Trados PowerShell MCP Server - Technical Design Document

Version: 1.7.7
Date: 29 March 2026
Author: multifarious
Platform: Node.js (TypeScript), stdio transport

---

1. Overview

The Trados PowerShell MCP Server is a local Model Context Protocol server that wraps three RWS PowerShell toolkits, exposing their capabilities as MCP tools. This allows AI assistants (Claude Desktop, Claude Code, or any MCP-compatible client) to manage Trados Studio projects, interact with GroupShare servers, and automate Language Cloud workflows through natural language.

The three toolkits - and the capabilities they expose - are:

Toolkit	Target system	Tools prefix
Trados Studio PowerShell Toolkit	Local Trados Studio installation (Project Automation API)	`studio_`
GroupShare API PowerShell Toolkit	GroupShare REST API (on-premise server)	`gs_`
Language Cloud PowerShell Toolkit	RWS Language Cloud / Trados Enterprise REST API (cloud)	`lc_`

The server is intentionally thin. It translates MCP tool calls into PowerShell script fragments, executes them via the appropriate PowerShell host, and returns JSON-serialised output. All project creation, analysis, package, TM, and user management logic remains in the toolkits. The MCP server adds zero business logic.

This server is a companion to the SDLXLIFF Refiner MCP Server. The two operate at different levels:

- **SDLXLIFF Refiner MCP** - works on bilingual file content (segments, tags, translation, search/replace).
- **Trados PowerShell MCP** - works on project infrastructure (creating projects, managing TMs, producing packages, user and organisation management).

Both servers are registered independently in Claude Desktop. Neither depends on the other, but they complement each other naturally in a full project lifecycle.

1.1 Tool Group Activation

Each of the three tool groups is independently optional. At startup, the server inspects its environment variables and registers only the tool groups for which the required credentials or configuration are present:

- `studio_*` tools are registered if `STUDIO_VERSION` is set (or defaults to `Studio18`). The Studio toolkit

requires Trados Studio Professional to be installed on the local machine; there is no credential check at startup.

- `gs_*` tools are registered if `GS_CREDENTIAL_STORE` points to a folder that exists and contains at least one `.xml` file, or if `GS_SERVER_URL`, `GS_USERNAME`, and `GS_PASSWORD` are all set.
- `lc_*` tools are registered if `LC_CREDENTIAL_STORE` points to a folder that exists and contains at least one `.xml` file, or if `LC_CLIENT_ID`, `LC_CLIENT_SECRET`, and `LC_TENANT_ID` are all set.

This means the same server binary works for users with only a local Studio installation, only GroupShare, only Language Cloud, or any combination of the three. Claude Desktop shows only the tools that are actually usable in the current environment.

1.2 Multi-Tenant Credential Model

Users who administer multiple GroupShare servers or Language Cloud tenants store one DPAPI-encrypted credential XML per environment in a credential store folder. The `GS_CREDENTIAL_STORE` and `LC_CREDENTIAL_STORE` environment variables point to these folders.

The workflow for selecting a credential at runtime:

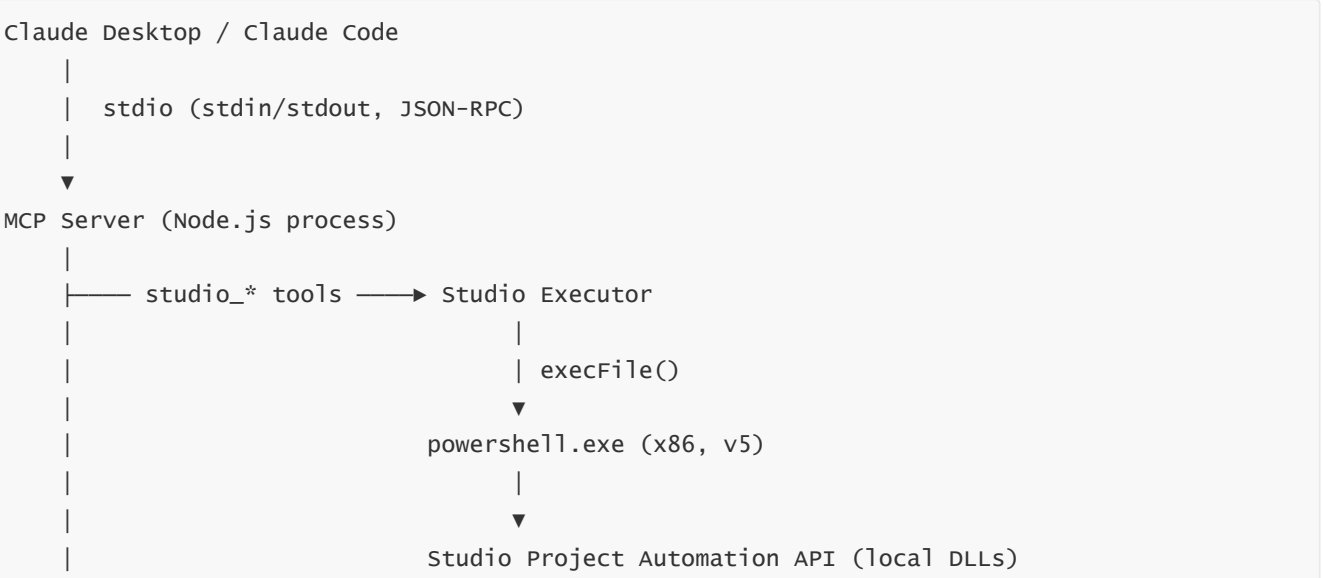
1. Call `gs_list_credentials` or `lc_list_credentials` to see all available credential files and the server URL / tenant ID each contains.
2. Call `gs_set_credential` or `lc_set_credential` with the chosen filename to activate it for the current session.
3. All subsequent GS or LC tool calls use the active credential until changed or until Claude Desktop is restarted.

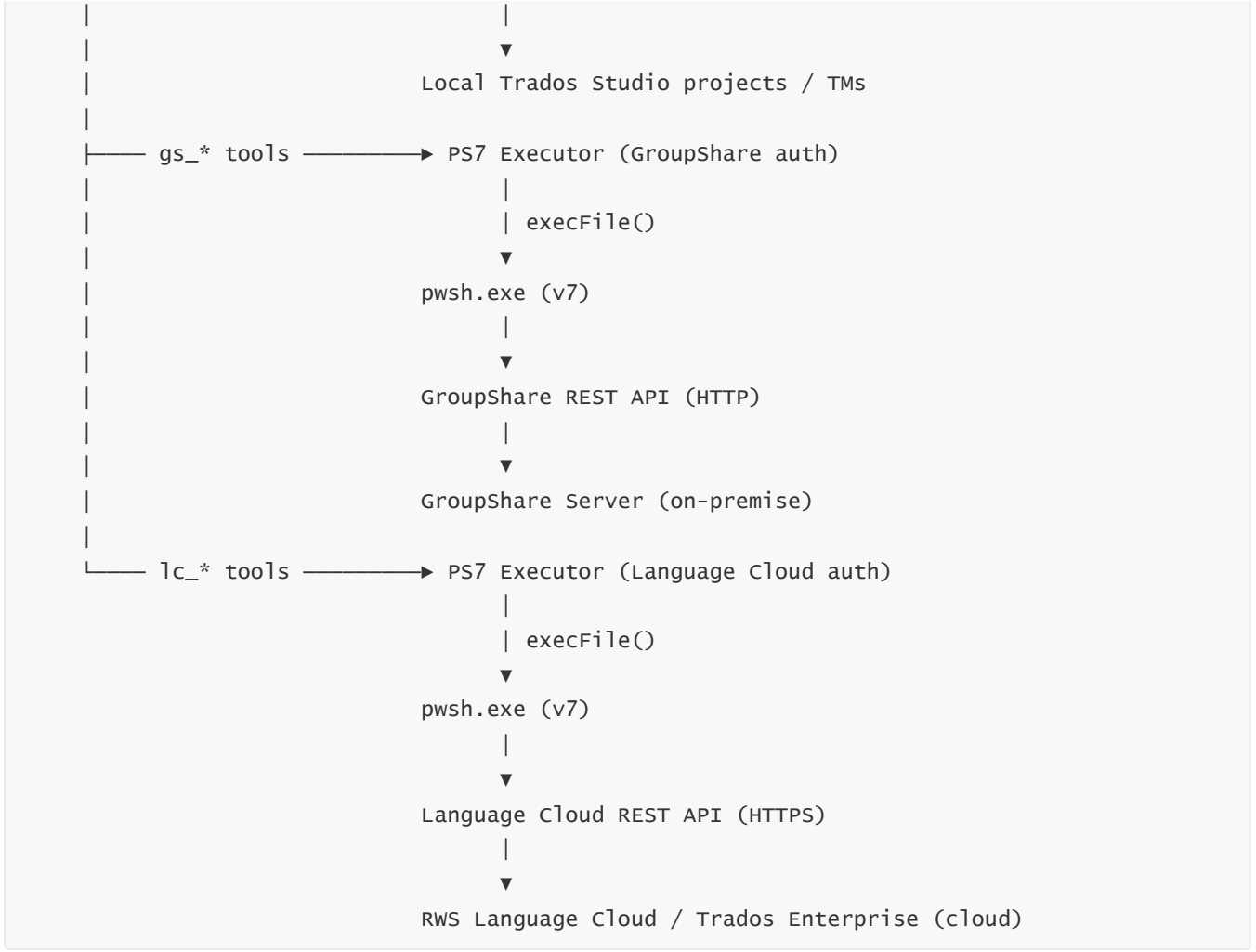
On startup, if a credential store folder contains exactly one XML file, that file is auto-selected as the active credential. If the folder contains multiple files, no credential is auto-selected and a call to `gs_set_credential` / `lc_set_credential` is required before any other GS or LC tool will work.

The active credential selection is held in process memory in `src/state.ts` and is not persisted across server restarts.

---

2. Architecture





2.1 Why Two Executors

The three toolkits have fundamentally different runtime requirements:

Studio toolkit: Requires PowerShell 5.1 x86 (`SysWOW64\WindowsPowerShell\v1.0\powershell.exe`). The toolkit loads 32-bit Trados Studio DLLs at runtime. PowerShell 7 or the 64-bit PS5 host fail with assembly load errors. The toolkit uses `Import-ToolkitModules` from `ToolkitInitializer` to load all modules and resolve Studio DLL paths. Authentication is implicit - the current Windows user's Studio licence.

GroupShare and Language Cloud toolkits: Both require PowerShell 7 (`pwsh.exe`). Both make HTTP REST calls rather than loading local DLLs. Authentication is explicit: GroupShare uses username/password passed to `SignIn` to obtain a session token; Language Cloud uses OAuth2 client credentials passed to `Get-AccessKey` to obtain a bearer token object.

Note: both toolkits have a module named `AuthenticationHelper` with entirely different implementations. Because each tool invocation spawns a fresh PowerShell process with its own module scope, there is no conflict.

The server therefore maintains two executor functions:

- `studioPs()` - invokes the PS5 x86 host, loads `ToolkitInitializer`, no explicit auth
- `ps7()` - invokes the PS7 host, loads toolkit-specific modules, handles auth token acquisition and injection

2.2 PS7 Auth Token Caching

The LC `AuthenticationHelper` module caches the access token to `accessToken.json` within the module directory. The GS `AuthenticationHelper` module makes a REST call on every `SignIn` invocation. For simplicity in v1, each tool invocation performs its own auth call. This is fast enough for interactive use and avoids cross-process state management.

---

3. Prerequisites

3.1 Always Required

- **Node.js** (LTS, v20+) on PATH.
- **Claude Desktop** (latest version) installed.

3.2 For studio_* tools

- **Trados Studio Professional** (2022 or 2024) installed on the local machine. Freelance/Starter licences do not include the Project Automation API.
- **PowerShell 5.1 x86** - present by default on Windows 10 1607+ and Windows 11.
- **Trados Studio PowerShell Toolkit** installed on the local machine. The installation path must be provided via the `Studio Toolkit Modules Path` field when installing the desktop extension.

3.3 For gs_* tools

- **GroupShare server** accessible from the machine (URL, username, password).
- **PowerShell 7.4+** (`pwsh.exe`) installed. Install via `winget install Microsoft.PowerShell`.
- **GroupShare API PowerShell Toolkit** installed on the local machine. The installation path must be provided via the `GroupShare Toolkit Modules Path` field when installing the desktop extension.

3.4 For lc_* tools

- **RWS Language Cloud / Trados Enterprise** account with an application configured (Client ID, Client Secret, Tenant ID). Obtained via Account menu → Integrations → Applications in the LC web UI.
- **PowerShell 7.4+** (`pwsh.exe`) installed.
- **Language Cloud PowerShell Toolkit** installed on the local machine. The installation path must be provided via the `Language Cloud Toolkit Modules Path` field when installing the desktop extension.

---

4. Project Structure

```
trados-powershell-mcp/
├── manifest.json
├── package.json
├── tsconfig.json
├── src/
```

```
|   ├── index.ts                        Entry point - server setup and conditional tool
registration
|   ├── state.ts                        Active credential state (module-level, in-process
only)
|   ├── executors/
|   |   ├── studio-ps.ts                PS5 x86 executor for Studio toolkit calls
|   |   ├── ps7.ts                      PS7 executor for GroupShare and Language Cloud calls
|   |   └── common.ts                   Shared utilities: psStr, psPath, extractPsError,
safeParseJson
|   ├── tools/
|   |   ├── studio/
|   |   |   ├── register.ts             Registers all studio_* tools
|   |   |   ├── list-projects.ts
|   |   |   ├── get-project.ts
|   |   |   ├── new-project.ts
|   |   |   ├── remove-project.ts
|   |   |   ├── analyze.ts
|   |   |   ├── export-package.ts
|   |   |   ├── import-package.ts
|   |   |   ├── pretranslate.ts
|   |   |   ├── list-tms.ts
|   |   |   ├── new-tm.ts
|   |   |   ├── import-tmx.ts
|   |   |   ├── export-tmx.ts
|   |   |   └── list-project-templates.ts
|   |   ├── groupshare/
|   |   |   ├── register.ts             Registers all gs_* tools
|   |   |   ├── list-credentials.ts
|   |   |   ├── set-credential.ts
|   |   |   ├── list-projects.ts
|   |   |   ├── get-project.ts
|   |   |   ├── new-project.ts
|   |   |   ├── update-project-status.ts
|   |   |   ├── export-package.ts
|   |   |   ├── import-package.ts
|   |   |   ├── get-analysis-report.ts
|   |   |   ├── save-project-files.ts
|   |   |   ├── list-containers.ts
|   |   |   ├── list-tms.ts
|   |   |   ├── new-tm.ts
|   |   |   ├── import-tmx.ts
|   |   |   ├── export-tmx.ts
|   |   |   ├── list-project-templates.ts
|   |   |   ├── list-organizations.ts
|   |   |   ├── list-users.ts
|   |   |   ├── get-background-tasks.ts
|   |   |   ├── new-user.ts
|   |   |   ├── update-user.ts
|   |   |   ├── new-role.ts
|   |   |   ├── update-role-to-user.ts
|   |   |   ├── move-organization-resources.ts
|   |   |   └── org-report.ts
|   |   └── languagecloud/
```

```
|   |       ├── register.ts             Registers all lc_* tools
|   |       ├── list-credentials.ts
|   |       ├── set-credential.ts
|   |       ├── list-projects.ts
|   |       ├── get-project.ts
|   |       ├── new-project.ts
|   |       ├── list-project-templates.ts
|   |       ├── new-project-template.ts
|   |       ├── remove-project-template.ts
|   |       ├── list-tms.ts
|   |       ├── get-tm.ts
|   |       ├── new-tm.ts
|   |       ├── remove-tm.ts
|   |       ├── import-tm.ts
|   |       ├── export-tm.ts
|   |       ├── list-locations.ts
|   |       ├── list-customers.ts
|   |       ├── new-customer.ts
|   |       ├── update-customer.ts
|   |       ├── remove-customer.ts
|   |       ├── list-workflows.ts
|   |       ├── list-translation-engines.ts
|   |       ├── list-file-type-configurations.ts
|   |       ├── list-language-processing-rules.ts
|   |       ├── list-field-templates.ts
|   |       ├── list-pricing-models.ts
|   |       ├── list-schedule-templates.ts
|   |       ├── list-supported-languages.ts
|   |       ├── list-groups.ts
|   |       ├── list-termbases.ts
|   |       ├── list-users.ts
|   |       ├── new-termbase.ts
|   |       ├── import-termbase.ts
|   |       └── export-termbase.ts
|   └── types.ts                        Shared TypeScript types
└── dist/
    └── index.js
```

4.1 Dependencies

Package	Purpose
`@modelcontextprotocol/sdk`	MCP server SDK
`zod`	Schema validation
`typescript`	Build-time only
`@types/node`	Build-time only

5. Server Entry Point

File: `src/index.ts`

Reads environment variables at startup, performs credential store checks, auto-selects a credential if exactly one file is present in the store, and conditionally registers each tool group.

```
import { McpServer } from "@modelcontextprotocol/sdk/server/mcp.js";
import { StdioServerTransport } from "@modelcontextprotocol/sdk/server/stdio.js";
import { readdirSync, existsSync } from "fs";
import { join } from "path";
import { registerStudioTools } from "./tools/studio/register.js";
import { registerGroupShareTools } from "./tools/groupshare/register.js";
import { registerLanguageCloudTools } from "./tools/languagecloud/register.js";
import { setActiveGsCredential, setActiveLcCredential } from "./state.js";

const server = new McpServer({
  name: "trados-powershell",
  version: "1.7.0",
});

// Studio: register unconditionally - toolkit will fail at runtime if not licensed
registerStudioTools(server);

// GroupShare: register if credential store has XMLs, or if raw vars are all set
const gsStore = process.env.GS_CREDENTIAL_STORE;
const gsStoreReady = gsStore && existsSync(gsStore) &&
  readdirSync(gsStore).filter(f => f.toLowerCase().endsWith(".xml")).length > 0;
const gsRawReady = process.env.GS_SERVER_URL && process.env.GS_USERNAME &&
process.env.GS_PASSWORD;

if (gsStoreReady || gsRawReady) {
  if (gsStoreReady) {
    const xmlFiles = readdirSync(gsStore!).filter(f => f.toLowerCase().endsWith(".xml"));
    if (xmlFiles.length === 1) {
      setActiveGsCredential(join(gsStore!, xmlFiles[0]));
    }
  }
  registerGroupShareTools(server);
}

// Language Cloud: register if credential store has XMLs, or if raw vars are all set
const lcStore = process.env.LC_CREDENTIAL_STORE;
const lcStoreReady = lcStore && existsSync(lcStore) &&
  readdirSync(lcStore).filter(f => f.toLowerCase().endsWith(".xml")).length > 0;
const lcRawReady = process.env.LC_CLIENT_ID && process.env.LC_CLIENT_SECRET &&
process.env.LC_TENANT_ID;

if (lcStoreReady || lcRawReady) {
  if (lcStoreReady) {
    const xmlFiles = readdirSync(lcStore!).filter(f => f.toLowerCase().endsWith(".xml"));
    if (xmlFiles.length === 1) {
      setActiveLcCredential(join(lcStore!, xmlFiles[0]));
```

```
    }
  }
  registerLanguageCloudTools(server);
}

const transport = new StdioServerTransport();
await server.connect(transport);
```

---

6. Executors

6.1 Studio Executor

File: `src/executors/studio-ps.ts`

Invokes the PS5 x86 host, loads `ToolkitInitializer`, and wraps the script body in error handling. The optional `bare` flag skips module loading entirely for filesystem-only scripts (e.g. `studio_list_projects`).

Encoding fix: The script is written to a temporary `.ps1` file with a UTF-8 BOM rather than passed via `-Command`. PowerShell 5.1 requires the BOM to read UTF-8 correctly, and `-Command` goes through Windows argument parsing which mangles non-ASCII characters on systems with an ANSI codepage. The executor uses `-ExecutionPolicy Bypass -File` to run the temp script, includes `[Console]::OutputEncoding = [System.Text.Encoding]::UTF8` inside the script body so stdout comes back clean, and cleans up the temp file in a `finally` block.

```
import { execFile } from "child_process";
import { promisify } from "util";
import { writeFile, unlink } from "fs/promises";
import { join } from "path";
import { tmpdir } from "os";
import { randomUUID } from "crypto";
import { extractPsError, safeParseJson } from "./common.js";

const execFileAsync = promisify(execFile);

const PS5_PATH =
  process.env.STUDIO_PS_PATH ??
  "C:\\Windows\\SysWOW64\\WindowsPowerShell\\v1.0\\powershell.exe";

const STUDIO_VERSION = process.env.STUDIO_VERSION ?? "Studio18";
const STUDIO_MODULES_PATH = process.env.STUDIO_MODULES_PATH ?? "";

interface StudioPsOptions {
  bare?: boolean; // skip Import-Module / Import-ToolkitModules (filesystem-only scripts)
}

export async function studioPs(scriptBody: string, options: StudioPsOptions = {}):
Promise<object> {
  const modulePathBlock = STUDIO_MODULES_PATH
    ? `$env:PSModulePath = "${STUDIO_MODULES_PATH.replace(/\\/g, "\\\\")};" +
$env:PSModulePath`
    : "";
```

```
  const toolkitPreamble = options.bare
    ? ""
    : `Import-Module -Name ToolkitInitializer -ErrorAction Stop
      Import-ToolkitModules -StudioVersion "${STUDIO_VERSION}"`;

  const script = `
    [Console]::OutputEncoding = [System.Text.Encoding]::UTF8
    Set-StrictMode -Off
    $ErrorActionPreference = "Stop"
    try {
      ${modulePathBlock}
      ${toolkitPreamble}
      ${scriptBody}
    } catch {
      $err = @{ error = $_.Exception.Message; detail = $_.ScriptStackTrace }
      Write-Error ($err | ConvertTo-Json -Compress)
      exit 1
    }
  `;

  const tempScript = join(tmpdir(), `trados-mcp-${randomUUID()}.ps1`);
  try {
    await writeFile(tempScript, "\uFEFF" + script, "utf-8");

    const { stdout, stderr } = await execFileAsync(
      PS5_PATH,
      ["-NonInteractive", "-NoProfile", "-ExecutionPolicy", "Bypass", "-File", tempScript],
      { timeout: 600000, maxBuffer: 50 * 1024 * 1024, windowsHide: true }
    );

    if (stderr?.trim()) throw new Error(extractPsError(stderr));
    return safeParseJson(stdout);
  } finally {
    try { await unlink(tempScript); } catch { /* ignore cleanup errors */ }
  }
}
```

6.2 PS7 Executor

File: `src/executors/ps7.ts`

Invokes the PS7 host. Accepts a `toolkitType` of `"groupshare"` or `"languagecloud"` and injects the appropriate module imports and authentication preamble.

The optional `bare` flag skips all module loading and authentication. This is used by `gs_list_credentials`, `gs_set_credential`, `lc_list_credentials`, and `lc_set_credential`, which operate on local credential files and do not connect to any server.

The GroupShare preamble loads six modules plus `SystemConfigurationHelper` (needed for container and DB server lookups). All GS toolkit functions take an `$authorizationToken` string. The preamble exposes `$authToken`.

The Language Cloud preamble loads five modules. All LC toolkit functions take an `$accessKey` PSObject (containing `.token` and `.tenant` properties). The preamble exposes `$accessKey`.

Locale fix: The `Get-AccessKey` call is wrapped in a temporary `InvariantCulture` override. The LC toolkit's `AuthenticationHelper` parses date strings from the OAuth token response using US date format assumptions. On non-US locale systems (e.g. German `de-DE`), this parse fails silently or throws. The culture override ensures correct parsing regardless of the host system's locale, and is restored in a `finally` block.

Both preambles read the active credential file path from the `state` module (set by `gs_set_credential` / `lc_set_credential` or auto-selected at startup). The raw environment variable fallback is used only when no credential store is configured.

Module name collision fix: Both toolkits contain modules with identical names (`AuthenticationHelper`, `ResourcesHelper`). To guarantee the correct version is loaded, each preamble first prepends the toolkit-specific modules folder to `$env:PSModulePath` within the generated script (using `GS_MODULES_PATH` or `LC_MODULES_PATH` respectively), before any `Import-Module` call. This is a no-op if the relevant env var is not set.

```
import { execFile } from "child_process";
import { promisify } from "util";
import { psStr, psPath, extractPsError, safeParseJson } from "./common.js";
import { activeGsCredentialFile, activeLcCredentialFile } from "../state.js";

const execFileAsync = promisify(execFile);

const PS7_PATH = process.env.PS7_PATH || "pwsh.exe";

type ToolkitType = "groupshare" | "languagecloud";

interface Ps7Options {
  bare?: boolean; // skip module loading and auth preamble entirely
}

function buildPreamble(type: ToolkitType): string {
  if (type === "groupshare") {
    const modulesPath = process.env.GS_MODULES_PATH;
    const modulePathBlock = modulesPath
      ? `$env:PSModulePath = ${psStr(modulesPath)} + [System.IO.Path]::PathSeparator +
$env:PSModulePath`
      : "";

    const credFile = activeGsCredentialFile;
    let authBlock: string;
    if (credFile) {
      authBlock = `
        $credData  = Import-CliXml -Path ${psPath(credFile)}
        $serverUrl = $credData.ServerUrl
        $gsUser    = $credData.Credential.UserName
        $gsPass    = $credData.Credential.GetNetworkCredential().Password
      `;
    } else {
      const server = process.env.GS_SERVER_URL!;
```

```
      const user   = process.env.GS_USERNAME!;
      const pass   = process.env.GS_PASSWORD!;
      authBlock = `
        $serverUrl = ${psStr(server)}
        $gsUser    = ${psStr(user)}
        $gsPass    = ${psStr(pass)}
      `;
    }
    return `
      ${modulePathBlock}
      ${authBlock}
      Import-Module -Name AuthenticationHelper        -ArgumentList $serverUrl -ErrorAction
Stop
      Import-Module -Name ProjectServerHelper         -ArgumentList $serverUrl -ErrorAction
Stop
      Import-Module -Name ResourcesHelper             -ArgumentList $serverUrl -ErrorAction
Stop
      Import-Module -Name UserManagerHelper           -ArgumentList $serverUrl -ErrorAction
Stop
      Import-Module -Name BackgroundTaskHelper        -ArgumentList $serverUrl -ErrorAction
Stop
      Import-Module -Name SystemConfigurationHelper   -ArgumentList $serverUrl -ErrorAction
Stop
      $authToken = SignIn -userName $gsUser -password $gsPass
    `;
  } else {
    const modulesPath = process.env.LC_MODULES_PATH;
    const modulePathBlock = modulesPath
      ? `$env:PSModulePath = ${psStr(modulesPath)} + [System.IO.Path]::PathSeparator +
$env:PSModulePath`
      : "";

    const credFile = activeLcCredentialFile;
    let authBlock: string;
    if (credFile) {
      authBlock = `
        $credData     = Import-CliXml -Path ${psPath(credFile)}
        $ptr          =
[System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($credData.lcTenant)
        $lcTenant     = [System.Runtime.InteropServices.Marshal]::PtrToStringBSTR($ptr)
        [System.Runtime.InteropServices.Marshal]::ZeroFreeBSTR($ptr)
        $ptr          =
[System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($credData.clientId)
        $clientId     = [System.Runtime.InteropServices.Marshal]::PtrToStringBSTR($ptr)
        [System.Runtime.InteropServices.Marshal]::ZeroFreeBSTR($ptr)
        $ptr          =
[System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($credData.clientSecret)
        $clientSecret = [System.Runtime.InteropServices.Marshal]::PtrToStringBSTR($ptr)
        [System.Runtime.InteropServices.Marshal]::ZeroFreeBSTR($ptr)
      `;
    } else {
      const clientId     = process.env.LC_CLIENT_ID!;
      const clientSecret = process.env.LC_CLIENT_SECRET!;
```

```
      const tenant       = process.env.LC_TENANT_ID!;
      authBlock = `
        $clientId     = ${psStr(clientId)}
        $clientSecret = ${psStr(clientSecret)}
        $lcTenant     = ${psStr(tenant)}
      `;
    }
    return `
      ${modulePathBlock}
      ${authBlock}
      Import-Module -Name AuthenticationHelper -ErrorAction Stop
      Import-Module -Name ProjectHelper        -ErrorAction Stop
      Import-Module -Name ResourcesHelper      -ErrorAction Stop
      Import-Module -Name UsersHelper          -ErrorAction Stop
      Import-Module -Name TerminologyHelper    -ErrorAction Stop
      $prevCulture = [System.Threading.Thread]::CurrentThread.CurrentCulture
      [System.Threading.Thread]::CurrentThread.CurrentCulture =
[System.Globalization.CultureInfo]::InvariantCulture
      try {
        $accessKey = Get-AccessKey -id $clientId -secret $clientSecret -lcTenant $lcTenant
      } finally {
        [System.Threading.Thread]::CurrentThread.CurrentCulture = $prevCulture
      }
    `;
  }
}

export async function ps7(type: ToolkitType, scriptBody: string, options: Ps7Options = {}):
Promise<object> {
  const preamble = options.bare ? "" : buildPreamble(type);

  const script = `
    Set-StrictMode -Off
    $ErrorActionPreference = "Stop"
    try {
      ${preamble}
      ${scriptBody}
    } catch {
      $err = @{ error = $_.Exception.Message; detail = $_.ScriptStackTrace }
      Write-Error ($err | ConvertTo-Json -Compress)
      exit 1
    }
  `;

  const { stdout, stderr } = await execFileAsync(
    PS7_PATH,
    ["-NonInteractive", "-NoProfile", "-Command", script],
    { timeout: 600000, maxBuffer: 50 * 1024 * 1024, windowsHide: true }
  );

  if (stderr?.trim()) throw new Error(extractPsError(stderr));
  return safeParseJson(stdout);
}
```

6.3 State Module

File: `src/state.ts`

Holds the active credential file path for each tool group. Set by `gs_set_credential` / `lc_set_credential` at runtime, or auto-initialised at startup when the credential store contains exactly one XML. Not persisted across server restarts.

```
export let activeGsCredentialFile: string | null = null;
export let activeLcCredentialFile: string | null = null;

export function setActiveGsCredential(filePath: string): void {
  activeGsCredentialFile = filePath;
}

export function setActiveLcCredential(filePath: string): void {
  activeLcCredentialFile = filePath;
}
```

6.4 Common Utilities

File: `src/executors/common.ts`

```
// Escape a string for single-quoted PowerShell interpolation
export function psStr(value: string): string {
  return `'${value.replace(/'/g, "''")}'`;
}

// Normalise path separators and escape for PS
export function psPath(value: string): string {
  return psStr(value.replace(/\//g, "\\"));
}

// Extract the error message from PS stderr JSON or raw text
export function extractPsError(stderr: string): string {
  try {
    const parsed = JSON.parse(stderr.trim());
    return parsed.error || stderr.trim();
  } catch {
    return stderr.trim();
  }
}

/**
 * Safely parse JSON from PowerShell stdout.
 *
 * Some toolkit functions write warnings or error messages to stdout via
 * Write-Host / Write-Warning before the JSON output. This function:
 *  1. Tries JSON.parse on the full string (fast path).
 *  2. If that fails, scans for the last top-level JSON object ({...})
 *     and tries to parse that.
 *  3. If that also fails, throws with the raw stdout text so the caller
```

```
 *     can surface it as a readable error.
 *
 * If prefix text was found before the JSON, it is attached as a `_warnings`
 * property on the returned object.
 */
export function safeParseJson(stdout: string): object {
  const trimmed = stdout.trim();
  if (!trimmed) return {};

  try {
    return JSON.parse(trimmed) as object;
  } catch {
    // Fall through to extraction.
  }

  const lastBrace = trimmed.lastIndexOf("}");
  if (lastBrace === -1) throw new Error(trimmed);

  let depth = 0;
  let start = -1;
  for (let i = lastBrace; i >= 0; i--) {
    if (trimmed[i] === "}") depth++;
    if (trimmed[i] === "{") depth--;
    if (depth === 0) { start = i; break; }
  }

  if (start === -1) throw new Error(trimmed);

  const jsonCandidate = trimmed.substring(start, lastBrace + 1);
  try {
    const parsed = JSON.parse(jsonCandidate) as object;
    const prefix = trimmed.substring(0, start).trim();
    if (prefix) {
      (parsed as Record<string, unknown>)._warnings = prefix;
    }
    return parsed;
  } catch {
    throw new Error(trimmed);
  }
}
```

All string parameters from tool calls must pass through `psStr()` or `psPath()` before interpolation into script bodies. Numeric and boolean parameters are validated by Zod schemas before use and are never interpolated as strings.

6.5 Output Serialisation

All toolkit functions pipe their return values through `ConvertTo-Json -Depth 10`. Several .NET types require special handling:

- **Circular references** - the Studio API object graph can contain them. Tool scripts use `Select-Object` to project only the required properties before serialising.

- `DateTime` - serialises to a verbose .NET format. A normalisation pass in each executor converts detected date strings to ISO 8601.
- `Language` **objects** (Studio toolkit) - expose `DisplayName` and `IsoAbbreviation`, both of which survive serialisation cleanly.

---

7. Configuration

7.1 Environment Variables

Variable	Tool group	Description
`STUDIO_VERSION`	studio_*	Studio folder name: `Studio17` (2022) or `Studio18` (2024). Defaults to `Studio18`.
`STUDIO_PS_PATH`	studio_*	Override path to PS5 x86 host. Defaults to `C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe`.
`STUDIO_MODULES_PATH`	studio_*	Path to the PS5 modules folder containing the Studio toolkit. When set, prepended to `$env:PSModulePath` in the Studio executor. Leave blank if using the default `Documents\WindowsPowerShell\Modules` location.
`PS7_PATH`	gs*, *lc**	Override path to PS7 host. Defaults to `pwsh.exe` (assumes it is on PATH).
`GS_MODULES_PATH`	gs_*	Path to the PS7 modules folder containing the GroupShare toolkit. When set, prepended to `$env:PSModulePath` before module imports in the GS preamble, ensuring the correct `AuthenticationHelper` and `ResourcesHelper` are loaded. Leave blank if using the default `Documents\PowerShell\Modules` location.
`LC_MODULES_PATH`	lc_*	Path to the PS7 modules folder containing the Language Cloud toolkit. When set, prepended to `$env:PSModulePath` before module imports in the LC preamble, ensuring the correct `AuthenticationHelper` and `ResourcesHelper` are loaded. Leave blank if using the default `Documents\PowerShell\Modules` location.
`GS_CREDENTIAL_STORE`	gs_*	Path to a folder containing DPAPI-encrypted GroupShare credential XML files. **Preferred over raw vars.** See §7.3.
`GS_SERVER_URL`	gs_*	GroupShare server URL. Used when `GS_CREDENTIAL_STORE` is not set.
`GS_USERNAME`	gs_*	GroupShare username. Used when `GS_CREDENTIAL_STORE` is not set.
`GS_PASSWORD`	gs_*	GroupShare password. Used when `GS_CREDENTIAL_STORE` is not set.
`LC_CREDENTIAL_STORE`	lc_*	Path to a folder containing DPAPI-encrypted Language Cloud credential XML files. **Preferred over raw vars.** See §7.3.
`LC_CLIENT_ID`	lc_*	Language Cloud application Client ID. Used when `LC_CREDENTIAL_STORE` is not set.
`LC_CLIENT_SECRET`	lc_*	Language Cloud application Client Secret. Used when `LC_CREDENTIAL_STORE` is not set.
`LC_TENANT_ID`	lc_*	Language Cloud Tenant ID (Trados Account ID). Used when `LC_CREDENTIAL_STORE` is not set.

7.2 Claude Desktop Configuration

Windows: `%APPDATA%\Claude\claude_desktop_config.json`

This file is maintained automatically by Claude Desktop when the extension is installed via the `.mcpb` installer. The env block below corresponds to the fields defined in `manifest.json` (see §7.4).

A full configuration using all three tool groups alongside the SDLXLIFF Refiner MCP, using credential store folders for GroupShare and Language Cloud:

```
{
  "mcpServers": {
    "sdlxliff-refiner": {
      "command": "node",
      "args": ["C:\\Tools\\sdlxliff-refiner-mcp\\dist\\index.js"],
      "env": {
        "SDLXLIFF_REFINER_PATH": "C:\\Tools\\SdlxliffRefiner\\sdlxliff-refiner.exe"
      }
    },
    "trados-powershell": {
      "command": "node",
      "args": ["C:\\Tools\\trados-powershell-mcp\\dist\\index.js"],
      "env": {
        "STUDIO_VERSION": "Studio18",
        "GS_CREDENTIAL_STORE":
"C:\\Your\\Chosen\\Path\\GroupSharePowershellToolkit\\CredentialStore",
        "LC_CREDENTIAL_STORE":
"C:\\Your\\Chosen\\Path\\LanguageCloudPowershellToolkit\\CredentialStore"
      }
    }
  }
}
```

Alternatively, using raw environment variables (credentials visible in the config file):

```
{
  "mcpServers": {
    "trados-powershell": {
      "command": "node",
      "args": ["C:\\Tools\\trados-powershell-mcp\\dist\\index.js"],
      "env": {
        "STUDIO_VERSION": "Studio18",
        "GS_SERVER_URL": "https://groupshare.example.com",
        "GS_USERNAME": "admin",
        "GS_PASSWORD": "password",
        "LC_CLIENT_ID": "your-client-id",
        "LC_CLIENT_SECRET": "your-client-secret",
        "LC_TENANT_ID": "your-tenant-id"
      }
    }
  }
}
```

A minimal configuration using only the Studio tools:

```
{
  "mcpServers": {
    "trados-powershell": {
      "command": "node",
      "args": ["C:\\Tools\\trados-powershell-mcp\\dist\\index.js"],
      "env": {
        "STUDIO_VERSION": "Studio18"
      }
    }
  }
}
```

7.3 Creating Credential Files

Credential files are DPAPI-encrypted XML files created once with PowerShell's `Export-CliXml`. They can only be decrypted by the same Windows user who created them, so they are safe to store on disk. Create one file per environment and save all files for a given toolkit into the same `CredentialStore` folder. The folder path is what you enter into the `GroupShare Credential Store` or `Language Cloud Credential Store` field when installing the desktop extension.

Run the following in a PowerShell 7 terminal once per environment:

GroupShare:

```
# You will be prompted for your GroupShare username and password
$cred = Get-Credential

[PSCustomObject]@{
    ServerUrl  = 'https://groupshare.example.com'
    Credential = $cred
} | Export-CliXml -Path
'C:\Your\Chosen\Path\GroupSharePowershellToolkit\CredentialStore\groupshare-prod.xml'
```

The GS credential XML contains `ServerUrl` (plain string) and `Credential` (a `PSCredential` object). At runtime the executor reads `$credData.ServerUrl`, `$credData.Credential.UserName`, and `$credData.Credential.GetNetworkCredential().Password`.

Language Cloud:

```
# Paste your LC Client ID, Client Secret, and Tenant ID when prompted
$lcTenant     = Read-Host 'Tenant ID'     -AsSecureString
$clientId     = Read-Host 'Client ID'     -AsSecureString
$clientSecret = Read-Host 'Client Secret' -AsSecureString

[PSCustomObject]@{
    lcTenant     = $lcTenant
    clientId     = $clientId
    clientSecret = $clientSecret
} | Export-CliXml -Path
'C:\Your\Chosen\Path\LanguageCloudPowershellToolkit\CredentialStore\languagecloud-prod.xml'
```

The LC credential XML contains `lcTenant`, `clientId`, and `clientSecret` as `SecureString` values. At runtime the executor decrypts each via `Marshal::SecureStringToBSTR` / `PtrToStringBSTR` before passing to `Get-AccessKey`. Note the property order: `lcTenant` first, matching the object structure used by the toolkit's own `01_Add_Credentials.ps1` helper script.

LC credentials from the Language Cloud web UI: Account menu → Integrations → Applications → your application → Client ID and Client Secret. Tenant ID is shown under Account → Account Information.

7.4 manifest.json

The `manifest.json` file at the project root defines the extension metadata and the user configuration form presented by Claude Desktop during installation. The `user_config` fields map directly to the environment variables read by the server. Fields left blank by the user are not written to the env block and the corresponding tool group is not registered.

```
{
  "name": "trados-powershell-mcp",
  "version": "1.7.0",
  "description": "MCP server exposing Trados Studio, GroupShare, and Language Cloud
management tools via the RWS PowerShell toolkits.",
  "author": "multifarious",
  "entry": "dist/index.js",
  "runtime": "node",
  "user_config": [
    {
      "key": "STUDIO_VERSION",
      "label": "Studio Version",
      "description": "Trados Studio folder name: Studio17 (2022) or Studio18 (2024)",
      "type": "string",
      "default": "Studio18",
      "required": false
    },
    {
      "key": "STUDIO_MODULES_PATH",
      "label": "Studio Toolkit Modules Path",
      "description": "Path to the PS5 modules folder containing the Studio PowerShell
Toolkit (e.g. C:\\Users\\YourName\\Documents\\WindowsPowerShell\\Modules). Leave blank if
installed in the default Documents\\WindowsPowerShell\\Modules location.",
      "type": "string",
      "required": false
    },
    {
      "key": "PS7_MODULES_PATH",
      "label": "GroupShare Toolkit Modules Path",
      "description": "Path to the PS7 modules folder containing the GroupShare PowerShell
Toolkit (e.g. C:\\Users\\YourName\\Documents\\PowerShell\\Modules\\GroupShare). Leave blank
if installed in the default Documents\\PowerShell\\Modules location.",
      "type": "string",
      "required": false
    },
    {
      "key": "LC_MODULES_PATH",
```

```
      "label": "Language Cloud Toolkit Modules Path",
      "description": "Path to the PS7 modules folder containing the Language Cloud
PowerShell Toolkit (e.g.
C:\\Users\\YourName\\Documents\\PowerShell\\Modules\\LanguageCloud). Leave blank if
installed in the default Documents\\PowerShell\\Modules location.",
      "type": "string",
      "required": false
    },
    {
      "key": "GS_CREDENTIAL_STORE",
      "label": "GroupShare Credential Store",
      "description": "Path to a folder containing DPAPI-encrypted GroupShare credential XML
files created with Export-CliXml. Preferred over raw credentials. See documentation for how
to create credential files.",
      "type": "string",
      "required": false
    },
    {
      "key": "GS_SERVER_URL",
      "label": "GroupShare Server URL",
      "description": "GroupShare server URL (e.g. https://groupshare.example.com). Used only
if credential store is not set.",
      "type": "string",
      "required": false
    },
    {
      "key": "GS_USERNAME",
      "label": "GroupShare Username",
      "description": "GroupShare username. Used only if credential store is not set.",
      "type": "string",
      "required": false
    },
    {
      "key": "GS_PASSWORD",
      "label": "GroupShare Password",
      "description": "GroupShare password. Used only if credential store is not set.",
      "type": "string",
      "required": false,
      "secret": true
    },
    {
      "key": "LC_CREDENTIAL_STORE",
      "label": "Language Cloud Credential Store",
      "description": "Path to a folder containing DPAPI-encrypted Language Cloud credential
XML files created with Export-CliXml. Preferred over raw credentials. See documentation for
how to create credential files.",
      "type": "string",
      "required": false
    },
    {
      "key": "LC_CLIENT_ID",
      "label": "Language Cloud Client ID",
```

```
      "description": "Language Cloud application Client ID. Used only if credential store is
not set.",
      "type": "string",
      "required": false
    },
    {
      "key": "LC_CLIENT_SECRET",
      "label": "Language Cloud Client Secret",
      "description": "Language Cloud application Client Secret. Used only if credential
store is not set.",
      "type": "string",
      "required": false,
      "secret": true
    },
    {
      "key": "LC_TENANT_ID",
      "label": "Language Cloud Tenant ID",
      "description": "Language Cloud Tenant ID (Trados Account ID). Used only if credential
store is not set.",
      "type": "string",
      "required": false
    }
  ]
}
```

---

8. Tool Definitions - Studio Group (studio_*)

All Studio tools use the `studioPs()` executor. The Studio toolkit requires PS5 x86 and the local Studio Professional installation.

Important note on `Get-Project`: This toolkit function takes a **directory path** (the project folder), not a `.sdlproj` file path. It searches the folder for the `.sdlproj` file. All Studio tools that open an existing project pass the project folder path to `Get-Project -projectDestinationPath`.

Important note on `Get-AnalyzeStatistics`: The exported toolkit function writes statistics to the console via `Write-Host` and returns nothing. To obtain serialisable statistics, tool scripts call `$project.GetProjectStatistics()` directly on the project object.

8.1 studio_list_projects

Description: List all Trados Studio file-based projects registered for the current user. Returns name, status, creation date, and project folder path.

Implementation note: Uses `studioPs({ bare: true })` - reads `[Environment]::GetFolderPath('MyDocuments')\{Studio folder}\Projects\projects.xml` directly via `Get-Content` + XML parsing (`Studio18` maps to folder `Studio 2024`, `Studio17` to `Studio 2022`). Does not load the Studio API. Faster and avoids a licence check for a read-only query. Source and target languages are not available from `projects.xml` - use `studio_get_project` for language details. The XML structure is `ProjectServer.Projects.ProjectListItem`, with project metadata in the child `ProjectInfo` element's

attributes. `ProjectFilePath` may be relative (resolved against the Projects folder) or absolute (used as-is). Note that Studio stores status `Started` internally for projects shown as "In Progress" in the UI.

Parameter	Type	Required	Description
`status`	string	no	Filter by status: `InProgress`, `Completed`, `Archived`

8.2 studio_get_project

Toolkit: `Get-Project`, `Get-TaskFileInfoFiles`, `$project.GetProjectStatistics()`

Description: Open an existing project and return its details - language pairs, bilingual files, and TM assignments. Optionally includes pre-computed analysis statistics.

Parameter	Type	Required	Description
`project_path`	string	yes	Full path to the project folder (directory containing the `.sdlproj` file)
`include_statistics`	boolean	no	Include analysis statistics (default: false)

8.3 studio_new_project

Toolkit: `New-Project`, `Get-Language`, `Get-Languages`

Description: Create a new file-based project from a source files folder. Returns the project folder path, project name, and language pair details.

Important note on `output_path`: The tool performs a pre-flight check before calling `New-Project`. If `output_path` already exists and contains any files or folders, the tool returns an error immediately. This prevents the Studio API from recursing into the existing content and creating thousands of nested sub-projects, which requires manual PowerShell cleanup to resolve. If `output_path` does not exist, the tool creates it automatically via `New-Item -ItemType Directory -Force`.

Source folder validation: The tool also validates that `source_folder` exists before calling `New-Project`. If the folder is missing, it returns an immediate error rather than letting the toolkit fail silently.

Parameter	Type	Required	Description
`name`	string	yes	Project name
`output_path`	string	yes	Folder where the project will be created
`source_language`	string	yes	Source language code (e.g. `en-GB`)
`target_languages`	string	yes	Comma-separated target language codes
`source_folder`	string	yes	Path to the folder containing source files
`tm_path`	string	no	Path to a `.sdltm` file to assign
`due_date`	string	no	ISO date string

Parameter	Type	Required	Description
`task_sequence`	string	no	One of `Prepare without project TM` (default), `Prepare`, `Analyse only`, `Translate only`

8.4 studio_remove_project

Toolkit: `Remove-Project`

Description: Remove a project from Studio. Does not delete files from disk unless `delete_files` is true.

Parameter	Type	Required	Description
`project_path`	string	yes	Full path to the project folder
`delete_files`	boolean	no	Also delete the project folder (default: false)

8.5 studio_analyze

Toolkit: `Get-Project`, `$project.GetProjectStatistics()`

Description: Return pre-computed analysis statistics for a project by language and match category (new, exact, fuzzy bands, repetitions, locked). The analysis task must have been run previously (typically by `studio_new_project` with a task sequence that includes analysis, or via the Studio UI). This tool reads existing statistics only - it does not trigger a new analysis run.

Parameter	Type	Required	Description
`project_path`	string	yes	Full path to the project folder
`target_language`	string	no	Limit to one target language

8.6 studio_export_package

Toolkit: `Get-TaskFileInfoFiles`, `Get-Language`; Studio API `ProjectPackageCreationOptions`, `CreateProjectPackage`, `SavePackageAs`

Description: Create a translation package (`.sdlppx`) for sending to a linguist. `Get-PackageOptions` from the toolkit is not used because it hardcodes `IncludeMainTranslationMemories = false` with no override; `ProjectPackageCreationOptions` is constructed directly instead.

Parameter	Type	Required	Description
`project_path`	string	yes	Full path to the project folder
`output_path`	string	yes	Destination path for the `.sdlppx` file, or a folder when exporting all languages
`target_language`	string	no	Export one target language only
`include_tm`	boolean	no	Include project TM in package (default: false)

8.7 studio_import_package

Toolkit: `Import-Package`

Description: Import a return package (`.sdlrpx`) from a linguist back into the project.

Parameter	Type	Required	Description
`project_path`	string	yes	Full path to the project folder
`package_path`	string	yes	Path to the `.sdlrpx` file

8.8 studio_pretranslate

Toolkit: `Get-Project`; Studio API `RunAutomaticTask`

Description: Run pre-translation on a project against its assigned TMs using the `Sdl.ProjectApi.AutomaticTasks.Translate` task. Returns the task status and any messages per target language.

Parameter	Type	Required	Description
`project_path`	string	yes	Full path to the project folder
`target_language`	string	no	Limit to one target language

8.9 studio_list_tms

Toolkit: `Open-FileBasedTM`

Description: List file-based TMs (`.sdltm` files). Supports two modes: if `folder` is provided, scans that folder for `.sdltm` files; if `folder` is omitted, reads registered TMs from `TranslationMemoryRepository.xml` at `%LOCALAPPDATA%\Trados\Trados Studio\{STUDIO_VERSION}\`. Opens each TM to read its source and target language from `LanguageDirection.SourceLanguage.Name` / `TargetLanguage.Name`.

Important: The repository XML path uses the AppData version key (e.g. `Studio18`), not the Documents display name (`Studio 2024`). These are different. The `STUDIO_VERSION` env var provides the correct key.

Parameter	Type	Required	Description
`folder`	string	no	Folder path to search (if omitted, reads registered TMs)
`recursive`	boolean	no	Search subfolders when using folder mode (default: false)

8.10 studio_new_tm

Toolkit: `New-FileBasedTM`, `Get-Language`

Description: Create a new file-based TM.

Parameter	Type	Required	Description
`path`	string	yes	Full path for the new `.sdltm` file
`source_language`	string	yes	Source language code
`target_language`	string	yes	Target language code
`description`	string	no	Optional description

8.11 studio_import_tmx

Toolkit: `Import-Tmx`

Description: Import a TMX file into an existing file-based TM. `Import-Tmx` accepts `-tmPath` directly and resolves the language direction internally; no prior call to `Open-FileBasedTM` is needed.

Parameter	Type	Required	Description
`tm_path`	string	yes	Path to the `.sdltm` file
`tmx_path`	string	yes	Path to the `.tmx` file

8.12 studio_export_tmx

Toolkit: `Export-Tmx`

Description: Export a file-based TM to TMX. `Export-Tmx` accepts `-tmPath` directly; no prior call to `Open-FileBasedTM` is needed.

Parameter	Type	Required	Description
`tm_path`	string	yes	Path to the `.sdltm` file
`output_path`	string	yes	Destination path for the `.tmx` file

8.13 studio_list_project_templates

Toolkit: `ApplicationFactory.CreateApplication()` (Studio Project Automation API)

Description: List all project templates registered in Trados Studio. Uses the `ApplicationFactory` API to access the `LocalProjectServer.GetProjectTemplates()` method. Requires `CallEnsurePluginRegistryIsCreated()` before `CreateApplication()` to initialise the plugin registry.

No parameters.

---

9. Tool Definitions - GroupShare Group (gs_*)

All GroupShare tools use the `ps7("groupshare", ...)` executor. The preamble loads six modules - `AuthenticationHelper`, `ProjectServerHelper`, `ResourcesHelper`, `UserManagerHelper`, `BackgroundTaskHelper`, and `SystemConfigurationHelper` - all initialised with `$serverUrl` as an argument list. The preamble exposes `$authToken`.

`gs_list_credentials` and `gs_set_credential` use `ps7("groupshare", ..., { bare: true })` and do not load any modules or connect to any server.

Key pattern: GroupShare toolkit functions take PSObjects, not ID strings. Any tool that needs to act on a project, TM, template, organisation, or container must first call the appropriate lookup function to obtain the PSObject, then pass it to the action function.

9.1 gs_list_credentials

Description: List GroupShare credential files in the credential store folder (`GS_CREDENTIAL_STORE`). Decrypts and returns the server URL and username from each XML file. Does not connect to GroupShare. Use this to discover which credentials are available before calling `gs_set_credential`.

Uses `ps7({ bare: true })` - no module loading or server connection.

Parameter	Type	Required	Description
`folder_path`	string	no	Override the store folder path. Defaults to `GS_CREDENTIAL_STORE`.

9.2 gs_set_credential

Description: Select a GroupShare credential file from the store and activate it for the current session. Decrypts the file to confirm it is a valid GS credential, then stores the file path in the server's state module. All subsequent GS tool calls will use this credential until changed or until Claude Desktop is restarted.

Uses `ps7({ bare: true })` - no module loading or server connection.

Returns the server URL and username from the selected file as confirmation.

Parameter	Type	Required	Description
`credential_file`	string	yes	Filename (e.g. `groupshare-prod.xml`) or full path. If a filename only, resolved against `GS_CREDENTIAL_STORE`.

9.3 gs_list_projects

Toolkit: `Get-AllProjects`

Description: List projects on the GroupShare server. Supports filtering by status, organisation, date range, and name. Set `include_sub_organizations` to true to include projects from child organisations. Use `group_by` to get a count summary (e.g. by organisation or status) instead of individual projects. Use `compact` mode for a scannable listing. All list results include `total` and `returnedCount` metadata.

Implementation note: Always passes `-defaultPublishDates $false -defaultDueDates $false` to `Get-AllProjects` to avoid the toolkit's restrictive date defaults that hide projects. Without these flags, the toolkit applies narrow date ranges that cause most projects to be excluded from results.

Parameter	Type	Required	Description
`statuses`	string	no	Comma-separated statuses: `Pending`, `In Progress`, `Completed`, `Archived` - split into a string array before passing to the toolkit
`organization_name`	string	no	Filter to projects within a specific organisation (resolved to PSObject via `Get-Organization`)
`include_sub_organizations`	boolean	no	Include projects from child organisations (default: false). Maps to `-includeSubOrganizations`
`due_start`	string	no	Due date range start (YYYY-MM-DD)
`due_end`	string	no	Due date range end (YYYY-MM-DD)
`name_filter`	string	no	Wildcard filter on project name (e.g. `*Newsletter*`). Applied client-side via `Where-Object`
`group_by`	string	no	Return count summary grouped by `organisation`, `status`, or `source_language` instead of individual projects
`compact`	boolean	no	Return only name, status, organisation, and languages (default: false)
`max_results`	number	no	Maximum number of projects to return (default: 50)

9.4 gs_get_project

Toolkit: `Get-Project`, `Get-ProjectSettings`, `Get-FilesPhasesFromProject`

Description: Get full details of a GroupShare project - settings, language pairs, and file list with phase and assignment information.

Parameter	Type	Required	Description
`project_id`	string	no	Project ID (preferred; passed to `Get-Project -projectId`)
`project_name`	string	no	Project name (used if ID not provided; passed to `Get-Project -projectName`)

9.5 gs_new_project

Toolkit: `New-Project`, `Get-Organization`, `Get-ProjectTemplate`

Description: Create a new project on the GroupShare server. `New-Project` takes `$organization` and `$projectTemplate` as PSObjects looked up before the call. Returns the new project's details.

Parameter	Type	Required	Description
`name`	string	yes	Project name
`organization_name`	string	yes	Organisation name (resolved to PSObject via `Get-Organization`)
`template_name`	string	yes	Project template name (resolved to PSObject via `Get-ProjectTemplate`)
`files_path`	string	yes	Path to a source file, folder, or zip
`due_date`	string	no	Due date (YYYY-MM-DD or YYYY-MM-DDThh:mm)
`description`	string	no	Project description

9.6 gs_update_project_status

Toolkit: `Get-Project`, `Update-ProjectStatus`

Description: Change a project's status. `Update-ProjectStatus` takes a `$project` PSObject; the project is looked up first. Accepted status values are `Completed` and `Started`.

Parameter	Type	Required	Description
`project_id`	string	no	Project ID (preferred)
`project_name`	string	no	Project name (used if ID not provided)
`status`	string	yes	`Completed` or `Started`

9.7 gs_export_package

Toolkit: `Get-Project`, `Export-Package`

Description: Export a translation package from a GroupShare project. `Export-Package` takes a `$project` PSObject and `$packageDestinationPath`.

Parameter	Type	Required	Description
`project_id`	string	no	Project ID (preferred)
`project_name`	string	no	Project name (used if ID not provided)
`output_path`	string	yes	Destination path for the `.sdlppx` file

9.8 gs_import_package

Toolkit: `Get-Project`, `Import-Package`

Description: Import a return package into a GroupShare project. `Import-Package` takes a `$project` PSObject and `$packagePath`.

Parameter	Type	Required	Description
`project_id`	string	no	Project ID (preferred)
`project_name`	string	no	Project name (used if ID not provided)
`package_path`	string	yes	Path to the `.sdlrpx` file

9.9 gs_get_analysis_report

Toolkit: `Get-Project`, `Get-AnalysisReports`

Description: Save and return the analysis report for a GroupShare project. `Get-AnalysisReports` takes a `$project` PSObject, `$outputFile`, and optional `$languageCode`.

Parameter	Type	Required	Description
`project_id`	string	no	Project ID (preferred)
`project_name`	string	no	Project name (used if ID not provided)
`output_path`	string	yes	Destination path for the report
`language_code`	string	no	Target language code; all languages if omitted

9.10 gs_save_project_files

Toolkit: `Get-Project`, `Save-AllProjectsFile`

Description: Download project files as a zip archive. `Save-AllProjectsFile` takes a `$project` PSObject, `$outputLocation`, `$type`, and `$includeTMs`.

Parameter	Type	Required	Description
`project_id`	string	no	Project ID (preferred)
`project_name`	string	no	Project name (used if ID not provided)
`output_path`	string	yes	Destination path for the `.zip` file
`type`	string	no	`all` or `targetnativefiles` (default: `all`)
`include_tms`	boolean	no	Include TMs in the download (only valid with `targetnativefiles`)

9.11 gs_list_tms

Toolkit: `Get-AllTMs`, `Get-TMsByContainer`, `Get-Container` (SystemConfigurationHelper)

Description: List translation memories on the GroupShare server. If `container_name` is provided, resolves it to a PSObject via `SystemConfigurationHelper.Get-Container` and passes it to `Get-TMsByContainer`. Otherwise calls `Get-AllTMs`. Use `name_filter` with wildcards to search. All list results include `total`, `matchingCount`, and `returnedCount` metadata.

Parameter	Type	Required	Description
`container_name`	string	no	Filter to TMs within a specific container
`name_filter`	string	no	Wildcard filter on TM name (e.g. `*Marketing*`). Applied client-side via `Where-Object`
`compact`	boolean	no	Return only name and language pair (default: false)
`max_results`	number	no	Maximum number of TMs to return (default: 50)

9.12 gs_new_tm

Toolkit: `New-TM`, `Get-Container` (SystemConfigurationHelper), `Get-Organization`, `Get-LanguageDirections`

Description: Create a new server-based TM. `New-TM` takes a `$container` PSObject, an `$organization` PSObject, and a `$languageDirections` PSObject array. The container is looked up via `SystemConfigurationHelper.Get-Container`, the organisation via `Get-Organization`, and language directions are built via `ResourcesHelper.Get-LanguageDirections -source $source -target @($target)`.

Parameter	Type	Required	Description
`name`	string	yes	TM name
`container_name`	string	yes	Container name
`organization_name`	string	yes	Owner organisation name
`source_language`	string	yes	Source language code
`target_language`	string	yes	Target language code
`description`	string	no	Optional description

9.13 gs_import_tmx

Toolkit: `Get-TM`, `Import-TMX`

Description: Import a TMX file into a server-based TM. `Import-TMX` takes a `$tm` PSObject (from `Get-TM`) plus source and target language codes and the TMX path.

Parameter	Type	Required	Description
`tm_name`	string	yes	TM name
`tmx_path`	string	yes	Path to the `.tmx` file
`source_language`	string	yes	Source language code
`target_language`	string	yes	Target language code

9.14 gs_export_tmx

Toolkit: `Get-TM`, `Export-TMX`

Description: Export a server-based TM to TMX. `Export-TMX` takes a `$tm` PSObject plus source and target language codes and output path. The function enforces a `.tmx.gz` extension on the output file.

Parameter	Type	Required	Description
`tm_name`	string	yes	TM name
`output_path`	string	yes	Destination path (must end in `.tmx.gz`)
`source_language`	string	yes	Source language code
`target_language`	string	yes	Target language code

9.15 gs_list_project_templates

Toolkit: `Get-AllProjectTemplates`

Description: List project templates available on the GroupShare server. Use this to discover template names for `gs_new_project`. Templates have an `OrganizationId` property and can be filtered to a specific organisation by combining with `gs_list_organizations`. All list results include `total` and `returnedCount` metadata.

Parameter	Type	Required	Description
`name_filter`	string	no	Wildcard filter on template name (e.g. `*Legal*`). Applied client-side via `Where-Object`
`compact`	boolean	no	Return only template names (default: false)
`max_results`	number	no	Maximum number of templates to return (default: 50)

9.16 gs_list_organizations

Toolkit: `Get-AllOrganizations`

Description: List organisations on the GroupShare server. Use `parent_path` to browse the hierarchy: `/` for top-level organisations, `/Consoltec` for its direct children, etc. The filter works by stripping the last path segment from each organisation's `Path` property and comparing it to `parent_path`. Use `name_filter` with wildcards to search across all organisations. Use `compact` mode for a scannable name + path listing. All list results include `total`, `matchingCount`, and `returnedCount` metadata.

Parameter	Type	Required	Description
`parent_path`	string	no	Show only direct children of this path (e.g. `/` for top-level, `/Consoltec` for its children)
`name_filter`	string	no	Wildcard filter on organisation name (e.g. `*QA*`). Applied client-side via `Where-Object`
`compact`	boolean	no	Return only name and path (default: false)
`max_results`	number	no	Maximum number of organisations to return (default: 50)

9.17 gs_list_users

Toolkit: `Get-AllUsers`, `Get-Organization`

Description: List users on the GroupShare server. If `organization_name` is provided, resolves it to a PSObject and passes it to `Get-AllUsers -organization`. The `$maxLimit` parameter controls the maximum returned (default 100).

Parameter	Type	Required	Description
`organization_name`	string	no	Filter to users within this organisation
`max_limit`	number	no	Maximum number of users to return (default: 100)

9.18 gs_get_background_tasks

Toolkit: `Get-AllBackgroundTasks` (BackgroundTaskHelper)

Description: List background tasks on the GroupShare server. Use this to poll the status of async operations such as project creation and TMX import. Call after `gs_new_project` or `gs_import_tmx` to check whether the operation has completed. Filter by `status` to surface pending or failed tasks only.

Parameter	Type	Required	Description
`status`	string	no	Filter by status: `Pending`, `Running`, `Completed`, `Failed`, `Cancelled`
`task_type`	string	no	Filter by task type substring (e.g. `CreateProject`, `ImportTMX`)

9.19 gs_new_user

Toolkit: `New-User`, `Get-Organization` (UserManagerHelper)

Description: Create a new user on the GroupShare server and add them to an organisation. The organisation is resolved to a PSObject via `Get-Organization` before the call.

Parameter	Type	Required	Description
`username`	string	yes	Username (login name)
`password`	string	yes	Initial password
`email`	string	yes	Email address
`display_name`	string	no	Display name (defaults to username if omitted)
`organization_name`	string	yes	Organisation to add the user to

9.20 gs_update_user

Toolkit: `Get-User`, `Update-User` (UserManagerHelper)

Description: Update an existing user's properties. Only the fields provided are changed. The user is looked up via `Get-User` before the update.

Parameter	Type	Required	Description
`username`	string	yes	Username of the user to update
`email`	string	no	New email address
`display_name`	string	no	New display name
`is_enabled`	boolean	no	Enable or disable the account
`new_password`	string	no	New password

9.21 gs_new_role

Toolkit: `New-Role`, `Get-Organization` (UserManagerHelper)

Description: Create a new role on the GroupShare server. Use `gs_update_role_to_user` to assign the role to users.

Parameter	Type	Required	Description
`name`	string	yes	Role name
`organization_name`	string	yes	Organisation the role belongs to
`permissions`	string	no	Comma-separated permission strings to assign

9.22 gs_update_role_to_user

Toolkit: `Get-User`, `Get-Organization`, `Get-AllRoles`, `Update-RoleToUser` (UserManagerHelper)

Description: Assign a role to a GroupShare user within an organisation. The user, organisation, and role are all resolved to PSObjects before the assignment call.

Parameter	Type	Required	Description
`username`	string	yes	Username to assign the role to
`role_name`	string	yes	Name of the role to assign
`organization_name`	string	yes	Organisation context for the role assignment

9.23 gs_move_organization_resources

Toolkit: `Get-Organization`, `Move-OrganizationResources` (UserManagerHelper)

Description: Move all resources (projects, TMs, users) from one organisation to another. Both organisations are resolved to PSObjects before the call. Use with caution - this affects all resources in the source organisation.

Parameter	Type	Required	Description
`source_organization_name`	string	yes	Organisation to move resources from
`target_organization_name`	string	yes	Organisation to move resources to

9.24 gs_list_containers

Toolkit: `Get-AllContainers` (SystemConfigurationHelper), `Get-Organization`

Description: List containers on the GroupShare server. If `organization_name` is provided, filters to containers owned by that organisation by matching the container's `OwnerId` property against the organisation's `UniqueId`. Use `name_filter` with wildcards to search by `DisplayName`. All list results include `total`, `matchingCount`, and `returnedCount` metadata.

Parameter	Type	Required	Description
`organization_name`	string	no	Filter to containers owned by this organisation
`name_filter`	string	no	Wildcard filter on container name (e.g. `*Production*`). Applied client-side via `Where-Object` on `DisplayName`
`compact`	boolean	no	Return only name and containerId (default: false)
`max_results`	number	no	Maximum number of containers to return (default: 50)

9.25 gs_org_report

Toolkit: `Get-Organization`, `Get-AllOrganizations`, `Get-AllProjects`, `Get-AllContainers`, `Get-AllTMs`, `Get-AllProjectTemplates`, `Get-AllUsers`

Description: Generate a comprehensive report for a GroupShare organisation. Gathers child organisations (full subtree), projects with status and language summaries, containers owned by the organisation with TMs per container, project templates belonging to the organisation, and users - all in a single authenticated script call. Returns a structured JSON object with counts and items for each resource type.

Implementation note: Containers are matched by `$_.OwnerId -eq $org.UniqueId`. TMs are matched by `$_.ContainerId -eq $container.ContainerId`. Project templates are matched by `$_.OrganizationId -eq $org.UniqueId`. Child organisations are found by matching paths that start with the target org's path followed by `/`. Projects use `-includeSubOrganizations`, `-defaultPublishDates $false`, and `-defaultDueDates $false` for complete results.

Parameter	Type	Required	Description
`organization_name`	string	yes	Organisation name to report on
`include_sub_organizations`	boolean	no	Include projects from child organisations (default: true)
`max_projects`	number	no	Maximum number of projects to return in detail (default: 100)
`max_users`	number	no	Maximum number of users to return (default: 100)

---

10. Tool Definitions - Language Cloud Group (lc_*)

All Language Cloud tools use the `ps7("languagecloud", ...)` executor. The preamble loads five modules - `AuthenticationHelper`, `ProjectHelper`, `ResourcesHelper`, `UsersHelper`, and `TerminologyHelper`. The preamble exposes `$accessKey` (a PSObject with `.token` and `.tenant` properties, returned by `Get-AccessKey`). Every toolkit function receives `$accessKey` as its first argument.

`lc_list_credentials` and `lc_set_credential` use `ps7("languagecloud", ..., { bare: true })` and do not load any modules or connect to any server.

Key pattern: Unlike GroupShare, most LC toolkit functions accept resource IDs or names directly (as `$translationMemoryId` / `$translationMemoryName`, `$projectTemplateId` / `$projectTemplateName`, etc.) rather than requiring a PSObject lookup first. Where both are optional, passing the name is sufficient.

10.1 lc_list_credentials

Description: List Language Cloud credential files in the credential store folder (`LC_CREDENTIAL_STORE`). Decrypts and returns the tenant ID and client ID from each XML file. Does not connect to Language Cloud. Use this to discover which credentials are available before calling `lc_set_credential`.

Uses `ps7({ bare: true })` - no module loading or server connection.

Parameter	Type	Required	Description
`folder_path`	string	no	Override the store folder path. Defaults to `LC_CREDENTIAL_STORE`.

10.2 lc_set_credential

Description: Select a Language Cloud credential file from the store and activate it for the current session. Decrypts the file to confirm it is a valid LC credential, then stores the file path in the server's state module. All subsequent LC tool calls will use this credential until changed or until Claude Desktop is restarted.

Uses `ps7({ bare: true })` - no module loading or server connection.

Returns the tenant ID and client ID from the selected file as confirmation.

Parameter	Type	Required	Description
`credential_file`	string	yes	Filename (e.g. `languagecloud-prod.xml`) or full path. If a filename only, resolved against `LC_CREDENTIAL_STORE`.

10.3 lc_list_projects

Toolkit: `Get-AllProjects`

Description: List projects in Language Cloud.

No parameters.

10.4 lc_get_project

Toolkit: `Get-Project`

Description: Get details of a specific Language Cloud project. Pass either ID or name.

Parameter	Type	Required	Description
`project_id`	string	no	Project ID
`project_name`	string	no	Project name

10.5 lc_new_project

Toolkit: `New-Project`

Description: Create a new Language Cloud project. `$dueDate` (YYYY-MM-DD) and `$dueTime` (HH:MM) are separate mandatory parameters. `$filesPath` is the path to the source files folder. The project template, workflow, translation engine, and file type configuration are specified by name or ID via the `*IdOrName` parameters. An optional `$locationId` scopes the project to a specific location (use `lc_list_locations` to discover location IDs). Reference files can be attached via `$referenceFileNames`.

Parameter	Type	Required	Description
`name`	string	yes	Project name
`due_date`	string	yes	Due date (YYYY-MM-DD)
`due_time`	string	yes	Due time (HH:MM, e.g. `17:00`)
`files_path`	string	yes	Path to the folder containing source files
`template_name`	string	no	Project template name or ID (`$projectTemplateIdOrName`)
`source_language`	string	no	Source language code (overrides template if provided)
`target_languages`	string	no	Comma-separated target language codes (overrides template if provided)
`translation_engine`	string	no	Translation engine name or ID
`workflow`	string	no	Workflow name or ID
`file_type_configuration`	string	no	File type configuration name or ID (`$fileTypeConfigurationIdOrName`)
`location_id`	string	no	Location ID to scope the project to (from `lc_list_locations`)
`reference_file_names`	string	no	Comma-separated reference file names to attach
`description`	string	no	Project description

10.6 lc_list_project_templates

Toolkit: `Get-AllProjectTemplates`

Description: List all project templates in Language Cloud. Use this to discover template names for `lc_new_project`.

No parameters.

10.7 lc_list_tms

Toolkit: `Get-AllTranslationMemories`

Description: List all translation memories in Language Cloud.

No parameters.

10.8 lc_get_tm

Toolkit: `Get-TranslationMemory`

Description: Get details of a specific Language Cloud TM.

Parameter	Type	Required	Description
`tm_name`	string	no	TM name
`tm_id`	string	no	TM ID

10.9 lc_new_tm

Toolkit: `New-TranslationMemory`, `Get-LanguagePair`

Description: Create a new Language Cloud TM. `$languageProcessingIdOrName` and `$fieldTemplateIdOrName` are mandatory toolkit parameters. Use `lc_list_language_processing_rules` and `lc_list_field_templates` to discover available values. The language pair is built via `Get-LanguagePair -sourceLanguage $src -targetLanguages @($tgt)`. An optional `$locationId` scopes the TM to a specific location.

Parameter	Type	Required	Description
`name`	string	yes	TM name
`source_language`	string	yes	Source language code
`target_language`	string	yes	Target language code
`language_processing`	string	yes	Language processing rule name or ID (use `lc_list_language_processing_rules` to discover)
`field_template`	string	yes	Field template name or ID (use `lc_list_field_templates` to discover)
`location_id`	string	no	Location ID to scope the TM to (from `lc_list_locations`)
`description`	string	no	Optional description

10.10 lc_import_tm

Toolkit: `Import-TranslationMemory`

Description: Import a TMX file into a Language Cloud TM. The TM is specified by name or ID. The import file path is passed as `$importFileLocation`. Optional parameters control how duplicate and unknown content is handled during import.

Parameter	Type	Required	Description
`tm_name`	string	no	TM name
`tm_id`	string	no	TM ID
`import_file_path`	string	yes	Path to the `.tmx` file (`$importFileLocation`)
`source_language`	string	yes	Source language code
`target_language`	string	yes	Target language code
`import_as_plain_text`	boolean	no	Import as plain text, stripping formatting (default: false)
`export_invalid_tus`	boolean	no	Export invalid translation units to a separate file (default: false)

Parameter	Type	Required	Description
`trigger_recompute_statistics`	boolean	no	Recompute TM statistics after import (default: true)
`target_segments_differ_option`	string	no	How to handle TUs where target differs: `addNew`, `overwrite`, `keepExisting`, `mergeIntoExisting`
`unknown_fields_option`	string	no	How to handle unknown fields: `addToTranslationMemory`, `ignore`, `skipTranslationUnit`
`confirmation_levels`	string	no	Comma-separated confirmation levels to import: `translated`, `approvedTranslation`, `approvedSignOff`, `draft`, `rejectedTranslation`, `rejectedSignOff`. Imports all levels if omitted. Maps to `-onlyImportSegmentsWithConfirmationLevels`

10.11 lc_export_tm

Toolkit: `Export-TranslationMemory`

Description: Export a Language Cloud TM to TMX. The TM is specified by name or ID.

Parameter	Type	Required	Description
`tm_name`	string	no	TM name
`tm_id`	string	no	TM ID
`output_path`	string	yes	Destination path for the export (`$outputFilePath`)
`source_language`	string	yes	Source language code
`target_language`	string	yes	Target language code

10.12 lc_list_customers

Toolkit: `Get-AllCustomers`

Description: List all customers in Language Cloud.

No parameters.

10.13 lc_list_workflows

Toolkit: `Get-AllWorkflows`

Description: List all workflows available in Language Cloud. Use this to understand available workflow options when creating projects.

No parameters.

10.14 lc_list_translation_engines

Toolkit: `Get-AllTranslationEngines`

Description: List all translation engines configured in Language Cloud.

No parameters.

10.15 lc_list_termbases

Toolkit: `Get-AllTermbases` (TerminologyHelper)

Description: List all termbases in Language Cloud.

No parameters.

10.16 lc_list_users

Toolkit: `Get-AllUsers` (UsersHelper)

Description: List users in Language Cloud.

No parameters.

10.17 lc_new_termbase

Toolkit: `New-Termbase` (TerminologyHelper)

Description: Create a new termbase in Language Cloud. The termbase structure can be defined either by a named termbase template (`$termbaseTemplateName`) or by an XDT file (`$pathToXDT`). When using an XDT file, set `inherit_languages` to true to derive the termbase languages from the XDT definition. An optional `$locationId` scopes the termbase to a specific location.

Parameter	Type	Required	Description
`name`	string	yes	Termbase name
`termbase_template`	string	no	Termbase template name or ID. Required if `xdt_path` is not provided.
`xdt_path`	string	no	Path to an XDT file defining the termbase structure. Alternative to `termbase_template`.
`inherit_languages`	boolean	no	Inherit languages from the XDT file (default: true when `xdt_path` is provided)
`location_id`	string	no	Location ID to scope the termbase to (from `lc_list_locations`)
`description`	string	no	Optional description

10.18 lc_import_termbase

Toolkit: `Import-Termbase` (TerminologyHelper)

Description: Import a termbase file (MultiTerm XML, TBX, or other supported format) into a Language Cloud termbase. Provide either `termbase_name` or `termbase_id` . Optional parameters control duplicate handling and import strictness.

Parameter	Type	Required	Description
`termbase_name`	string	no	Termbase name
`termbase_id`	string	no	Termbase ID
`import_file_path`	string	yes	Path to the import file
`duplicate_entries_strategy`	string	no	How to handle duplicate entries: `merge`, `overwrite`, `skipExisting`
`strict_import`	boolean	no	Enable strict import validation (default: false)

10.19 lc_export_termbase

Toolkit: `Export-Termbase` (TerminologyHelper)

Description: Export a Language Cloud termbase to a file. Provide either `termbase_name` or `termbase_id`.

Parameter	Type	Required	Description
`termbase_name`	string	no	Termbase name
`termbase_id`	string	no	Termbase ID
`output_path`	string	yes	Destination path for the exported file

10.20 lc_list_locations

Toolkit: `Get-AllLocations` (ResourcesHelper)

Description: List all locations in Language Cloud. Locations are the organisational hierarchy in LC (equivalent to GroupShare organisations). Many LC resources (TMs, termbases, project templates, customers) are scoped to a location. Use this tool to discover location IDs for use with `lc_new_project`, `lc_new_tm`, `lc_new_termbase`, `lc_new_project_template`, and `lc_new_customer`. The root location is typically the first entry returned.

No parameters.

10.21 lc_list_groups

Toolkit: `Get-AllGroups` (UsersHelper)

Description: List all user groups in Language Cloud.

No parameters.

10.22 lc_list_file_type_configurations

Toolkit: `Get-AllFileTypeConfigurations` (ResourcesHelper)

Description: List all file type configurations in Language Cloud. Use this to discover file type configuration names or IDs for `lc_new_project`. Each configuration has an associated location.

No parameters.

10.23 lc_list_language_processing_rules

Toolkit: `Get-AllLanguageProcessingRules` (ResourcesHelper)

Description: List all language processing rules in Language Cloud. Use this to discover the language processing rule name or ID required by `lc_new_tm`. Each rule has an associated location. Use `location_strategy` to control how rules are resolved relative to a location.

Parameter	Type	Required	Description
`location_id`	string	no	Scope to a specific location
`location_strategy`	string	no	Location resolution strategy (e.g. `bloodline`). When set, also searches parent locations.

10.24 lc_list_field_templates

Toolkit: `Get-AllFieldTemplates` (ResourcesHelper)

Description: List all field templates in Language Cloud. Use this to discover the field template name or ID required by `lc_new_tm`. Each template has an associated location. Use `location_strategy` to control how templates are resolved relative to a location.

Parameter	Type	Required	Description
`location_id`	string	no	Scope to a specific location
`location_strategy`	string	no	Location resolution strategy (e.g. `bloodline`). When set, also searches parent locations.

10.25 lc_list_pricing_models

Toolkit: `Get-AllPricingModels` (ResourcesHelper)

Description: List all pricing models in Language Cloud.

No parameters.

10.26 lc_list_schedule_templates

Toolkit: `Get-AllScheduleTemplates` (ResourcesHelper)

Description: List all schedule templates in Language Cloud.

No parameters.

10.27 lc_list_supported_languages

Toolkit: `Get-SupportedLanguages` (ResourcesHelper)

Description: List all languages supported by Language Cloud. Returns language code, English name, script direction, and whether the code is a neutral (region-independent) code. Useful for discovering valid language codes before creating projects, TMs, or termbases.

No parameters.

10.28 lc_new_customer

Toolkit: `New-Customer` (ResourcesHelper)

Description: Create a new customer in Language Cloud. The customer is scoped to a location. Creating a customer also creates a corresponding location as a child of the specified parent location; this new location may take a few seconds to become visible via `lc_list_locations`.

Parameter	Type	Required	Description
`name`	string	yes	Customer name
`location_id`	string	yes	Parent location ID (from `lc_list_locations`)

10.29 lc_update_customer

Toolkit: `Update-Customer` (ResourcesHelper)

Description: Update an existing customer's properties. Currently supports updating the RAG (Red/Amber/Green) status.

Parameter	Type	Required	Description
`customer_id`	string	yes	Customer ID (from `lc_list_customers`)
`rag_status`	string	no	RAG status: `red`, `amber`, `green`

10.30 lc_remove_customer

Toolkit: `Remove-Customer` (ResourcesHelper)

Description: Delete a customer from Language Cloud. Child customers must be removed before their parent. Any resources (projects, TMs, termbases) still associated with the customer's location must be removed first.

Parameter	Type	Required	Description
`customer_id`	string	yes	Customer ID (from `lc_list_customers`)

10.31 lc_remove_tm

Toolkit: `Remove-TranslationMemory` (ResourcesHelper)

Description: Delete a translation memory from Language Cloud. The TM must not be in use by any active project.

Parameter	Type	Required	Description
`tm_id`	string	yes	Translation memory ID (from `lc_list_tms` or `lc_get_tm`)

10.32 lc_new_project_template

Toolkit: `New-ProjectTemplate` (ProjectHelper)

Description: Create a new project template in Language Cloud. The template defines default settings (source/target languages, file type configuration, translation engine, workflow) that can be reused across multiple projects via `lc_new_project`.

Parameter	Type	Required	Description
`name`	string	yes	Template name
`location_id`	string	yes	Location ID to scope the template to (from `lc_list_locations`)
`source_language`	string	yes	Source language code
`target_languages`	string	yes	Comma-separated target language codes
`file_type_configuration`	string	yes	File type configuration name or ID (`$fileTypeConfigurationIdOrName`)
`translation_engine`	string	yes	Translation engine name or ID
`workflow`	string	yes	Workflow name or ID

10.33 lc_remove_project_template

Toolkit: `Remove-ProjectTemplate` (ProjectHelper)

Description: Delete a project template from Language Cloud.

Parameter	Type	Required	Description
`template_id`	string	yes	Project template ID (from `lc_list_project_templates`)

---

11. Tool Registration Pattern

Each `register.ts` file imports the tool definition files and registers them on the server. Example for the Studio group:

```
import { McpServer } from "@modelcontextprotocol/sdk/server/mcp.js";
import { registerListProjectsTool }   from "./list-projects.js";
import { registerGetProjectTool }     from "./get-project.js";
// ... other imports

export function registerStudioTools(server: McpServer) {
  registerListProjectsTool(server);
  registerGetProjectTool(server);
  // ...
}
```

Each individual tool file follows the same pattern:

```
export function registerAnalyzeTool(server: McpServer) {
  server.tool(
    "studio_analyze",
    "Return pre-computed analysis statistics for a Trados Studio project by match category.
" +
    "The analysis task must have been run previously. " +
    "Requires the project folder path from studio_list_projects or studio_new_project.",
    {
      project_path: z.string().describe("Full path to the project folder"),
      target_language: z.string().optional().describe("Limit to one target language (e.g.
de-DE)"),
    },
    async (params) => {
      try {
        const script = `
          $project = Get-Project -projectDestinationPath ${psPath(params.project_path)}

          if ($null -eq $project) {
            throw "No project found at: ${psPath(params.project_path)}"
          }

          $stats = $project.GetProjectStatistics()
          $results = $stats.TargetLanguageStatistics ${
            params.target_language
              ? `| Where-Object { $_.TargetLanguage.IsoAbbreviation -eq
${psStr(params.target_language)} }`
              : ""
          } | ForEach-Object {
            [PSCustomObject]@{
              language = $_.TargetLanguage.DisplayName
              code     = $_.TargetLanguage.IsoAbbreviation
              new      = $_.AnalysisStatistics.New.Words
              exact    = $_.AnalysisStatistics.Exact.Words
              inContextExact = $_.AnalysisStatistics.InContextExact.Words
```

```
                repeated = $_.AnalysisStatistics.Repeated.Words
                fuzzy    = @($_.AnalysisStatistics.Fuzzy | ForEach-Object {
                  [PSCustomObject]@{ band = $_.Band; words = $_.Words }
                })
              }
            }

            @{ statistics = @($results) } | ConvertTo-Json -Depth 5 -Compress
          `;
          const result = await studioPs(script);
          return { content: [{ type: "text", text: JSON.stringify(result, null, 2) }] };
        } catch (error: any) {
          return { content: [{ type: "text", text: `Error: ${error.message}` }], isError: true
};
        }
      }
    );
}
```

12. Error Handling

12.1 PowerShell Errors

Both executors wrap toolkit calls in `try/catch`. PowerShell exceptions are serialised to stderr as `{ "error": "...", "detail": "..." }`. The MCP server catches these and returns them as MCP error responses with `isError: true`.

12.2 PS Host Not Found

If the PS5 x86 host or `pwsh.exe` is not found, `execFile` throws `ENOENT`. The error message surfaced to the client names the missing host and the environment variable to use to override the path.

12.3 Toolkit Not Installed

If the module import fails, the PowerShell error is surfaced verbatim. It will indicate that the module was not found in any module path, which is actionable.

12.4 Studio Not Licenced

`Import-ToolkitModules` fails if Studio is not installed or is not licenced for Professional. The Studio API's error message is descriptive enough to diagnose without wrapping.

12.5 GroupShare Authentication Failure

If `SignIn` fails (wrong credentials, server unreachable), the toolkit throws and the error is surfaced to the client. Use `gs_list_credentials` to verify which credential is active and `gs_set_credential` to switch to a different one. If using raw environment variables, restart Claude Desktop after correcting the `GS_*` vars.

12.6 Language Cloud Authentication Failure

If `Get-AccessKey` fails, the error names the step that failed (usually an HTTP 401 from the token endpoint). Use `lc_list_credentials` to verify which credential is active and `lc_set_credential` to switch to a different one. If using raw environment variables, restart Claude Desktop after correcting the `LC_*` vars.

12.7 No Active Credential

If a GS or LC tool is called and no active credential has been selected (store contains multiple XMLs and `gs_set_credential` / `lc_set_credential` has not been called), `buildPreamble` will find both `activeGsCredentialFile` / `activeLcCredentialFile` null and all raw env vars unset. The PowerShell script will fail when attempting to read `$serverUrl` / `$clientId`. The error message should prompt the user to call `gs_set_credential` / `lc_set_credential` first.

12.8 Timeout

The timeout is 10 minutes for all executors. Long-running operations (pre-translation on a large project, server TM export) are the most likely to hit this. The error message returned to the client suggests scoping the operation to a single target language to reduce execution time.

---

13. Security Considerations

13.1 Local Only

Identical to other MCP servers in this ecosystem - stdio transport only, no open ports, no network exposure beyond what the toolkits themselves make.

13.2 Script Injection

All string parameters are passed through `psStr()` or `psPath()` before interpolation. Zod validates types before any parameter reaches the script builder. Numeric and boolean values are never interpolated as strings.

13.3 Credential Store Security

DPAPI-encrypted XML files in the credential store can only be decrypted by the Windows user who created them. The store folder path is stored in `claude_desktop_config.json` but the credential files themselves are not readable by other users on the machine. Raw environment variable credentials (GS password, LC client secret) are stored in plaintext in `claude_desktop_config.json` and should only be used for single-user local setups. The credential store approach is strongly preferred.

Decrypted credential values (passwords, client secrets) are never logged or returned in tool output. They appear only in the script body passed to the PowerShell child process, which is not persisted to disk by the executor.

13.4 Destructive Operations

`studio_remove_project` with `delete_files: true` permanently deletes the project folder. `gs_update_project_status` to `Completed` affects assignments and visibility for all project members on the server. `lc_remove_tm`, `lc_remove_customer`, and `lc_remove_project_template` permanently delete resources from Language Cloud with no undo. Claude Desktop's built-in tool permission prompt provides the human-in-the-loop checkpoint before any tool executes.

---

14. Build and Run

14.1 Setup, Build, and Pack

Step 1 - Install dependencies and compile:

```
cd trados-powershell-mcp
npm install
npm run build
```

Step 2 - Pack the desktop extension:

```
mcpb pack . trados-powershell-mcp-1.7.0.mcpb
```

This produces a `.mcpb` file in the project directory. The version number in the filename should match `version` in `manifest.json`.

Step 3 - Install in Claude Desktop:

Go to Settings → Extensions → Advanced settings → Install Extension and select the `.mcpb` file. Claude Desktop will present a configuration form for all fields defined in `user_config`. Fill in the paths and credentials relevant to your environment. Fields for tool groups you are not using can be left blank - the server will simply not register those tools.

14.2 package.json

```
{
  "name": "trados-powershell-mcp",
  "version": "1.7.0",
  "type": "module",
  "scripts": {
    "build": "tsc",
    "start": "node dist/index.js"
  },
  "dependencies": {
    "@modelcontextprotocol/sdk": "^1.12.0",
    "zod": "^3.25.0"
  },
  "devDependencies": {
    "@types/node": "^22.0.0",
    "typescript": "^5.7.0"
```

```
  }
}
```

14.3 tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "Node16",
    "moduleResolution": "Node16",
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true
  },
  "include": ["src/**/*"]
}
```

14.4 Testing Individual Executors

These manual tests use `-Command` for simplicity. The actual Studio executor uses a temp `.ps1` file with UTF-8 BOM instead (see §6.1) to handle non-ASCII paths correctly on ANSI codepage systems. For manual testing with ASCII-only paths, `-Command` works fine.

Test the Studio executor without Claude Desktop:

```
& "C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe" -NonInteractive -NoProfile -
Command "
    Import-Module -Name ToolkitInitializer
    Import-ToolkitModules -StudioVersion 'Studio18'
    Get-Project -projectDestinationPath 'C:\Projects\Test' |
      Select-Object Name, LocalProjectFolder | ConvertTo-Json
"
```

Test the PS7 executor for GroupShare:

```
& "pwsh.exe" -NonInteractive -NoProfile -Command "
    Import-Module -Name AuthenticationHelper -ArgumentList 'https://groupshare.example.com'
    \$token = SignIn -userName 'admin' -password 'password'
    Get-AllProjects -authorizationToken \$token | Select-Object -First 3 | ConvertTo-Json -
Depth 5
"
```

Test the PS7 executor for Language Cloud:

```
& "pwsh.exe" -NonInteractive -NoProfile -Command "
    Import-Module -Name AuthenticationHelper
    \$key = Get-AccessKey -id 'your-client-id' -secret 'your-client-secret' -lcTenant 'your-tenant-id'
    Get-AllProjects -accessKey \$key | Select-Object -First 3 | ConvertTo-Json -Depth 5
"
```

Test the full MCP server:

```
echo '{"jsonrpc":"2.0","id":1,"method":"tools/list"}' | node dist/index.js
```

---

15. Conversation Examples

Discovering available tools:

> User: "What Trados tools do you have available?"
>
> Claude reports the registered tool groups based on what was activated at startup.

Listing and selecting a credential:

> User: "List the Language Cloud tenants I have credentials for."
>
> Claude calls `lc_list_credentials`. Returns filename, tenant ID, and client ID for each XML in the store.

> User: "Use the production tenant."
>
> Claude calls `lc_set_credential` with the matching filename. Confirms the tenant ID and client ID from the selected file.

End-to-end file-based project workflow:

> User: "Create a Studio project called 'Q3 Newsletter' from the files in C:\Source. English to German. Use the marketing TM."
>
> Claude calls `studio_list_tms` to find the marketing TM, then `studio_new_project`.
> Reports the project folder path.

> User: "Analyse it and tell me the new word count."
>
> Claude calls `studio_analyze`. Reports new word count by file.

> User: "Pre-translate it against the TM then export a package."
>
> Claude calls `studio_pretranslate`, then `studio_export_package`. Reports the package path.

GroupShare workflow:

> User: "What projects are currently in progress on GroupShare?"
>
> Claude calls `gs_list_projects` with `statuses: "In Progress"`.

> User: "Get me the analysis report for the Annual Report project."
>
> Claude calls `gs_get_project` to find the project, then `gs_get_analysis_report`.

> User: "The return package from the linguist is at C:\Returns\Q3.sdlrpx. Import it."

> Claude calls `gs_import_package`.

> User: "Mark it as completed."
>
> Claude calls `gs_update_project_status` with `status: "Completed"`.

GroupShare hierarchy browsing:

> User: "Show me the top-level organisations on GroupShare."
>
> Claude calls `gs_list_organizations` with `parent_path: "/"` and `compact: true`. Returns a scannable list of names and paths with a total count showing how many organisations exist on the server overall.

> User: "Drill into the Consoltec organisation."
>
> Claude calls `gs_list_organizations` with `parent_path: "/Consoltec"` and `compact: true`. Returns direct children only.

> User: "How many projects are in that organisation, grouped by status?"
>
> Claude calls `gs_list_projects` with `organization_name: "Consoltec"`, `include_sub_organizations: true`, and `group_by: "status"`. Returns a count summary per status value.

GroupShare organisation report:

> User: "Give me a full summary of everything in the multifarious organisation."
>
> Claude calls `gs_org_report` with `organization_name: "multifarious"`. Returns child organisations, projects with status/language summaries, containers with TMs per container, project templates, and users - all from a single call.

Language Cloud workflow:

> User: "What project templates do we have in Language Cloud?"
>
> Claude calls `lc_list_project_templates`.

> User: "Create a new project called 'Investor Pack' using the Legal template. Due Friday at 17:00."
>
> Claude calls `lc_new_project` with `due_date` and `due_time` as separate parameters.

> User: "Export the Legal TM to TMX so I can back it up."
>
> Claude calls `lc_list_tms` to confirm the TM exists, then `lc_export_tm`.

Language Cloud resource discovery:

> User: "I need to create a TM in Language Cloud but I don't know what language processing rules or field templates are available."
>
> Claude calls `lc_list_language_processing_rules` and `lc_list_field_templates`. Reports available values so the user can choose.

> User: "What locations exist in our Language Cloud account?"
>
> Claude calls `lc_list_locations`. Returns the location hierarchy with IDs and names.

> User: "What language codes does Language Cloud support for Japanese?"

> Claude calls `lc_list_supported_languages`. Filters the results to show Japanese-related codes (e.g. `ja-JP`).

Language Cloud customer management:

> User: "Create a new customer called 'ACME Corp' under the Customers location."
>
> Claude calls `lc_list_locations` to find the Customers location ID, then `lc_new_customer` with the location ID. Reports the new customer ID.

> User: "Set the RAG status to red for ACME Corp."
>
> Claude calls `lc_list_customers` to find the customer ID, then `lc_update_customer` with `rag_status: "red"`.

> User: "Delete the test TM called 'Demo TM' and remove the 'Test Customer'."
>
> Claude calls `lc_list_tms` to find the TM ID, calls `lc_remove_tm`, then calls `lc_list_customers` to find the customer ID, then `lc_remove_customer`.

Cross-system workflow combining all four MCP servers:

> User: "Take the completed German SDLXLIFF from the GroupShare project, check for any empty segments, fix any terminology issues with 'utilise' → 'use', then import it back."
>
> Claude uses `gs_get_project` to find the file path, switches to `refiner_list_segments` (SDLXLIFF Refiner MCP) to inspect content, runs `refiner_replace` for the terminology fix, then uses `gs_import_package` to return the corrected file.

---

16. Limitations

1. **Studio group: Windows and Professional licence only.** The Project Automation API requires Windows and Trados Studio Professional. This is a known, accepted constraint.
2. **Studio group: PS5 x86 required.** The toolkit's DLLs are 32-bit. The PS5 x86 host must be present or configured via `STUDIO_PS_PATH`.
3. **GroupShare and LC groups: PS7 required.** Both toolkits require PowerShell 7. `pwsh.exe` must be installed and on PATH, or configured via `PS7_PATH`.
4. **No streaming progress.** Output is captured after the PowerShell process exits. There is no progress indication during long-running operations.
5. **Object projection required.** The Studio API's .NET object graph requires `Select-Object` projection before serialisation to avoid circular reference failures. Each tool must implement the appropriate projection.
6. **PS5 and PS7 cannot share a process.** The two executors are genuinely separate processes. There is no way to reuse a PS5 session for Studio calls and a PS7 session for GS/LC calls within a single MCP request. Each tool invocation starts a fresh PowerShell process.
7. **GS toolkit exports to `.tmx.gz`.** The GroupShare `Export-TMX` function appends `.tmx.gz` to the output path if not already present. The `gs_export_tmx` tool's `output_path` parameter must end in `.tmx.gz`. This is enforced by the toolkit, not the server.
8. **LC toolkit: no package workflow.** The Language Cloud API does not expose a package import/export

workflow equivalent to the file-based and GroupShare toolkits. LC projects are managed entirely through the API and web UI.

9. **Active credential is session-scoped.** The credential selected via `gs_set_credential` or `lc_set_credential` is held in process memory only. It is lost when Claude Desktop is restarted. If the credential store contains exactly one XML per toolkit, that file is auto-selected at startup and no manual selection is needed. Users with multiple credentials in the store must call the set-credential tool at the start of each session.
10. **Studio group is always registered.** Unlike the GS and LC groups, Studio tools are registered unconditionally because there is no credential to check at startup. If Studio is not installed or not licenced, Studio tool calls will fail at execution time rather than being silently absent from the tool list.
11. **GS PSObject lookup overhead.** Because GroupShare toolkit functions take PSObjects rather than ID strings, every tool invocation that acts on an existing resource incurs at least one additional REST call to look up the object. This is unavoidable given the toolkit's design and is fast in practice.
12. **LC `lc_new_tm` mandatory resource parameters.** `New-TranslationMemory` requires both a language processing rule name/ID and a field template name/ID. These cannot be omitted. Use `lc_list_language_processing_rules` and `lc_list_field_templates` to discover available values, or use `lc_list_tms` to see what values are in use on existing TMs.
13. **GS list tools fetch all then filter client-side.** `gs_list_organizations`, `gs_list_tms`, `gs_list_project_templates`, and `gs_list_containers` call `Get-All*` to retrieve the full dataset, then apply `name_filter`, `parent_path`, and `max_results` via PowerShell pipeline. On very large servers this means the full dataset is loaded into memory even when only a small subset is needed. The `total` / `matchingCount` / `returnedCount` metadata helps the user understand the scope.
14. **GS toolkit stdout warnings.** Some GroupShare toolkit functions write non-JSON warnings or error messages to stdout via `Write-Host` before the expected JSON output. The `safeParseJson` function in `common.ts` handles this by extracting the last top-level JSON object from stdout and attaching any prefix text as a `_warnings` property. This means tool results may include a `_warnings` field containing raw toolkit error text alongside valid data.
15. **GS `gs_org_report` loads global datasets.** The org report tool calls `Get-AllContainers`, `Get-AllTMs`, and `Get-AllProjectTemplates` to retrieve server-wide datasets, then filters locally by organisation. On servers with very large numbers of these resources, this may be slow or hit memory limits. The containers-to-TMs join is particularly expensive as it filters `Get-AllTMs` results by `ContainerId` for each container owned by the target organisation.
16. **LC customer location propagation delay.** When `lc_new_customer` creates a customer, the corresponding child location may take several seconds to appear in `lc_list_locations`. Scripts that create a customer and immediately need its location ID should allow for a brief delay before querying locations.
17. **LC destructive operations require empty resources.** `lc_remove_customer` fails if projects, TMs, termbases, or project templates are still associated with the customer's location. Child customers must be removed before their parent. Resources must be deleted in the correct order: TMs and termbases first, then project templates, then child customers, then the parent customer.
18. **Studio executor: non-ASCII path support.** The Studio executor writes scripts to a temp `.ps1` file with a UTF-8 BOM because PS5's `-Command` argument goes through Windows argument parsing, which mangles non-ASCII characters on systems with an ANSI codepage. This adds a small I/O overhead per

tool call (write + delete temp file) but is necessary for correct operation on non-English Windows installations.

19. **LC locale sensitivity.** The Language Cloud toolkit's `Get-AccessKey` parses date strings from the OAuth token response using US date format assumptions. On non-US locale systems the parse fails. The server works around this by temporarily switching to `InvariantCulture` during the `Get-AccessKey` call.
20. **Studio version naming trap.** `Studio18` is the AppData/registry key used for paths like `TranslationMemoryRepository.xml`. `Studio 2024` is the Documents display name used for the Projects folder. Mixing these causes silent failures. The `STUDIO_VERSION` env var must use the AppData key form.