>	Enabled	CUTools::LocateChildWindow	Yes	Yes	Locate a child window/control by te...
<input checked="" type="checkbox"/>	Enabled	CUTools::LocateGUIElement	Yes	Yes	Use iterative vision grid refinement t...
<input checked="" type="checkbox"/>	Enabled	CUTools::GetWindowState	Yes	Yes	Read full state details for a target wi...
<input checked="" type="checkbox"/>	Disabled	CUTools::GenerateImage	No	Yes	Generate images via OpenAI (enabl...
<input checked="" type="checkbox"/>	Enabled	CUTools::MouseDown	No	Yes	Perform real mouse click actions (left...
<input checked="" type="checkbox"/>	Enabled	CUTools::PictureConvert	Yes	Yes	Convert image files/memory/clipboar...
<input checked="" type="checkbox"/>	Disabled	CUTools::PictureEdit	No	Yes	Edit existing images with OpenAI pro...
<input checked="" type="checkbox"/>	Enabled	CUTools::PictureResize	Yes	Yes	Resize image files/memory/clipboar...
<input checked="" type="checkbox"/>	Enabled	CUTools::Screenshot	Yes	Yes	Capture screenshots from desktop o...
<input checked="" type="checkbox"/>	Enabled	CUTools::SetPosAndSize	Yes	Yes	Move and/or resize the currently loc...
<input checked="" type="checkbox"/>	Enabled	CUTools::SetWindowState	Yes	Yes	Set visible/enabled/iconic/maximize...
<input checked="" type="checkbox"/>	Enabled	CUTools::SimKeyPress	No	Yes	Simulate typing/keypress sequence...
<input checked="" type="checkbox"/>	Enabled	DataTools::CsvToJson	No	Yes	Convert CSV text to JSON.
<input checked="" type="checkbox"/>	Enabled	DataTools::FindText	Yes	Yes	Find text in an input (supports option...
<input checked="" type="checkbox"/>	Enabled	DataTools::GetInfo	Yes	Yes	Get basic info about the provided te...
<input checked="" type="checkbox"/>	Enabled	DataTools::GetLines	Yes	Yes	Split text into lines and return them a...
<input checked="" type="checkbox"/>	Enabled	DataTools::JsonSelect	No	Yes	Select/extract parts of JSON using ...
<input checked="" type="checkbox"/>	Enabled	DataTools::JsonToCsv	No	Yes	Convert JSON to CSV.
<input checked="" type="checkbox"/>	Enabled	DataTools::ParseKey/Value	No	Yes	Parse key=value text into structured ...
<input checked="" type="checkbox"/>	Enabled	DataTools::RegexReplace	No	Yes	Regex search/replace on text.
<input checked="" type="checkbox"/>	Enabled	DataTools::RegexSearch	No	Yes	Regex search on text (returns match...
<input checked="" type="checkbox"/>	Enabled	DataTools::Replace	Yes	Yes	Plain text replace (fast non-regex rep...
<input checked="" type="checkbox"/>	Enabled	DataTools::SortList	No	Yes	Sort a list of strings (and optionally u...
<input checked="" type="checkbox"/>	Enabled	DataTools::Split	Yes	Yes	Split text by delimiter into a list.
<input checked="" type="checkbox"/>	Enabled	DataTools::TableToJson	No	Yes	Convert a delimited table to JSON o...
<input checked="" type="checkbox"/>	AskUser	FileTools::AppendToFile	No	Yes	Append text to a file (confirm if need...
<input checked="" type="checkbox"/>	AskUser	FileTools::CreateDirectory	No	Yes	Create a directory (confirm if needed).
<input checked="" type="checkbox"/>	AskUser	FileTools::Delete	No	Yes	Delete a file or directory (confirm if n...
<input checked="" type="checkbox"/>	Enabled	FileTools::FindText	Yes	Yes	Search for text inside a file.

Loaded: C:\Users\theog\Desktop\workspace\workfolder\HTTP\_JSON\_Lib\MCP-Config\mcp\_config.json | Disabled=14 Enabled=92 AskUser=21 Essential=46

Toggle

Toggle Ess

All 0

All 1

All 2

Open...

Save As...

Reverse

Pattern...

Remember

Restore

Add

Reset

Reload

SAVE

Close

## 9) Referenzkarten der Tool-Kategorien

Diese Referenzkarten zeigen je Kategorie: wofür sie gedacht ist, was sie macht und welche Tools enthalten sind.

### ClipboardTools

Wofür gut: Zwischenablage lesen/schreiben/löschen und Formate prüfen.

Umfang: 2 aktiv / 2 definiert (laut aktueller mcp\_config.json).

Tools: ClipboardTools::Read, ClipboardTools::Write

### CUTools

Wofür gut: Computer-Use: Fenster, Screenshots, Maus/Tastatur, Vision-Lokalisierung, Bildfunktionen.

Umfang: 14 aktiv / 14 definiert (laut aktueller mcp\_config.json).

Tools: CUTools::GenerateImage, CUTools::GetWindowState, CUTools::ListWindows, CUTools::LocateAppWindow, CUTools::LocateChildWindow, CUTools::LocateGUIElement, CUTools::MouseDownClick, CUTools::PictureConvert, CUTools::PictureEdit, CUTools::PictureResize, CUTools::Screenshot, CUTools::SetPosAndSize, CUTools::SetWindowState, CUTools::SimKeyPress

### DataTools

Wofür gut: Text, JSON, CSV und Struktur-Umwandlung für Datenfluss in Prompts und Automationen.

Umfang: 13 aktiv / 13 definiert (laut aktueller mcp\_config.json).

Tools: DataTools::CsvToJson, DataTools::FindText, DataTools::GetInfo, DataTools::GetLines, DataTools::JsonSelect, DataTools::JsonToCsv, DataTools::ParseKeyValue, DataTools::RegexReplace, DataTools::RegexSearch, DataTools::Replace, DataTools::SortList, DataTools::Split, DataTools::TableToJson

### FileTools

Wofür gut: Dateien und Ordner lesen, schreiben, suchen, kopieren, verschieben und verwalten.

Umfang: 13 aktiv / 13 definiert (laut aktueller mcp\_config.json).

Tools: FileTools::AppendToFile, FileTools::CreateDirectory, FileTools::Delete, FileTools::FindText, FileTools::GetDirectoryListing, FileTools::GetInfo, FileTools::GetLines, FileTools::List, FileTools::Move, FileTools::ReadFile, FileTools::Unzip, FileTools::WriteFile, FileTools::Zip

## IQTools

Wofür gut: Mehrstufiges KI-Reasoning (z. B. Chain, Debate, MultiVote, Validate).

Umfang: 18 aktiv / 18 definiert (laut aktueller mcp\_config.json).

Tools: IQTools::BetterPrompt, IQTools::ChainOfThought, IQTools::CrossResolve, IQTools::Debate, IQTools::Ensemble, IQTools::ExpertPanel, IQTools::GetModel, IQTools::GetResult, IQTools::Hybrid, IQTools::Iterate, IQTools::MetaReason, IQTools::MultiVote, IQTools::RedTeam, IQTools::SelectModel, IQTools::SelfCritique, IQTools::Socratic, IQTools::TreeOfThought, IQTools::Validate

## MathTools

Wofür gut: Formeln, Sequenzen, Optimierung und Berechnungsschritte für präzise Antworten.

Umfang: 9 aktiv / 9 definiert (laut aktueller mcp\_config.json).

Tools: MathTools::Integrate, MathTools::LoopCalc, MathTools::LoopStart, MathTools::LoopStep, MathTools::MonteCarlo, MathTools::Optimizer, MathTools::ResolveFormula, MathTools::RootFinder, MathTools::SmartSequence

## MemoryTools

Wofür gut: Persistente/temporäre Wissensspeicher für Workflows, Kontext und Zwischenstände.

Umfang: 12 aktiv / 12 definiert (laut aktueller mcp\_config.json).

Tools: MemoryTools::Delete, MemoryTools::Format, MemoryTools::FromClipboard, MemoryTools::FromFile, MemoryTools::GetLines, MemoryTools::JSON\_Escape, MemoryTools::JSON\_Unescape, MemoryTools::ListKeys, MemoryTools::Recall, MemoryTools::Store, MemoryTools::ToClipboard, MemoryTools::ToFile

## SystemTools

Wofür gut: Systemnahe Aufgaben wie Prozesse, Dienste, Registry und Kommandos (kontrolliert).

Umfang: 20 aktiv / 20 definiert (laut aktueller mcp\_config.json).

Tools: SystemTools::CreateRegistryKey, SystemTools::DeleteRegistryKey, SystemTools::DeleteRegistryValue, SystemTools::ExecuteCommand, SystemTools::GetCommandResult, SystemTools::GetEnvironmentVariable, SystemTools::GetInstallPath, SystemTools::GetServiceDetails, SystemTools::IdentifyConhostParent, SystemTools::IdentifySvchostService, SystemTools::ListProcesses, SystemTools::ListRegistrySubKeys, SystemTools::ListRegistryValues, SystemTools::ListServices, SystemTools::RestartService, SystemTools::RunBatchFile, SystemTools::RunPowerShell, SystemTools::StartService, SystemTools::StopService, SystemTools::WriteRegistryValue

## TaskTools

Wofür gut: SLL-Betrieb: Code-/Script-Ausführung synchron/asynchron in Robot-Umgebungen.

Umfang: 0 aktiv / 5 definiert (laut aktueller mcp\_config.json).

Tools: TaskTools::CheckResult, TaskTools::RunCodeAsync, TaskTools::RunCodeSync, TaskTools::RunScriptAsync, TaskTools::RunScriptSync

## TextTools

Wofür gut: Diff/Unified-Diff anwenden für Patch- und Textkorrektur-Workflows.

Umfang: 1 aktiv / 2 definiert (laut aktueller mcp\_config.json).

Tools: TextTools::ApplyDiff, TextTools::ApplyUnifiedDiff

## TimerTools

Wofür gut: Zeitgesteuerte Jobs, wiederkehrende Aufgaben und sofortige Ausführung mit Parametern.

Umfang: 6 aktiv / 6 definiert (laut aktueller mcp\_config.json).

Tools: TimerTools::ClearTimer, TimerTools::FindTimer, TimerTools::GetTimer, TimerTools::ListTimers, TimerTools::SetTimer, TimerTools::UpdateTimer

## UserTools

Wofür gut: SLL-Betrieb: benutzerdefinierte Rezepte und dynamisch geladene Tool-Flows.

Umfang: 0 aktiv / 1 definiert (laut aktueller mcp\_config.json).

Tools: UserTools::\*

## WebTools

Wofür gut: Web-Inhalte abrufen, Header prüfen und Such-/Fetch-Aufgaben automatisieren.

Umfang: 5 aktiv / 5 definiert (laut aktueller mcp\_config.json).

Tools: WebTools::ExtractText, WebTools::FetchHeaders, WebTools::FetchMany,  
WebTools::FetchText, WebTools::Search

Hinweis: In der Verteilung kann die sichtbare Tool-Liste je nach Registration-Mode und Feature-Flags variieren.



## 4) FAQ und häufige Fehler

Problem	Wahrscheinliche Ursache	Lösung
Tool nicht gefunden	LM Studio cached alte Tool-Liste	MCP-Server in LM Studio neu laden / LM Studio neu starten.
OpenAI-Aufruf schlägt fehl	API-Key fehlt/falsch oder Netzwerkproblem	Key prüfen, <code>`https://api.openai.com`</code> erreichbar machen, Uhrzeit/System prüfen.
Transcribe stoppt nicht	Audio-Device blockiert oder Rechte fehlen	Windows Mikrophon-Rechte prüfen, andere Recorder schließen, erneut starten.
Build-Fehler trotz scheinbar korrektem Code	CRLF/LF-Mix in INC-Datei	Datei auf CRLF normalisieren und neu kompilieren.
GUI-Element wurde falsch angeklickt	Skalierung/Fensterrahmen nicht berücksichtigt	Locate-Funktionen + Vision-Refine-Flow verwenden.
Tests hängen gelegentlich	Warte-Loop ohne Timeout	Watchdog-Timeout setzen, Prozess killen, Debug-Marker auswerten.

## 5) Wo kann ich SindByte verbinden?

- LM Studio (lokale Modelle)
- OpenAI-kompatible Flows (für IQ-Reasoning/Transcribe/Bilder)
- Codex / VS Code-Umgebungen mit MCP-Anbindung
- Roo Code und andere MCP-fähige Clients
- Eigene Skripte per HTTP-POST auf `/mcp`

## 6) Online-Bedienungsanleitung (Kurzform)

6. Schritt 1: Config öffnen und nur notwendige Tool-Kategorien aktivieren.
7. Schritt 2: Server starten und Status in GUI prüfen.
8. Schritt 3: Verbindung im Client testen (einfaches Tool zuerst).
9. Schritt 4: IQTools/Timer für regelmäßige Aufgaben nutzen.
10. Schritt 5: Ergebnisse prüfen, dann Workflows erweitern (z. B. Bilder, Automatisierung).

## 7) Rechtliches

**Copyright** (c) Theo Gottwald, 76706 Dettenheim / Germany

**Kontakt:** [info@it-berater.org](mailto:info@it-berater.org) | <http://smart-ai-robot.com/>

**Nutzungsrecht:** Kostenfrei für private Nutzung.

**Gewährleistung/Haftung:** Die Nutzung erfolgt auf eigenes Risiko. Es wird keine Haftung für Schäden, Datenverlust oder Folgeschäden übernommen.

**Hinweis:** Vor produktivem Einsatz bitte eigene Sicherheits- und Compliance-Prüfung durchführen.

