

PULSE / Revit

Guide and commands

Autodesk Revit 2023-2027

Installation, first steps and natural-language examples. Technical command names remain unchanged.

Verified source: <https://github.com/LuDattilo/pulse-for-revit>

Branch: main | Commit: 2c0407f58c5266eb9466a4bdd2553a05f7da30b1

2026-10-04 | EN

Descriptions in English. Command names, parameters and technical values are unchanged. Source catalogue dated 4 October 2026; translations updated on 8 October 2026.

<https://pulse-ai.it/guida.html>

Installation and connection

Close Revit, fully extract the package and run the EXE installer delivered by the website. Restart Revit, open a project and switch PULSE on from its tab. Use the AI client configuration supplied with the package.

Connect an MCP-compatible AI client: Claude, Codex, opencode or another client, including local models.

Open the project and describe the goal. Check the result and approve changes when requested.

Work on a project copy. Request a read operation or preview before changes, check the selection and approve only the intended operations.

On the first eligible registration, a personal 14-day Premium Trial. Another request does not renew an existing trial.

Install PULSE, connect your AI assistant (Claude, Codex, opencode or any other MCP client - including local open-weight models), and talk to your Revit model in plain language. This documentation covers installation, setup, the full command catalogue (Free and Premium), and practical examples.

1. Installation

PULSE is a plugin for Autodesk® Revit® 2023-2027. Installation takes a few minutes and needs no technical skills.

Download PULSE Free from the site: fill in the form on pulse-ai.it and get a download link by email (valid 48 hours). The ZIP contains Pulse-Setup.exe and the guide.

Close Revit and Claude Desktop : the installer must be able to replace plugin files and update the MCP configuration.

Fully extract the ZIP , run Pulse-Setup.exe , and approve the UAC prompt. The installer auto-detects your installed Revit versions (2023-2027) and installs the plugin for each.

Restart Revit. A PULSE ribbon tab appears, with "Pulse Switch" (starts/stops the server, off at launch) and "Settings".

Start the server: open a project, click Pulse Switch to turn on the local server. From now on your AI assistant can operate on the model.

MCP CONFIGURATION

Your AI development assistant cannot see PULSE?

PULSE is already installed correctly. On some PCs, the installer may not finish the automatic setup for your AI development assistant - Claude Code, Codex, opencode, or Cursor - because paths, permissions, versions, and MCP configuration files vary.

Paste the matching prompt into the assistant you use. It completes only its own MCP configuration ; it must not reinstall PULSE or modify the Revit model.

Claude Code

Configure PULSE as an MCP server for Claude Code on this Windows PC. PULSE is already installed: do not reinstall it and do not modify any Revit model.

Confirm that `%USERPROFILE%\pulse\server\Pulse.Server.exe` exists. If it does, register or update the MCP server named pulse-revit with Claude Code's native command, preserving all other servers. Restart Claude Code if needed and verify that pulse-revit is available. Report the files checked, changes made, and any blocker.

Codex

Configure PULSE as an MCP server for Codex on this Windows PC. PULSE is already installed: do not reinstall it and do not modify any Revit model.

Confirm that `%USERPROFILE%\pulse\server\Pulse.Server.exe` exists. Use the native Codex command when available; otherwise update `%USERPROFILE%\codex\config.toml`. Create a backup first, preserve all other servers, and repair duplicate `[mcp_servers.pulse-revit]` blocks so exactly one remains. Verify that Codex can see pulse-revit and summarize the result.

opencode

Configure PULSE as an MCP server for opencode on this Windows PC. PULSE is already installed: do not reinstall it and do not modify any Revit model.

Confirm that `%USERPROFILE%\pulse\server\Pulse.Server.exe` exists. Identify the opencode configuration file actually in use (global or project), create a backup, and add or update one local MCP entry named pulse-revit without removing other configuration. Restart opencode if needed, verify the connection, and report the result.

Cursor

Configure PULSE as an MCP server for Cursor on this Windows PC. PULSE is already installed: do not reinstall it and do not modify any Revit model.

Confirm that `%USERPROFILE%\pulse\server\Pulse.Server.exe` exists. Identify the MCP configuration actually used by Cursor, create a backup, and add or update the local server named pulse-revit without removing other servers. Do not duplicate the entry. Restart Cursor if needed, verify that PULSE is available, and report the files checked, changes made, and any blocker.

2. Connect the AI assistant

PULSE exposes an MCP (Model Context Protocol) server: a standard bridge connecting Revit to the AI assistant of your choice. The installer auto-configures Claude Desktop and Codex when detected; Claude Code and other clients are registered with the commands below. Always replace `<YOUR_USER>` with your Windows user name.

Claude Desktop

Open Settings -> Developer -> Edit Config : this safely opens the file actually read by the app. Classic installations use `%APPDATA%\Claude\claude_desktop_config.json` ; some MSIX installations use `%LOCALAPPDATA%\Packages\Claude_*\LocalCache\Roaming\Claude\claude_desktop_config.json` instead. The installer detects and updates both paths while preserving preferences and other MCP servers.

```
{
  "mcpServers": {
    "pulse-revit": {
      "command": "C:\\Users\\<YOUR_USER>\\.pulse\\server\\Pulse.Server.exe"
    }
  }
}
```

Restart Claude Desktop after the change. Local MCP servers are available in Claude Desktop chats, not in the web app or Cowork.

For IT managers

Claude Code

From a terminal, one command:

```
claude mcp add pulse-revit -- "C:\\Users\\<YOUR_USER>\\.pulse\\server\\Pulse.Server.exe"
```

Codex (OpenAI) - ChatGPT desktop app or CLI

Codex is built into the ChatGPT desktop app (Windows and macOS): the Codex tab, the Codex CLI and the IDE extension share the same MCP configuration and support local servers. Add to %USERPROFILE%.codex\\config.toml :

```
[mcp_servers.pulse-revit]
command = "C:\\Users\\<YOUR_USER>\\.pulse\\server\\Pulse.Server.exe"
```

Alternatively, from a terminal: `codex mcp add pulse-revit -- "C:\\Users\\<YOUR_USER>\\.pulse\\server\\Pulse.Server.exe"`
. The web version (chatgpt.com) does not support local MCP servers.

opencode

A free, open-source client that also works with local open-weight models (Ollama, LM Studio or any OpenAI-compatible endpoint): this way PULSE runs with no cloud at all. Add to %USERPROFILE%.config\\opencode\\opencode.json (or to the project's opencode.json):

```
{
  "mcp": {
    "pulse-revit": {
      "type": "local",
      "command": ["C:\\Users\\<YOUR_USER>\\.pulse\\server\\Pulse.Server.exe"],
      "enabled": true
    }
  }
}
```

Other clients

Cursor (%USERPROFILE%.cursor\\mcp.json), Cline , Continue and Gemini CLI (%USERPROFILE%.gemini\\settings.json) use the same mcpServers block shown for Claude Desktop. In general: any desktop MCP client with stdio transport works with PULSE.

Connection checklist

Revit open with an active document;

Pulse Switch on (amber icon);

the AI assistant restarted after editing the config.

3. First steps

With the server on, tell the assistant what you want as you would tell a colleague. Start with a read-only command, risk-free: " Run a model health check " or " Show me all the walls ". PULSE turns the request into verified operations on the model and replies with the results.

5. Practical examples

Real natural-language requests. The read-only ones already work in Free; the write ones need Premium.

6. Model language

PULSE auto-detects the Revit language (Italian, English, French, German) and uses the correct parameter and category names. You can write to the assistant in your own language: PULSE handles the translation into Revit operations.

Command catalog

Descriptions in English. Command names, parameters and technical values are unchanged. Source catalogue dated 4 October 2026; translations updated on 8 October 2026.

Cad

`get_cad_geometry`

Read nested geometry of a CAD import/link: curves, polylines, meshes, solid faces and points in Revit model mm. Summary by default; detail returns vertices and triangle indices, plus analytic arcs/planes when available. Filters use sampled bounds, not exact clipping; uniform curve samples do not guarantee chord error. Starts with coarse geometry; increase detailLevel/curveSamples when needed. The 10-second scan budget and cancellation are cooperative between native calls. A native geometry or triangulation call cannot be interrupted; maxVertices limits output, not that call's cost. Check scanComplete, truncated and warnings. Solid primitives are individual faces; paths are scan-local, not persistent DWG handles. Read-only; no text extraction or BIM classification.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CadTools.cs

Computo Document

`export_computo_pdf`

Renders an already-materialized .xpwe computo (or price list) to a PDF delivery document. Runs entirely server-side and does NOT require Revit or the plugin to be running - reads the .xpwe from disk, checks the Pulse license state locally, and writes the PDF. Preview-first (dryRun defaults true): call once to see rowCount/sectionsIncluded/sectionsOmitted, then again with dryRun:false to actually write the file. Deliberate exception to the rest of this project: every other Pulse.Server tool ends with revit.ExecuteAsync, a passthrough to the plugin over the bridge - this one never touches the bridge, which is also why it keeps working with Revit closed.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ComputoDocumentTools.cs

`export_computo_word`

Renders an already-materialized .xpwe computo (or price list) to a Word (.docx) delivery document. Runs entirely server-side and does NOT require Revit or the plugin to be running - reads the .xpwe from disk, checks the Pulse license state locally, and writes the .docx. Preview-first (dryRun defaults true): call once to see rowCount/sectionsIncluded/sectionsOmitted, then again with dryRun:false to actually write the file. Same pipeline as export_computo_pdf (ComputoDocumentExporter) - only the renderer and the required .docx extension differ. Deliberate exception to the rest of this project: every other Pulse.Server tool ends with revit.ExecuteAsync, a passthrough to the plugin over the bridge - this one never touches the bridge, which is also why it keeps working with Revit closed.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ComputoDocumentTools.cs

Creation

batch_export

Export views/sheets to DWG, DXF, DGN, PDF, image (PNG), or Navisworks (NWC) formats.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

change_element_type

Change the type of one or more elements to a target type specified by ID or name.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

color_elements

Color the active view's elements of a category by grouping them on a parameter value, or reset (clear) those color overrides. action=color|reset. Operates on the ACTIVE model view (not a sheet).

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

create_array

Create a linear or radial array. Builds a real associative Revit ArrayElement (editable count) unless associative=false. linear uses spacingX/Y/Z; radial uses centerX/Y and totalAngle.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

create_color_fill_scheme

Colors Rooms, Areas, or MEP Spaces by parameter value using Revit's NATIVE ColorFillScheme + ColorFillLegend (an alternative to create_color_legend's manual overrides + hand-drawn legend, which is not a real Revit legend element and remains the only option for other categories). Always duplicates an existing scheme of the target category and sets its ParameterDefinition -- Revit has no from-scratch ColorFillScheme creation API. A view has only one shared color-fill slot for all spatial categories (Rooms/Areas/Zones/Spaces): applying this scheme silently replaces any other spatial category's active scheme on that view unless none is active -- the response reports which schemes would be (or were) replaced. Preview-first: dryRun defaults to true and runs the whole sequence in a sandboxed transaction, then rolls it back.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

create_color_legend

Color elements by parameter value and optionally create a legend view.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

create_dimensions

Create dimension annotations in the active view. Pass a JSON array of dimension specs. Element mode: [{viewId, referencelds:[...], linePoint:{x,y,z}, dimensionStyleId?}]. Point-to-point mode: [{viewId, startPoint:{x,y,z}, endPoint:{x,y,z}, linePoint?, dimensionStyleId?}]. dimensionStyleId is honored in both modes.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

create_filled_region

Create a filled region in a view from a closed boundary, optionally with holes (inner loops).

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

`create_floor`

Create an architectural floor from a boundary (or a room), optionally with holes. Provide boundaryPoints OR roomId.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

`create_grid`

Create named straight grids at any plan angle using specs [{name,startMm:[x,y,z],endMm:[x,y,z]}], or legacy X/Y count + spacing; rename/delete also supported. All coordinates are model mm. Explicit specs require same-Z endpoints, reject duplicate names/coincident infinite XY axes (0.1 mm), and cannot mix with count/spacing. Optional bottomMm/topMm set vertical extents. Explicit specs default to a Revit transaction preview that rolls back; set dryRun=false to commit atomically. Legacy actions retain their apply default when dryRun is omitted; dryRun=true previews them without writes.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

`create_line_based_element`

Create walls, beams or braces. Specs: [{category, locationLine:{p0:{x,y,z},p1:{x,y,z},pMid?:{x,y,z}},typeId?,height?,baseLevel?,baseOffset?,structural?,structuralRole?,startOffsetMm?,endOffsetMm?,crossSectionRotationDegrees?}]. Coordinates/heights in mm. baseLevel is an elevation, NOT an id. structural=true creates a structural wall. Framing structuralRole is Beam (default) or Brace; endpoint offsets are relative to the resolved level and override the common baseOffset. Section rotation is in degrees. No dryRun.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

`create_point_based_element`

Create point-based elements (columns, furniture, sinks, faucets). Pass a JSON array of creation specs: [{category, locationPoint:{x,y,z}, typeId|typeName|familyName, baseLevel, baseOffset?, rotation?, hostWallId?, hostElementId?, faceReference?, referenceDirection?, facingFlipped?, mirror?, params?}]. hostElementId hosts on ANY element (not just walls); faceReference (a stable-representation string, e.g. from get_instance_transform's hostFaceReference or create_reference_plane's faceReference) places a WorkPlaneBased family on a face/work plane instead - mutually exclusive with hostWallId/hostElementId. mirror mirrors the created instance in place. params sets instance parameters on it in the same call. REQUIRED for WorkPlaneBased families (check with get_instance_transform's isWorkplaneBased or familyPlacementType): the call is refused without faceReference, because placing one any other way silently lands it at an arbitrary position/orientation that no later move, rotate, mirror, or set_instance_flip call can correct - it must be created on the right face/plane from the start.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

`create_revision`

List, create, update, or assign revisions to sheets. action=list|create|set|add_to_sheets. 'set' updates an existing revision (needs revisionId).

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

`create_room_separation_lines`

Create Room Separation Lines (OST_RoomSeparationLines) from a closed boundary polygon, so `create_room` can succeed in an open-plan area with no enclosing walls - an atrium, a lobby, or a zone split whose boundary is not a wall. The boundary elevation is derived from the given view's own level, never from a separately passed level, so the two can never disagree. Preview-first: `dryRun` defaults to true and validates the polygon without creating elements. Returns `awaitingUser` and a single-use `planId`. After user approval, repeat the same input with `dryRun:false` and `planId` within five minutes. Changed documents require a new preview. Native confirmation and failure processing occur on apply; preview cannot guarantee acceptance.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

`create_structural_framing_system`

Create a beam system on a level over a rectangular area. Default builds a real associative Revit BeamSystem (editable layout); set `associative=false` for loose independent beams.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

`create_surface_based_element`

Create floors, foundation slabs, ceilings or flat roofs. Specs: [{category, boundary:{outerLoop:[{p0:{x,y,z},p1:{x,y,z}},...]}, typeId?, baseLevel?, baseOffset?, structural?}]. XYZ and elevations/offsets in mm; `baseLevel` is an elevation, NOT an `ElementId`. `structural` applies to floors; `OST_StructuralFoundation` requires a foundation-slab `FloorType`. Roofs are flat; outer loop only. No `dryRun`.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

`create_text_note`

Create text notes in a view. Pass a JSON array: [{text, position:{x,y,z}, viewId?, textNoteTypeId?, width?, horizontalAlignment?, verticalAlignment?, rotation?, leader?}]. rotation is degrees; leader is left|right|leftArc|rightArc.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

`export_elements_data`

Export element data by category as JSON or CSV, with parameter filtering and an optional include filter. Use for simple single-parameter filters; use `ai_element_filter` for complex multi-criteria queries.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

`export_families`

Export loaded families as .rfa files into a target directory.

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

`export_room_data`

Export room data: number, name, level, area, volume and perimeter. (Does not return bounding elements - use `get_room_openings` or `get_elements_in_spatial_volume` for what a room contains.)

Scope: | sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs

`export_schedule`

Export schedule rows as JSON, and optionally write them to a delimited text file by passing `exportPath`.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs`

`export_to_excel`

Export element data from a Revit category to an Excel file.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs`

`import_from_excel`

Import parameter values from an Excel file into Revit elements.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs`

`import_table`

Import a CSV/TSV file as a formatted table in a drafting or legend view.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\CreationTools.cs`

Deferred Write

`modelsnapshot_capture_read`

PULSEplusO Core snapshot capture: reads ProjectInformation PROJECT_STATUS of the open document whose document key equals `enrolled_model_id` and returns a core-modelsnapshot-capture-1.0 frame. Read-only.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\DeferredWriteTools.cs`

`set_project_status_conditional`

PULSEplusO Core deferred write: sets ProjectInformation PROJECT_STATUS in the open document identified by `documentKey` ONLY if its current value equals the signed expected value, at most once per `executionIdentity`. Returns a receipt frame. Pass either `grant+signature+payload` alone, or the other arguments. Not for interactive use: use `set_project_info` to edit project information.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\DeferredWriteTools.cs`

Dynamo

`dynamo_generate_graph`

Generate and save a valid Python-centric Dynamo .dyn graph. Use ONLY when no native Pulse tool covers the task AND the user approved a Dynamo/Python approach. Requires `EnableDynamo=true`.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\DynamoTools.cs`

`dynamo_get_status`

Report Dynamo for Revit status (present, version, CPython3 availability) and whether `EnableDynamo` is set.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\DynamoTools.cs`

dynamo_list_graph_io

List inputs/outputs of a .dyn graph (parses the file, does not run it).

Scope: | sources\revit-official\src\Pulse.Server\Tools\DynamoTools.cs

dynamo_run_graph

Run a Dynamo .dyn graph headless inside Revit. Use ONLY when no native Pulse tool covers the task AND the user approved a Dynamo/Python approach. Requires EnableDynamo=true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\DynamoTools.cs

Element

add_curtain_grid_line

Add a grid line to a curtain grid, subdividing a facade. Preview-first: dryRun defaults to true and the preview REALLY creates the line in a sandboxed transaction, measures the effect, then rolls back. Everything below was measured on a real model before the tool was written. A REFUSAL HAS TWO DISTINCT CAUSES AND BOTH THROW WITH AN EMPTY MESSAGE, so the exception TYPE is the only signal - measured via MCP: ArgumentException = the position is OUTSIDE THE HOST; InvalidOperationException = A GRID LINE ALREADY EXISTS at that position. Both return a structured failure, and the payload carries the type. (An earlier version of this text claimed the collision returned null as the benign outcome 'notCreated'; that was measured to be FALSE - it throws. The 'notCreated' branch is kept because AddGridLine may return null, but no known input reaches it.) ON A REFUSAL THE 'after' COUNTS ARE DECLARED NOT MEASURED rather than reported, because Revit mutates the in-transaction state before throwing: reading them showed panels 28 -> 32 for a call that created nothing. POSITION IS IN MODEL COORDINATES, in mm - measured, not host or normalised space. AND MIND THE ASYMMETRY: adding a line DOES increase the panel count, while add_remove_grid_segment's RemoveSegment does NOT decrease it (it resizes panels instead) - so do not assume the two operations mirror each other. oneSegmentOnly does not create a shorter line: measured, it creates the line with half its segments already skipped.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`add_remove_grid_segment`

Add or remove segments on a curtain grid line - opening and closing a facade's grid. Preview-first: `dryRun` defaults to true and the preview REALLY performs the operation in a sandboxed transaction, measures the effect, then rolls back. TWO THINGS HERE CONTRADICT THE REVIT DOCUMENTATION, both measured on a real model before this tool was written: (1) `RemoveSegment` DOES NOT MERGE PANELS. The remarks promise 'the neighboring two panels will be merged into one panel'; measured, the panel count stays identical (28 -> 28) and so do the panel IDS (0 disappeared, 0 new - checked by diffing `GetPanelIds`, since one deleted plus one created would also give delta 0). What changes is the panel AREA (222.29 -> 222.78): the panels are RESIZED. So the preview reports `panelAreaM2` as the real effect and `panelCount` only for completeness - do NOT expect one panel fewer. (2) THE LOCK DOES NOT PROTECT: `RemoveSegment` succeeds on a grid line with `Lock == true`, no exception. Revit documents `Lock` as a writable property, not a guarantee, so this tool refuses a locked line unless `allowLockedLines: true` is passed deliberately - measured, 4 of 9 lines on the reference host are locked. (3) IT DESTROYS MULLIONS, AND 'add' DOES NOT BRING THEM BACK - measured, and undocumented by Revit: removing ONE segment took the reference grid from 53 mullions to 50, and restoring that same segment with 'add' left it at 50, i.e. the restored span stays bare. Not 1:1 with segments (mullions are shared between adjacent spans, hence -3 for one segment). The preview reports `mullionCount` so this is visible before committing, because the opposite action does not undo it. Also: a segment already in the requested state is a silent NO-OP in the API (documented on `AllSegmentCurves`), so it is reported as skipped with the reason rather than counted as done - while a curve Revit cannot locate THROWS, and the two are kept apart. Segments are addressed by 0-based INDEX into the existing (for remove) or skipped (for add) curves, whose counts `get_curtain_grid` reports per line.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs`

`ai_element_filter`

Find or list elements by category, element class, family symbol, bounding box, or level - walls, doors, windows, columns, floors, roofs, rooms, structural framing, etc. (via `OST_` codes). The primary tool for 'find/list/how many <category>' or 'all the <category> on level X'. Filters combine with AND (default) or OR and can be inverted (NOT); supports type and instance filtering. For parameter-VALUE filtering use `filter_by_parameter_value`.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs`

`change_curtain_panel_type`

Change the type of curtain panels. Preview-first: `dryRun` defaults to true and the preview REALLY performs the change in a sandboxed transaction, reads the resulting type, then rolls back - so `typeAfter` and `changedCount` are MEASURED, not predicted. Four things decide whether a call will work, and all are checked BEFORE the transaction opens so an ambiguity can never leave a half-written facade: (1) A CURTAIN PANEL IS NOT ALWAYS A Panel - `GetPanelIds` returns 'either Panel or Wall Elements', and running `get_curtain_grid` on a real model also found loaded DOOR families in facades (6 of 65 hosts), which report as `FamilyInstance`. Retyping glass, a nested wall and a curtain-wall door are three different operations, so every target is reported for what it actually IS (`elementClass`) and category-incompatible combinations are refused with the reason, because `ChangePanelType` throws `ArgumentException` 'if the input symbol can't be used for the panel'. (2) TYPE NAMES ARE AMBIGUOUS HERE: the measured model contains a panel type literally named 'Wall'. So `targetTypeName` resolves through the `AmbiguousInput` contract - one match resolves, several return the candidates and `ASK`, none returns what IS available. Prefer `targetTypeId`. (3) A panel already of the requested type is reported as rejected with that reason rather than counted as changed. (4) `ChangePanelType` also throws `InvalidOperationException` 'when the type change failed' - unforeseeable, so it is caught PER PANEL: one failure does not abort the batch and is never counted as a success. Capped at 200 panels per call because the method regenerates per panel.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs`

`compare_instance_state`

Reports a family instance's placement state (transform, `FacingFlipped/HandFlipped/Mirrored`, host, bounding box, cut-relationship counts) and, when expected is given, compares it field-by-field with tolerances. Capture with no expected before an operation, then compare the result afterward - works even when the original element was deleted and recreated (e.g. by `rehost_instance`), since the comparison is against a captured snapshot, not a second live element id. Read-only.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs`

`connect_mep_elements`

Connect or disconnect two MEP connectors - the remedy for the open ends `get_mep_connectivity` reports. Identify each side with `elementId` + `connectorId` from `get_mep_connectivity`. Preview-first: `dryRun` defaults to true and the preview RUNS the operation in a sandboxed transaction then rolls it back, so `elementsCreated/elementsRemoved` are MEASURED, not predicted (both Revit methods return void, so a diff is the only way to know what happened). READ fittingCaveat before concluding anything about fittings: `fittingsAddedByRevit:0` does NOT promise Revit will never add one - `ConnectTo` is documented as possibly creating a fitting 'if necessary', and rollback fidelity was measured for a disconnect and a direct reconnect but not for a `ConnectTo` that creates. Three refusals happen BEFORE the transaction, each backed by a documented exception: different domains, already connected, and same connector on both sides. Shape and size differences are deliberately NOT refused - Revit adds a transition fitting 'if necessary', so they are reported as fields (`shapeA/shapeB`, `sizeMmA/sizeMmB`, `gapMm`) and the call is yours. Optional fingerprint: pass `expectedDomainA/B` and `expectedOriginMmA/B` and the tool verifies the connector is the one you mean before writing - worth using, because `Connector.Id` is a POSITIONAL index inside the element (measured), so an id saved from an earlier read can point at a different connector.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs`

copy_elements

Copy elements with optional mm offset. Can target a different view (sourceViewId+targetViewId) or another OPEN document (targetDocumentTitle).

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

create_area

Creates an Area inside an explicit closed boundary polygon (mm), on an existing Area Scheme. Revit has NO API to create a NEW area scheme (it is UI-only, via Area and Volume Computations) -- use an existing one, e.g. "Gross Building Area" or "Rentable Area", which every project ships by default. Unlike a Room, an Area is never auto-detected from enclosing walls: this tool creates the AreaBoundaryLine elements for the given polygon itself, then places the Area at interiorPoint. Areas can only live in an Area Plan view scoped to the same (scheme, level) -- one is reused if it already exists, otherwise created and reported (areaPlanViewWasCreated), since adding a whole new view is a visible side effect, not a silent one. Preview-first: dryRun defaults to true and runs the whole creation in a sandboxed transaction, then rolls it back.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

create_mep_curve

Create a duct, pipe, conduit, or cable tray between two points (coordinates in MILLIMETRES). Uses Revit's two-point overload ONLY, and that is a safety decision, not a limitation to work around: the Connector-based overloads are documented to still create the element when the connection fails, throwing InvalidOperationException and losing the new ElementId - so they can leave an unreachable orphan. This tool cannot. Revit may add fittings of its own ('additional fitting(s) are included to make a valid connection'), so the response reports totalElementsCreated and fittingsAddedByRevit, not just the curve. Minimum length 1/10 inch (~2.54 mm), checked up front. conduit/cabletray have NO system-type concept in the Revit API (unlike duct/pipe) - omit systemTypeId for those two kinds; passing it is simply unused, never an error. Preview-first: dryRun defaults to true and the preview RUNS the creation in a sandboxed transaction, then rolls it back, so the element counts are measured rather than predicted.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

create_path_of_travel

Create Path of Travel elements (egress routes) between pairs of points, in MILLIMETRES, in the ACTIVE view - which must be a plan view. READ calculationStatus before concluding anything: Revit's Create returns nothing for at least four different reasons, and only the status separates 'there really is no route' (NoPathOfTravel) from 'the VIEW is the problem' (PointOutsideActiveCrop, SplitView) and 'the input is' (StartAndEndPointsTooClose). A path returned with ResultAffectedByCrop EXISTS but may not be the shortest route - it is reported as a success with isOptimalityLimited:true, not as a failure. Measured on a real model: 10 of 11 floor plans had an active crop, so that is the normal case. Every result carries its own startIndex/ endIndex and both points, so the pairing never has to be inferred from array order (Revit's own mapped ordering varies the START first, which is the opposite of the intuitive reading). viewId is NOT accepted: activate the plan you want first - whether Create behaves identically on a non-active view is undocumented, so the parameter is refused rather than silently ignored. Preview-first: dryRun defaults to true and really creates the paths in a sandboxed transaction to record their statuses, then rolls back.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`create_reference_plane`

Creates a reference plane from three points (bubbleEnd, freeEnd, thirdPoint). The third point fixes the plane's orientation and must not lie on the bubbleEnd-freeEnd line. Preview-first: dryRun defaults true, but Revit does not document which way the resulting normal points, so the preview cannot show it - the real normal is only returned after creation (dryRun:false). Check it before placing anything on the plane: a reference plane is cheap and reversible (delete_element), a family placed on the wrong-facing one is not. The created plane's faceReference (a stable-representation string) can be passed to create_point_based_element to place a work-plane-based family on it.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`delete_selection`

Delete a saved selection filter by name

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`divide_mep_system`

Split an MEP system whose components form more than one disconnected physical network into one system per network - the remedy for get_mep_systems' isMultipleNetwork:true. HVAC and piping ONLY (documented Revit API limit); electrical is refused with a named error. IMPORTANT, measured not documented: the original system is CONSUMED, so N networks produce N new systems and the model's system count grows by N-1, not N. The preview reports physicalNetworkCount (what the API knows) and netSystemCountDelta (what the model will do) - it deliberately does not promise a creation count the API never guarantees. Not a clean undo: reconnecting the elements later merges the systems back but leaves a suffixed name like 'X (1)'. Preview-first: dryRun defaults to true; re-call with dryRun:false to apply.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`find_undimensioned_elements`

Find elements not referenced by dimensions

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`find_untagged_elements`

Find elements without tags in a view

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`get_cad_lines`

Read the lines of a DWG import or link. Default: a summary per layer+colour. Pass layer and/or color to get the individual curves (endpoints in mm, model coordinates), each flagged constructible (straight and long enough for an MEP duct/pipe). Read-only - the first step of turning DWG lines into MEP. Omit importInstancelId to auto-pick the single import, or to list candidates when several exist.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

get_curtain_grid

Report the curtain grid(s) of curtain walls, curtain systems and curtain roofs: U/V grid lines with their segment states, panels, and mullions. Read-only, active document only. Four API contracts shape the answer, and each one is a trap if ignored: (1) A CURTAIN PANEL MAY BE A WALL - GetPanelIds is documented to return 'either Panel or Wall Elements', and on a real measured model 41 of 681 panels (6%) are walls, i.e. facades nested inside facades. So the total lives INSIDE panels.byElementClass / panels.byFamily rather than as a bare panelCount beside them: reading only the first field must not let you conclude '681 Panel elements' when reality is 464 glazing + 152 photovoltaic + 41 walls, which are three different operations. (2) A CURTAIN SYSTEM HAS ONE GRID PER FACE - CurtainSystem.CurtainGrids is plural ('each CurtainGrid corresponds to one face'), so the result is always host -> grids[] and a wall simply contributes one. Never assume one grid per element. (3) SEGMENTS HAVE THREE STATES, not two: allSegmentCount / existingSegmentCount / skippedSegmentCount, because which state a segment is in decides what you may do to it (AddSegment restores a skipped one; RemoveSegment/AddMullions act on an existing one). (4) CELLS ARE NOT COMPUTED BY DEFAULT and the omission is DECLARED in the payload as cells.status='omitted_for_cost': GetCurtainCells returns CurveLoops geometry whose cost the Revit API does not document. Pass verbose=true to compute it. Anything this tool could not read says 'not_measured' rather than reporting a zero it never measured.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

get_element_parameters

Get all parameters of specific elements by their Revit element IDs.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

get_element_solid_geometry

Get an element's REAL solid geometry (bounding box, centroid, volume m3, face/edge counts AND inferred cross-section shape: circular/rectangular/complex) in mm and model coordinates. Unlike get_BoundingBox this reflects the actual solid AFTER joins and cuts, and reports the section SHAPE - essential for rebar: a 613x613 bbox can be a Ø610 circular pile where corner bars/rectangular ties fall outside. Always use this, not the bounding box, to position rebar inside a host.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

get_elements_in_spatial_volume

Find elements within a 3D bounding box or room volume. volumeType=room uses volumelds; volumeType=custom uses customMinX..customMaxZ.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

get_instance_transform

Reads family instance placement geometry no other tool exposes: the placement transform (basis X/Y/Z + origin), FacingFlipped/HandFlipped/Mirrored, the host element (id, category, and hostFamilyName when the host is itself a family instance), levelId, familyPlacementType and isWorkplaneBased (the authoritative check before recreating this instance - WorkPlaneBased families MUST go through create_point_based_element's faceReference, or the call is refused), whether the instance sits on a face/work plane right now (with a reusable faceReference string), and the CanFlipFacing/CanFlipHand/CanRotate capability flags. Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`get_linked_elements`

Query elements from linked Revit models with optional filtering. `parameterNames` is additive - without it only basic fields are returned.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs`

`get_mep_connectivity`

Report the MEP connectors of specific elements (or a whole connector-bearing category): connector type, domain, location in mm, whether each is physically connected, and the referenced elements. Read-only, active document only. This is the per-connector detail behind `get_mep_systems'` `unusedPhysicalConnectorCandidates` count - use it to find WHICH connector on WHICH element is open. Two contracts shape the answer, both worth trusting over intuition: (1) `Connector.AllRefs` is documented to include BOTH physical and logical connections, so references are returned split into `physicalReferences` / `logicalReferences` / `unclassifiedReferences` (classified per pair with `IsConnectedTo`) and must NOT be summed to mean 'connected' - on an electrical circuit the logical set is membership, not a duct joint. (2) `isOpenPerPlumbingApi` comes from `PlumbingUtils.HasOpenConnector`, which the Revit API provides for PIPING ONLY (pipe curve, fitting, accessory); it is null elsewhere rather than false, because for HVAC and electrical no such authority exists and the answer there is the connector scan. `observedConnectorTypes` lists the `ConnectorType` values actually present: no equality filter is applied, since the enum values are composite (`Physical=19=End|Curve|Surface`) without `[Flags]`, so filtering by `==` silently excludes real connectors.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs`

`get_mep_systems`

List MEP systems (HVAC, piping, electrical) with domain, type, base equipment, terminal count and connectivity state. Read-only. ACTIVE DOCUMENT ONLY: systems inside linked models are NOT included, so 0 systems on an architectural host model whose MEP lives in a link is expected, not a clean result. `terminalElementCount` counts terminal elements and excludes the base equipment/panel (Revit API contract). Two DISTINCT fields, do not conflate them: `productType` is the Revit product of this session (`mep/revit/architecture/structure/lt/unknown`, from `Application.Product`) and `productReportsMep` says whether that product reports MEP data at all; `hasMepContent` says whether THIS MODEL has any MEP systems or elements. An architectural model on full Revit is `productReportsMep:true` + `hasMepContent:false` - a real empty result, not a blind one.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs`

`get_paths_of_travel`

List the Path of Travel elements (egress routes) of the active document: start/end/midpoint in mm, the length summed from their curves, waypoint count, the owning view and whether that view's crop is active. Read-only, active document only. Two things worth knowing before reading the numbers: `curveLengthMm` is summed from `GetCurves()` because the API exposes NO length property on the element (31 properties, verified on the shipped DLL) - so it is geometric length, not a Revit parameter that might round differently; and a path was computed INSIDE its owning view, so an active crop there can make the route non-optimal, which is why `ownerViewCropBoxActive` is reported. An empty result carries `emptyResultNote`: paths are view-specific and created explicitly, so a complete model can legitimately contain none.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs`

`get_room_openings`

Get doors/windows adjacent to rooms with dimensions. Filter by roomIds, roomNumbers, or levelName. Use `summaryOnly` for per-room counts first.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`get_selected_elements`

Get currently selected elements in Revit.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`import_from_powerbi`

Reads a previously-exported (or hand-edited) Power BI CSV and writes parameter values back to Revit elements. Accepts both ','-delimited (`push_to_powerbi`) and ';'-'delimited (`export_elements_data`) CSVs - the separator is auto-detected from the header line. Identifies elements via the `ElementId` column. Built-in fields and read-only parameters are skipped. Defaults to `dryRun=true` so callers can preview first.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`inspect_selection`

Reads the current Revit selection. Exactly one element selected returns full instance and type parameters. More than one, or none, returns a compact id/name/category summary with no parameter detail. Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`load_selection`

List saved selections, or load one by name. Omit name to list; pass name to load that selection (and select it in the active view unless `selectInView=false`).

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`match_element_properties`

Copy parameter values from one source element to one or more target elements. Overwrites the targets' existing values, so this defaults to a preview showing each `currentValue -> newValue`: pass `dryRun=false` to write.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`measure_between_elements`

Measure distance between two elements or two points in mm. Provide either `elementId1/elementId2`, or `point1/point2` (as JSON arrays `[x,y,z]`).

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`modify_element`

Move, rotate, mirror, or copy elements. Vectors are `{"x":mm,"y":mm,"z":mm}` JSON objects. move needs translation; rotate needs `rotationCenter` + `rotationAngle` (degrees; `rotationAxis` defaults to Z); mirror needs `mirrorPlaneOrigin` + `mirrorPlaneNormal`; copy needs `copyOffset`.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

operate_element

Select, highlight, isolate, hide, or zoom to elements. Actions: select, selectionbox, setcolor, settransparency, hide, temphide, isolate, unhide, resetisolate, delete. PREVIEW-FIRST ON 'delete' ONLY: it is the single irreversible action here (it calls doc.Delete), so it defaults to dryRun and reports what it WOULD delete plus any ids that do not exist - pass dryRun:false to actually delete, and Revit will then ask for confirmation. The other nine actions ignore dryRun: seven are graphic overrides undone in one click and two only change the UI selection, so gating them would be noise. Note on the preview's count: Revit may delete MORE than the listed ids, because deleting a host also deletes what it hosts.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

pbi_check_auth

Reports the user's Power BI sign-in state. With signIn=true, starts an MSAL device-code flow inside Revit (a TaskDialog will show the URL + code; the user finishes login in their browser). Token is cached DPAPI-encrypted in %LOCALAPPDATA%\pulse\msal_cache.bin.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

pbi_create_dataset

Creates an EMPTY Pulse v2 push dataset: one flat table 'Data' with the given columns (default: the 9 base element columns). Stops with awaitingUser if the name exists. Usually unnecessary - pbi_publish_* with mode:create creates and fills the dataset. dryRun defaults to true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

pbi_get_binding

Returns the Power BI ProjectBinding stored for the active Revit document (workspaceId, datasetId, datasetName, docKey, updatedAt). Returns bound=false with a tip if no binding exists yet. Read-only - call this to verify which workspace/dataset a document is linked to before publishing.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

pbi_list_datasets

Lists push datasets in a Power BI workspace. Useful for finding an existing Pulse dataset before publishing. Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

pbi_list_workspaces

Lists Power BI workspaces (groups) accessible to the signed-in user. Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`pbi_publish_elements`

Publishes Revit elements of the given categories to a Power BI push dataset with ONE flat table 'Data' (ElementId, UniqueId, Category, Family, Type, DocumentTitle, DocumentPath, EpisodeId, ProjectId + one typed column per chosen parameter, project display units, null when empty). Asks (awaitingUser with candidates) when categoryFilter, parameterNames or mode are missing; never guesses. mode: create (new dataset, stops on name conflict) | update (existing dataset: columns must match, else stops with the diff). dryRun defaults to true: preview columns/rows without writing.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`pbi_publish_schedules`

Publishes Revit schedules to Power BI: ONE push dataset per schedule, ONE flat table 'Data' (ElementId, UniqueId, DocumentTitle, DocumentPath, EpisodeId, ProjectId + the schedule's visible fields, one row per element). Asks (awaitingUser) when scheduleIds or mode are missing. Calculated/combined fields without a parameter are omitted with a warning; key schedules are refused. mode: create | update. dryRun defaults to true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`pbi_publish_selection`

Publishes the current Revit selection to a Power BI push dataset with ONE flat table 'Data' (same columns as `pbi_publish_elements`; Category tells the rows apart). One dataset per document ('{Project}_Selezione'), emptied and rewritten on every publish. Asks (awaitingUser) when nothing is selected or when parameterNames/mode are missing. mode: create | update. dryRun defaults to true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`pbi_query`

Executes a DAX query against the bound Power BI dataset and selects matching elements in Revit. Use template params (category, level, parameterName+parameterValue, exportRunId) for common filters, or a raw daxQuery for advanced queries. action=isolate hides other elements instead of selecting. workspaceId/datasetId can be omitted when a ProjectBinding exists.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`pbi_sign_out`

Signs out of Power BI by revoking all cached MSAL tokens. After this call `pbi_check_auth(signIn=false)` returns `signedIn=false`. The previously signed-in account is reported in the response. Use `pbi_check_auth(signIn=true)` to re-authenticate.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

`pbi_trigger_refresh`

Triggers an on-demand refresh of an Import-mode Power BI dataset after Pulse updates its OneDrive/CSV source. Premium only. Requires Power BI sign-in plus workspace and dataset GUIDs.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

powerbi_visual_action

Executes one Pulse Power BI visual action after powerbi_visual_preflight and required user confirmations. Actions: select | isolate | color | reset | createView. Always preflight first.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

powerbi_visual_preflight

Read-only action-specific readiness check for a command copied from the Pulse Power BI visual. Always call this before powerbi_visual_action.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

push_table_to_powerbi

Writes an arbitrary table (headers + rows) to a CSV in the Pulse/OneDrive folder for Power BI. Use when the data was computed by you (LLM analysis, multi-doc aggregation, etc.) and is NOT a direct dump of Revit elements. Pass headers as a JSON array of strings; pass rows as a JSON array where each item is either an array (positional) or an object keyed by header. The file lives next to push_to_powerbi outputs and can be picked up by the same Power BI folder source.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

push_to_powerbi

Export element parameters or existing schedules to CSV in OneDrive\Pulse\<DocumentName> for Power BI refresh. categories+parameterNames -> one elements.csv; scheduleIds -> one CSV per schedule. Scope can be limited to active view or selection.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

rehost_instance

Reassigns a point-based family instance to a new host element. FamilyInstance.Host has no setter in the Revit API, so this deletes the instance and recreates it on the new host at the same location, preserving type/rotation/FacingFlipped/HandFlipped - but NOT instance parameters, which are lost with the deleted instance (reapply with set_element_parameters if needed). Point-based instances only. Preview-first: dryRun defaults true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

renumber_elements

Renumber rooms/doors/windows by location or name. Writes into the specified parameter; supports prefix/suffix and start/increment.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

save_selection

Save element selection as named filter. Overwriting an existing selection set replaces its membership, so this defaults to a preview: pass dryRun=false to write.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

section_box_from_selection

Create a 3D section box from selected elements

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

select_from_powerbi

Selects/zooms/isolates elements in the active Revit view from a Power BI drillthrough. Action: select | highlight | isolate.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

set_element_phase

Assign created/demolished phase to elements. Pass a JSON array of requests: [{elementId, phaseCreatedId?, phaseDemolishedId?}].

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

set_element_workset

Move elements to a different workset. Pass a JSON array of requests: [{elementId, worksetName}]. Worksets are resolved by name only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

set_instance_flip

Sets FacingFlipped and/or HandFlipped to an exact state on already-placed family instances. flipFacing()/flipHand() are toggles with no direct setter in the Revit API, so this reads the current state first and only toggles when it disagrees with the request. Different from modify_element's mirror action: mirroring transforms geometry (Mirrored=true) but does not set these two flags. Preview-first: dryRun defaults true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ElementTools.cs

Family

family_abort

Close family documents WITHOUT saving: the one under authoring, or every family document in the session with all=true. The only way to close a family created or opened by an API call - such a document has no window, so the Window menu cannot reach it. Reports what it closed, what had unsaved changes, and what remains open. Preview-first.

Scope: | sources\revit-official\src\Pulse.Server\Tools\FamilyTools.cs

family_artifact

Size and sha256 of a saved .rfa, read once the file has stopped changing - the verified identity of an artifact. family_save cannot report it: Revit finishes writing after the save call returns, so it publishes sha256_observed instead. Read-only, needs no document.

Scope: | sources\revit-official\src\Pulse.Server\Tools\FamilyTools.cs

family_capabilities

What the family-authoring module can do on this Revit: version and build, which family documents Revit has open (all of them, including the ones with no window) and which is under authoring, where the installed .rft templates live (first N), and the capability of every authoring feature - including system and in-place families, which the API does not expose. Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\FamilyTools.cs

family_constraints_apply

Label the distance between two named reference planes with a family parameter: the parametric linkage. The parameter then DRIVES the distance, so the planes move to its current value - reported as planes_after. Only pairs a plan view can see (normals along x or y). Lengths use unit (default mm). Preview-first.

Scope: | sources\revit-official\src\Pulse.Server\Tools\FamilyTools.cs

family_create

Create a family document from an installed .rft template and make it the family under authoring. templatePath is required; family_capabilities lists the installed templates. Optional: category overrides the template's own, name sets the family name. The template's category and placement are reported before and after. Nothing is written to disk until family_save. Replaces any family already under authoring (reported as a warning).

Scope: | sources\revit-official\src\Pulse.Server\Tools\FamilyTools.cs

family_geometry_apply

Extrude a rectangle whose four edges are aligned to four named reference planes (x_min, x_max, y_min, y_max) on a sketch plane at sketch_z, with the depth DRIVEN by a Length family parameter: the extrusion's own edges are aligned to the planes, so it follows them when a type changes. Optional material_parameter ties the solid's material to a Material parameter. One extrusion per family. Lengths use unit (default mm). Preview-first.

Scope: | sources\revit-official\src\Pulse.Server\Tools\FamilyTools.cs

family_inspect

Inspect the family under authoring: title, category, placement type, parameters (name, kind, instance or type, formula), types and their values, an element inventory, and the Revit warnings. Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\FamilyTools.cs

family_open

Open an existing .rfa as the family under authoring. An already-open file is adopted, not reopened. Opening writes nothing. Preview-first.

Scope: | sources\revit-official\src\Pulse.Server\Tools\FamilyTools.cs

family_parameters_apply

Declare family parameters: name, kind (Length, Number, Text, Integer, YesNo, Material, Angle, Area, Volume), group, instance or type, optional formula. Idempotent; a differing existing kind is refused, not coerced. Values are per type: use family_types_apply. Preview-first.

Scope: | sources\revit-official\src\Pulse.Server\Tools\FamilyTools.cs

family_references_apply

Declare reference planes by name as (axis, offset) - the skeleton a parametric family is dimensioned to. Idempotent by name: an existing plane is reported, never moved, because a labelled plane's position is driven by its dimension. Lengths use unit (default mm). Preview-first.

Scope: | sources\revit-official\src\Pulse.Server\Tools\FamilyTools.cs

family_save

Save the family under authoring to a .rfa inside the user's allowed directories (or a network share). overwrite (default false) replaces an existing file and the user must confirm it. Preview-first. Reports size and sha256.

Scope: | sources\revit-official\src\Pulse.Server\Tools\FamilyTools.cs

family_tests_run

Flex test: for each type, select it, regenerate and MEASURE the solid's bounding box against what that type should be (x, y, z). Proves the geometry follows the parameters - the alignments family_geometry_apply records cannot be proven without moving something. Changes nothing: the transaction is rolled back. Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\FamilyTools.cs

family_types_apply

Create family types and set values: [{name, values:{parameter: value}}]. Lengths use unit (default mm), angles are degrees. Unknown parameter names are refused by name. Existing types are reused, not duplicated. Preview-first.

Scope: | sources\revit-official\src\Pulse.Server\Tools\FamilyTools.cs

Geometry Relationship

get_element_attachments

Read a wall/column's top and base attachment targets; columns also return cutStyle, justification and offsetMm. Walls require Revit 2025.2+. Read-only; does not infer attachments from matching elevations.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GeometryRelationshipTools.cs

get_geometry_join_state

Read join status, cutting element in the join, and directional steel coping for an explicit pair. Unsupported coping returns null, not false. Solid/void cuts are independent: use check_steel_cut_eligibility, get_steel_cut_data or get_instance_void_cut_relationships for those generic Revit cut APIs.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GeometryRelationshipTools.cs

set_element_attachment

Attach/detach one wall or column end to an explicit target in the active project. Wall API requires Revit 2025.2+. Default dryRun performs read-only validation and returns awaitingUser with a single-use planId. After asking the user, apply the same inputs with dryRun:false and planId within five minutes; a changed document requires a new preview. Native confirmation and failure processing occur on apply. Never silently replaces an existing column attachment; detach its explicit target first. Repeated same-state requests are no-ops. Column settings must be omitted for walls and detach. Inspect suppressed warning count and actual solid after apply.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GeometryRelationshipTools.cs

set_geometry_join

Join/unjoin geometry, set desired join cutting order, or apply/remove coping. One explicit pair. Coping: elementId is the steel beam being cut; targetId its steel beam/column cutter. Default preview performs read-only validation and returns awaitingUser with a single-use planId. After asking the user, apply the same inputs with dryRun:false and planId within five minutes; a changed document requires a new preview. Native confirmation and failure processing occur on apply. set_order requires cuttingElementId and never toggles blindly; repeated desired state is a no-op. This is not solid-solid cut or instance-void cut: use the existing add/remove_steel_solid_cut and add/remove_steel_instance_void_cut tools for those operations. No automatic pairing or material inference.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GeometryRelationshipTools.cs

Group Assembly

create_assembly

Create a Revit assembly from element ids (Revit's Create Assembly). Validates/derives the naming category and checks members with the API's own gates. Preview-first: dryRun defaults to true; dryRun=false creates. A mass creation (more than 20 elements) asks for confirmation.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GroupAssemblyTools.cs

create_assembly_views

Create an assembly view

(viewType=3d_orthographic|detail_section|material_takeoff|part_list|single_category_schedule|sheet) or copy views from another instance (viewType=acquire_views + sourceAssemblyId). detail_section takes direction (HorizontalDetail, DetailSectionA/B, ElevationTop/Bottom/Left/Right/Front/Back); sheet needs titleBlockId; single_category_schedule needs scheduleCategoryId; optional viewTemplateId (+assignTemplate) assigns a template. Preview-first: dryRun defaults to true; dryRun=false creates. Additive, so no confirmation dialog.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GroupAssemblyTools.cs

create_group

Create a model group from element ids (Revit's Group command). Optional name sets the group type name. Preview-first: dryRun defaults to true; dryRun=false creates. A mass creation (more than 20 elements) asks for confirmation.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GroupAssemblyTools.cs

dissolve_assembly

Dissolve an assembly: the assembly instance is deleted, its members stay in the model and are returned (AssemblyInstance.Disassemble); the assembly's own views are deleted with it, as Revit's own dissolve does. Preview-first: dryRun defaults to true; dryRun=false asks for confirmation.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GroupAssemblyTools.cs

get_assembly_data

Read a Revit assembly instance or assembly type: name, member ids (optionally with per-member details and nested-group expansion), center/origin/transform in mm, naming category, allowsAssemblyViewCreation, and for a type its instances. Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GroupAssemblyTools.cs

get_element_containers

For each element id, report the group instance and the assembly that contain it (Element.GroupId / Element.AssemblyInstanceId), with their names. Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GroupAssemblyTools.cs

get_group_data

Read a Revit group: member ids (Group.GetMemberIds), optionally expanded recursively through nested groups and with per-member details, plus attached detail groups (available; shown and compatible for a given view). Pass groupId (instance) or groupTypeId - a group type's members are readable only through one of its instances. Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GroupAssemblyTools.cs

modify_assembly

Modify an existing assembly: action=add_members|remove_members|set_members (elementIds), action=rename (name), action=set_naming_category (namingCategoryId). Uses the API's gates (CanRemoveElementsFromAssembly, AreElementsValidForAssembly, IsValidNamingCategory). Preview-first: dryRun defaults to true; dryRun=false applies. Mass member changes (more than 20 elements) ask for confirmation.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GroupAssemblyTools.cs

modify_group

Modify a group without touching its members: action=rename (groupTypeId+name, or groupId+name) renames the type all instances share; action=show_attached_details|hide_attached_details (groupId, viewId, detailGroupTypeId) or show_all_attached_details|hide_all_attached_details (groupId, viewId) toggles attached detail groups in one view. Preview-first (dryRun defaults true).

Scope: | sources\revit-official\src\Pulse.Server\Tools\GroupAssemblyTools.cs

place_group

Place another instance of an existing model group type at x/y/z in millimetres. Preview-first: dryRun defaults to true; dryRun=false places the instance. Additive, so no confirmation dialog.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GroupAssemblyTools.cs

ungroup_group

Ungroup a group instance and keep its members in the model (Group.UngroupMembers). Deletes the group element and returns the released member ids. Attached detail groups cannot be ungrouped. Preview-first: dryRun defaults to true; dryRun=false asks for confirmation.

Scope: | sources\revit-official\src\Pulse.Server\Tools\GroupAssemblyTools.cs

Ifc

ifc_analyze_rebuildability

Analyze IFC DirectShapes and score feasibility of rebuilding them as native Revit elements.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_compare_original_vs_rebuilt

Compare volume/geometry between the original DirectShape and its native rebuild.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_export_basic

Export the active document to IFC. First-class flags cover the common options; use overrides for any other IFCExportOptions key.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_export_with_configuration

Export using a named configuration (built-in or custom) with optional key/value overrides.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_get_capabilities

Detect IFC version support and revit-ifc add-in presence

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_get_export_configuration

Get full details of a specific export configuration by name.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_link

Link an IFC file into the active document (creates a .ifc.RVT sidecar file managed by Revit).

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_list_export_configurations

List available built-in export configurations

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_list_rebuild_candidates

List elements above a rebuild confidence threshold.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_open_or_import

Open or import an IFC file as a native Revit project (actions: open | import).

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_rebuild_family_instances

Place family instances (doors, windows, furniture) from IFC DirectShapes.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_rebuild_floors

Rebuild native floors from IFC DirectShapes. dryRun defaults to true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_rebuild_openings

Cut openings in rebuilt walls/floors based on IFC opening DirectShapes.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_rebuild_roofs

Rebuild native roofs from IFC DirectShapes. dryRun defaults to true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_rebuild_structural_members

Rebuild columns and beams from IFC DirectShapes. dryRun defaults to true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_rebuild_walls

Rebuild native walls from IFC DirectShapes. dryRun defaults to true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_reload_link

Reload an existing IFC link, optionally from a new file.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

ifc_set_family_mapping_file

Set the family mapping file used by subsequent IFC exports.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

`ifc_tag_unreconstructable_elements`

Tag IFC DirectShapes that cannot be rebuilt by writing a marker parameter.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

`ifc_validate_request`

Validate IFC file path, extension, and schema version.

Scope: | sources\revit-official\src\Pulse.Server\Tools\IfcTools.cs

Link

`add_linked_file`

Adds a new Revit linked file from a file path and optionally places an instance at the given position.

Scope: | sources\revit-official\src\Pulse.Server\Tools\LinkTools.cs

`align_link_to_host`

Aligns a link instance to the host project's internal origin, shared coordinates, or project base point.

Scope: | sources\revit-official\src\Pulse.Server\Tools\LinkTools.cs

`cad_link_cleanup`

Analyze and clean up imported/linked CAD files. action=list|delete.

Scope: | sources\revit-official\src\Pulse.Server\Tools\LinkTools.cs

`get_coordination_models`

Read-only listing of Autodesk Revit Coordination Models with type metadata and optional instances.

Scope: | sources\revit-official\src\Pulse.Server\Tools\LinkTools.cs

`get_link_transform`

Returns the full transform of a linked file instance.

Scope: | sources\revit-official\src\Pulse.Server\Tools\LinkTools.cs

`get_linked_file_instances`

Lists the LINKED Revit models (RVT links) in this project, grouped by type, with load status and path. Use for 'what links / linked models are in this project'. The default lean response gives instance ids and pinned state; for a link's placement transform use `get_link_transform`, or pass `verbose=true`.

Scope: | sources\revit-official\src\Pulse.Server\Tools\LinkTools.cs

`get_selected_linked_elements`

Returns info about currently selected link instances.

Scope: | sources\revit-official\src\Pulse.Server\Tools\LinkTools.cs

highlight_linked_element

Highlights an element inside a linked model with an optional section box.

Scope: | sources\revit-official\src\Pulse.Server\Tools\LinkTools.cs

manage_links

List, reload, reload-from-path, unload, or remove linked files. To add a NEW link use add_linked_file instead.

Scope: | sources\revit-official\src\Pulse.Server\Tools\LinkTools.cs

move_link_instance

Moves a linked file instance. mode=delta applies (x,y,z) as an offset; mode=absolute places the origin at (x,y,z). Values are in mm.

Scope: | sources\revit-official\src\Pulse.Server\Tools\LinkTools.cs

pin_unpin_link_instance

Pins or unpins linked file instances.

Scope: | sources\revit-official\src\Pulse.Server\Tools\LinkTools.cs

reload_linked_file_from

Reloads a linked Revit file from a different file path.

Scope: | sources\revit-official\src\Pulse.Server\Tools\LinkTools.cs

show_cross_model_elements

Select host elements plus elements in linked Revit models. Default strategy creates red DirectShape markers around each linked element (synchronous, robust). usePostCommandIsolate=true uses Revit's native IsolateElements instead - asynchronous, and cannot combine with section box / overrides in the same call.

Scope: | sources\revit-official\src\Pulse.Server\Tools\LinkTools.cs

Material

create_material

Create a new material in the Revit project.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MaterialTools.cs

delete_material

Delete a material from the project by ID or name.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MaterialTools.cs

duplicate_family_type

Duplicate a loadable family type with a new name and optional parameter overrides.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MaterialTools.cs

`duplicate_material`

Duplicate an existing material with a new name.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MaterialTools.cs

`get_compound_structure`

Get wall/floor/roof/ceiling layer structure by type ID or name.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MaterialTools.cs

`get_material_catalog_status`

Compare a material catalogue file against the materials in the open model and report the difference, per FIELD.

Read-only: nothing is written, so it is safe to run before any import. Accepts a CSV with a 'name' column plus any of manufacturer/model/description/url/cost/mark/keynote/comments (',' or ';' separated, both detected), or JSON with a 'materials' array. Read `fieldSummary` first: `change` counts values the model ALREADY has that differ from the catalogue - those an import would replace - while `fill` counts empty fields. An empty catalogue cell means 'not described', never 'clear it'. Names matching several materials are reported in `ambiguous` and skipped rather than guessed.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MaterialTools.cs

`get_material_properties`

Get detailed material properties by material ID or name: appearance, structural and thermal assets, plus the eight commercial identity fields (description, manufacturer, model, url, cost, mark, keynote, comments) under the same keys set `material_properties` writes, so the response can be sent back unchanged. An empty field is null; a field Revit does not carry on this material is listed in `identityUnavailable`.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MaterialTools.cs

`get_material_quantities`

Calculate material area and volume across elements, optionally filtered by category or restricted to the current selection.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MaterialTools.cs

`get_materials`

List the materials of the active Revit document, optionally filtered by class or name.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MaterialTools.cs

`import_material_catalog`

Apply a material catalogue (CSV/JSON) to the model's materials, writing manufacturer, model, description, url, cost, mark, keynote, comments. THE MODE IS THE CALLER'S CHOICE: mode='fillEmpty' (default) writes only where the model field is empty and so cannot lose data; mode='overwrite' also replaces values the model already has. Default to fillEmpty unless the user has said the catalogue wins - measured case: a model held keynote '09 29 00.D1' where the catalogue had the less specific '09 29 00', so the catalogue is not automatically the better source. An empty catalogue cell never writes in either mode. `cost` is NOT written by default even when the catalogue carries it: a price lands in the Revit Cost parameter and from there into schedules and delivered sheets, so it must be asked for - pass fields:["cost", ...]. Use `fields` to narrow the rest. Preview-first: dryRun defaults to true, and get_material_catalog_status shows the same diff for free. Names matching several materials are refused, never guessed.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MaterialTools.cs

`list_material_parameters`

Lists the parameters the model's materials actually carry - one row per parameter, native and SHARED alike - with storage type, whether it can be written, and on how many materials it is blank. Use this to decide WHICH parameters to enrich instead of assuming a fixed set: measured on a real model a material carries 44 parameters, so the eight commercial ones that get_material_properties returns are a subset. Two emptiness counts are reported side by side - `blankOn` (this tool's test: empty text, or a numeric zero) and `hasValueOn` (Revit's own verdict) - because the Revit docs do not say what HasValue returns for a text parameter holding "", and a disagreement between the two is information, not noise. Read-only. Then use get_parameter_values_by_category (OST_Materials, includeEmpty) to see WHICH materials leave a chosen parameter blank.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MaterialTools.cs

`set_compound_structure`

Modify compound structure on a wall/floor/roof/ceiling type. action=replace|add|remove|modify|set_wrapping. set_wrapping sets openingWrapping (none|exterior|interior|both), endCap (none|exterior|interior), and per-layer layerWrapping.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MaterialTools.cs

Meta

`analyze_send_code_gaps`

Groups every send_code_to_revit call recorded in the audit log by its `purpose` field, so a capability gap that keeps recurring shows up as a count instead of staying invisible across one-off calls. Grouping is exact-text match only (trimmed, case-insensitive) -- it does not guess that two differently-worded purposes describe the same gap; read the raw entries for that judgment. Read-only, no document required.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MetaTools.cs

`clear_cache`

Clear every entry from the plugin-side tool-result cache.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MetaTools.cs

flow_step

Declare which step of a registered flow you are on. Returns the flow envelope (flow, flowStep, flowStepCount) AND makes the step visible in the Revit ribbon - the one layer no MCP client can decline. Call it at the start of EVERY step including the last, before the tool doing that step's work; pass awaitingUser:true with the question at a gate. Then, when the flow ends - applied, nothing to do, declined, or abandoned - call it ONCE MORE with complete:true, so the indicator does not stay lit on a settled decision. That closing call does NOT enter a step: sending complete:true INSTEAD of announcing the last step would hide the indicator before that step's work even begins. Two calls on the last step, not one. Touches no model data and needs no open document.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MetaTools.cs

get_cache_stats

Return diagnostic hit/miss telemetry from the plugin-side tool-result cache.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MetaTools.cs

get_project_info

Get project name, address, levels, phases, worksets, and links from the active Revit document.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MetaTools.cs

get_session_info

Which Pulse pieces are actually running: plugin and server build timestamps, tool count, profile (dev vs production, and its port), locale, and whether a document is bound. Call it FIRST when a tool is missing from the list or behaves as if a fix were absent: a timestamp gap means a stale deploy, not a broken tool. Needs no open document.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MetaTools.cs

say_hello

Test MCP connection to Pulse. Displays a greeting in Revit.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MetaTools.cs

set_project_info

Set editable Project Information fields. Only the fields you pass are changed; others are left untouched.

Scope: | sources\revit-official\src\Pulse.Server\Tools\MetaTools.cs

Parameter

add_prefix_suffix

Add a prefix and/or suffix to parameter values across the model or a selection. Runs as a dry-run preview by default; set dryRun=false to apply the changes.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ParameterTools.cs

`add_shared_parameter`

Add a shared parameter to project categories. The data type of a newly created definition is honored (a typed shared parameter, not always Text). Runs as a dry-run preview by default; set `dryRun=false` to apply the binding.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ParameterTools.cs`

`bulk_modify_parameter_values`

Set or change a parameter value on MANY elements at once (bulk edit / find-and-replace) - e.g. set Comments, Mark, or a shared parameter across a whole category or the selection. For a single element use `set_element_parameters`.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ParameterTools.cs`

`clear_parameter_values`

Clear parameter values on elements by category or scope

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ParameterTools.cs`

`export_shared_parameter_file`

Export the shared parameter file: returns its groups and definitions, and writes a copy to `filePath` when one is given.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ParameterTools.cs`

`filter_by_parameter_value`

Find or list elements by parameter VALUE: one condition, or several combined with AND/OR via the conditions array. Conditions: `equals`, `not_equals`, `contains`, `not_contains`, `begins_with`, `ends_with`, `greater_than`, `less_than`, `is_empty`, `is_not_empty`. Use for 'fire-rated doors', 'walls of type X', 'elements missing a Mark', 'rooms with area > N'. For type-level string parameters (e.g. type name) set `parameterType="type"` - `parameterType="both"` may not resolve them. To find by CATEGORY only use `ai_element_filter`.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ParameterTools.cs`

`manage_global_parameters`

Manage global parameters (project-level named values). Actions: `list` | `get` | `create` | `set` | `delete` | `rename` | `set_formula` | `move_up` | `move_down` | `sort`. Unlike project/shared parameters, global parameters CAN be renamed. The writing actions default to a preview: pass `dryRun=false` to apply.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ParameterTools.cs`

manage_project_parameters

Manage project parameters. Actions: list | create | delete | modify | set_group | set_binding_type | rename. 'delete' also removes non-shared parameters (works around Revit bug REVIT-136670). Revit allows two project parameters to share one display name, so delete/modify/set_binding_type return AmbiguousInput with the candidates instead of picking one: pass parameterId (from 'list', or from error.context.candidates) to name exactly one. 'create' makes a SHARED parameter and says so - the Revit API cannot create a new non-shared one - with a random GUID whose backing file is discarded, so no shared-parameter FILE holds that definition - the model does: measured, the guid survives save/close/reopen and action:list reports it, but nothing portable carries it. Keep the guid if another model must ever pick up the same identity, because add_shared_parameter for the same name mints a different one and splits what looks like a single parameter. Use add_shared_parameter from the start when it must be reusable across models. Calling 'create' twice with one name makes TWO parameters with that name, which is how the ambiguity above arises; the response warns when the name is already bound. Every writing action previews by default - see the dryRun parameter for exactly which ones read it. 'rename' is NOT supported for a bound project parameter: Definition.Name is read-only, so the action always fails with PermissionDenied and hands back the only workaround there is - create the new name, read the old values with export_elements_data, write them back under the new name with sync_csv_parameters, verify them, and only then delete the old parameter; that last step is irreversible, so do not run it on the strength of the write reporting success.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ParameterTools.cs

set_element_parameters

Set parameter values on one or more elements. Pass requests as a JSON-encoded array string. Supports parameterName by display name and builtInParameter by Revit API enum name, e.g.

REBAR_SYSTEM_SPACING_TOP_DIR_1. UNITS: a numeric value on a length/area/etc. parameter is written in Revit internal units (feet) - to set a display value, pass a STRING WITH A UNIT (e.g. "3000 mm", "3 m") and Revit parses it unit- and locale-aware. A null value clears the parameter. Note: type-level parameters can be set through an instance when the type is resolvable.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ParameterTools.cs

sync_csv_parameters

Synchronize parameter values from CSV data into Revit elements.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ParameterTools.cs

transfer_parameters

Copy parameter values from source element to one or more target elements.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ParameterTools.cs

Point Cloud

pointcloud_clear_filter

Remove the selection filter of one point cloud and set its action to none, so it stops being highlighted or isolated. The state is read back. Preview-first: dryRun defaults to true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs

`pointcloud_delete`

Delete one point cloud instance. Irreversible. `deleteUnusedType` (default false) also deletes the type, but only when no other instance uses it. A pinned instance is refused unless `unpinTemporarily` is true, because Revit will not delete a pinned element. Preview-first: `dryRun` defaults to true and the preview reports the elements that would be removed with it.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs`

`pointcloud_detect_openings`

Find the openings (windows, doors) in a surveyed wall: regions of a facade where no points were returned while the masonry around them is covered. Needs `averageDistance` plus `referenceNormal/referencePoint` (the wall's plane); optional box region, `slabThickness` (0.3), `cellSize` (0.05), `minOpeningWidth/minOpeningHeight` (0.4). Each opening returns its rectangle in the wall's frame (u horizontal, w UP) and its corners in model coordinates. The rectangle is the empty CORE: up to one cell smaller per side. Read-only, sample-based.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs`

`pointcloud_detect_planes`

Detect planar surfaces in a sampled region (RANSAC, deterministic). Needs `averageDistance` (the sampling resolution) and tolerance (how close a point must be to count as on the plane); optional `minPointsPerPlane` (50), `maxPlanes` (5), `iterations` (500) and a box region (default: the cloud's extent). Each plane comes with its normal, inclination, inlier count and fraction and the inliers' extent. Read-only, sample-based: the response states the envelope the numbers are valid within.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs`

`pointcloud_floor_levelness`

Measure how level a floor or slab is, from a sampled region. Needs `averageDistance` and tolerance; optional `designElevation` measures against the design instead of the fitted plane, and `outlierTolerance` reports how much of the sample is beyond it. Box region optional (default: the cloud's extent). Reports the fitted plane's inclination and elevation plus signed deviation statistics. Read-only, sample-based.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs`

`pointcloud_get`

Read one point cloud in full: type, engine, source, placement transform, total transform, extent, scans, regions. Both transforms carry `uniform_scale`, which on a ReCap cloud is the type scale, not 1. Select by `elementId`, `uniqueId` or name; an ambiguous name is refused with candidates. Read-only, no transaction.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs`

`pointcloud_list`

List point cloud instances and types in the active document, with optional placement and total transform, bounds, scan and region counts. Read-only, no transaction; a missing source file is a per-item warning, not a failure. Lengths are in `lengthUnit`.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs`

`pointcloud_list_regions`

List the ReCap regions of one point cloud. Region names are the handle used by the per-region override tools. Read-only, no transaction; an empty list means the cloud defines no regions.

Scope: | sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs

`pointcloud_list_scans`

List the scans inside one point cloud, optionally with origins. Revit identifies a scan by name, so name is the handle returned. Read-only, no transaction; an unreadable origin is a warning on that scan.

Scope: | sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs

`pointcloud_load`

Load a point cloud source (ReCap .rcp/.rcs path, or an identification string for a non-file engine) and place one instance. The engine is verified against Revit's registry. Preview-first: dryRun defaults to true. Local paths must be under the user's own directories; network shares are allowed.

Scope: | sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs

`pointcloud_recap_project_info`

Read the ReCap SDK's own view of a point cloud's source project through an OPTIONAL reflective adapter that loads nothing: it reports the SDK type it found, the member inventory it really exposes, and the values it could extract unambiguously (each naming its member). If the SDK is not installed the response says so and names the assembly. Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs

`pointcloud_reset_overrides`

Reset a point cloud's graphics override in one view back to the default for one scope: visible, no colour override. viewId is required; scope is instance (default), scan or region, with scopeName for the last two. Only the requested scope is touched. Preview-first: dryRun defaults to true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs

`pointcloud_sample`

Sample points from one point cloud at a target spacing, optionally inside a volume given as planes. Returns a SAMPLE, never the whole cloud. output: summary (count/bounds/centroid), inline (points, up to the inline limit), artifact (CSV/PLY/JSONL into the Pulse sandbox). Read-only, no transaction.

Scope: | sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs

`pointcloud_sample_around_element`

Sample points of a point cloud around one or more elements: the union of their axis-aligned extents, expanded by padding. Returns a SAMPLE, never the whole cloud. Elements without a usable extent are reported as warnings. Read-only, no transaction.

Scope: | sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs

`pointcloud_sample_box`

Sample points of one point cloud inside a box volume (center, size, optional rotation about Z). Returns a SAMPLE, never the whole cloud. output: summary, inline (up to the inline limit) or artifact (CSV/PLY/JSONL into the Pulse sandbox). Read-only, no transaction.

Scope: | sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs

`pointcloud_set_filter`

Apply a selection filter to one point cloud and set the action: highlight the points inside the volume, or isolate them. filter is {kind:'box',box:{...}} or {kind:'planes',planes:[...]} in model coordinates and lengthUnit - the same volumes pointcloud_sample_box accepts. Revit does NOT preserve the filter action when the document is saved. Preview-first: dryRun defaults to true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs

`pointcloud_set_override`

Set a point cloud's graphics override FOR ONE VIEW: visibility and colouring, for the whole instance, one scan or one region. viewId is required (a model view, not a template). color_mode is no_override, fixed, elevation, normals or intensity; fixed needs fixedColor {r,g,b}; elevation/intensity need gradient {color1,color2}. Nothing outside the requested scope is touched. Preview-first: dryRun defaults to true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs

`pointcloud_set_transform`

Move and rotate one point cloud instance. mode absolute replaces the placement, relative applies the transform on top of the current one. transform is the object form or the 16-number row-major array pointcloud_get returns, translation in lengthUnit. Its uniform scale must be 1 or the instance's own (the round trip): a point cloud instance cannot be re-scaled or mirrored. A pinned instance needs unpinTemporarily. Preview-first: dryRun defaults to true. The placement is re-read after the change and rolled back if Revit disagrees.

Scope: | sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs

`pointcloud_wall_deviation`

Measure how far a sampled region sits from a reference plane: the modelled wall when referenceNormal and referencePoint are given (a point ON that plane, in lengthUnit), otherwise a plane fitted to the points. Needs averageDistance and tolerance; optional outlierTolerance and box region. Reports signed deviation statistics and which side the points fall on. Read-only, sample-based.

Scope: | sources\revit-official\src\Pulse.Server\Tools\PointCloudTools.cs

Project

`analyze_model_statistics`

Analyze element counts by category in the active Revit document.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`analyze_warning_fixability`

Triage model warnings for MANUAL correction. Per warning group it adds to `get_warnings`: a fixability classification (automatable | needs_design_decision | blocked) WITH the reason, plus per element the category, family/type, level, and what blocks an edit (pinned, group membership, CanDeleteElement, super-component). It FIXES NOTHING - it tells the user where to go in Revit and which warnings are design decisions only they can make. Classification matches ENGLISH description text; a localized Revit reports 'unknown' rather than guessing. Cost: one warning pass plus a per-element parameter read. `includeViews=true` also locates each element's views and is MUCH slower (one view-scoped collector per view, which can force a geometry rebuild) - off by default.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs`

`audit_families`

Audit families in the Revit project. Lists loadable (.rfa) families by default; set `includeSystemFamilies=true` to also list system-family types (wall/floor/roof/ceiling types, including Foundation Slab).

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs`

`batch_rename`

Batch rename elements or system types in the Revit project. Supports both loadable-family elements and system types (wall/floor/ceiling/roof types).

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs`

`check_model_health`

Run a model health check and return a health score.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs`

`clash_detection`

Detect clashes between two element categories. Uses true solid-geometry intersection by default (fewer false positives than bounding boxes).

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs`

`create_custom_voce`

Creates a direct-price custom voce (code + description + unit + price, no breakdown) in the personal 'Le mie voci' library, reusable across projects/computi. Preview-first: `dryRun` defaults true and writes nothing.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs`

`create_level`

Create, edit, rename, or delete a level. `action=create|set|rename|delete`. For set/rename/delete identify the level by `levelId` or name.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs`

`create_nuovo_prezzo_analisi`

Composes a Nuovo Prezzo (NP) via price analysis (elementary resources + overhead% + profit%, profit on cost+overhead), and saves it as a custom voce in 'Le mie voci'. Write tool; dryRun defaults true (preview the breakdown and the code that would be assigned, writes nothing).

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`create_room`

Create a new room in the Revit project.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`delete_custom_voce`

Deletes one voce from the personal 'Le mie voci' library by exact code. Preview-first: dryRun defaults true and deletes nothing. Does not affect assignments already recorded via `record_computo_voce`.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`delete_element`

Delete elements from the Revit project.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`duplicate_system_type`

Duplicate, rename, or delete a system type (wall, floor, roof, ceiling). `action=duplicate|rename|delete`. `action=delete` is preview-first (dryRun defaults to true) and reports the dependent elements - instances of the type - that would go with it; duplicate and rename apply immediately.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`export_elements_xpwe`

Build a PriMus XPWE computo directly from the model: one measurement row per instance, grouped by a parameter (the type by default), with a composable row description and the four PriMus dimensions mapped to parameters or constants. Reads Revit itself (category or current selection). Populates WBS/BIM tag fields. Preview-first: dryRun defaults true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`export_quantity_mapping`

Export the approved mapping between price-list items and Revit types/materials to a durable CSV. Rows carry the chosen comparison parameter and one approved value per mapped target. Preview-first.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`export_schedule_xpwe`

Build a PriMus XPWE computo from an EXISTING SCHEDULE: reads its visible columns and rows and maps them to XPWE roles. Conventional column names (Codice/Descrizione/Unità di misura/Quantità/WBS, IT or EN) are auto-detected - that is the 'XPWE-ready' schedule template; otherwise pass `columnMapping`. A mapped column missing from the schedule is reported, never ignored. Preview-first: dryRun defaults true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

export_xpwe

Export caller-assembled rows to a PriMus XPWE file (documentType 'pricelist' or 'computo'). This tool serializes what you pass; it does NOT query Revit or decide associations. It populates WBS (codiceWBSCAP/codiceWBS) and BIM tag (tagBIM) fields. Preview-first: dryRun defaults true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

get_available_family_types

List available family types in the Revit project.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

get_computo_cam_summary

Computes the CAM (Criteri Ambientali Minimi) share of an already-written computo: aggregates by importo (quantity × price) into CAM / non-CAM / unclassified buckets, from the item code's chapter. Read-only; needs NO open document (reads the .xpwe file). Does not decide compliance.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

get_computo_status

Read the computo work-store sidecar (<name>.computo.json) to RESUME a metric computo across sessions without losing the measurement recipe or the decisions: reports the recipe (unit, scope, dimension parameters, derived flags), per-scope progress (done/pending), decision counts and pinned rows, plus a resume hint. Read-only; needs NO open document (reads a file).

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

get_computo_totals

Reads the four monetary levels of a computo (direct cost, markup-applied price, taxable total, VAT-inclusive total) from an already-exported .xpwe plus the percentages set via set_computo_percentages. File-only, read-only; no Revit document required.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

get_element_ownership

WORKSHARED MODELS ONLY: who owns, who created and who last changed a set of elements. Per element it returns checkoutStatus (NotOwned | OwnedByCurrentUser | OwnedByOtherUser), the owner, modelUpdatesStatus versus central, and - unless statusOnly - the creator, the last user to save a change of it to central, and anyone with an outstanding editing request. Also rolls the set up per user (byOwner/byCreator/byLastChangedBy) so "who is working in this area" is one call. IT ANSWERS WHO, NEVER WHEN: the Revit worksharing API carries no timestamp of any kind, so no date can be inferred from this response - the payload says so in limits.timestampNote. Values are Revit's local cache of central and may be stale; they are for reporting to a person, not for deciding whether a write will succeed. A scope is REQUIRED (elementIds or categoryFilter, not both) - it never sweeps the whole model - and maxElements is capped at 1000 because the full detail level allocates one native object per element with no documented cost. Refuses on a non-workshared model, where ownership does not exist. Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`get_incidenza_manodopera`

Reads an already-exported .xpwe computo and aggregates the labor-cost incidence (art. 41 cc.13-14 D.Lgs 36/2023) per voce and on the works total. Read-only; needs NO open document (reads a file). groupBy defaults to "codice" (provisional - see the tool's own description).

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`get_parameter_values_by_category`

Histogram of one parameter's values across a category in a single call: each distinct value with its count, and whether it was read from the instance or the type. Replaces the `get_element_parameters + export_elements_data + filter_by_parameter_value` sequence. The scan is exhaustive so the counts are exact; only the distinct-value LIST is capped, and truncation is reported.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`get_phases`

List all project phases in the active Revit document.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`get_project_coordinates`

Where the model sits in the world: Project Base Point, Survey Point and Internal Origin - each with its position relative to the internal origin AND its shared position - plus the shared coordinates of the internal origin, the angle to True North, the active project location, and the site latitude/longitude. Answers 'where is this model' for a survey handover, a model federation or a georeferenced IFC check. Lengths in mm (shared coordinates also in metres and raw internal feet), angles in DEGREES: the Revit API returns decimal feet and radians, and this tool converts them. Read-only. The three quantities are NOT interchangeable and are returned separately: position (relative to the internal origin), sharedPosition (shared system), and the shared coordinates of the internal origin - which equal the base point's shared position only when the base point sits on the origin.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`get_project_units`

Declares the document's units: per measurable spec the Forge specType/d/unitType/d, accuracy, unit symbol and formatting flags, plus unitSystem (Metric|Imperial) and the decimal and digit-grouping symbols. Call it BEFORE interpreting any dimensional number - the same suite returns millimetres, internal feet and square metres depending on the field.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

`get_shared_parameters`

List all project parameters with their bindings and categories, optionally filtered by category.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

get_warnings

Get model warnings from the active Revit document. Warnings are GROUPED by failure type by default: warningGroups gives one row per class of problem with warningCount and affectedElementCount, ordered biggest-first, so 49 warnings read as ~25 distinct problems. The flat 'warnings' list is still returned. matchedWarnings is the true number passing the filters and 'truncated' says whether the flat page was capped.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

get_worksets

List the worksets of the active Revit document, with open/editable state, owner and element count. Only user worksets unless includeSystemWorksets is set.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

ingest_pricelist

Ingest a regional price list (PriMus XPWE) once into a searchable local index under ~/.pulse/pricelists. Then use search_pricelist. Does not read the Revit model.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

lines_per_view_count

Count detail lines per view (single document pass, safe on any model size) plus a project-wide model line count. Model lines have no owner view, so they are reported once at project level, not per view.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

list_custom_voci

Lists the personal 'Le mie voci' library of hand-written direct-price voci, optionally filtered by a substring query. Read-only; does not read the Revit model.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

list_family_sizes

List loaded families with type/instance counts and, when includeSize=true, the family file size in KB measured by exporting each family to a temp file. Disabled by default because exporting is slow on models with many families.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

list_pricelists

List every price list already ingested via ingest_pricelist (id, region, year, item count, when it was built). Use it to discover a valid prezzariold before calling search_pricelist. Read-only; does not read the Revit model.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

list_schedulable_fields

Discover available schedulable fields for a category.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

load_family

Load a family into the Revit project.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

manage_additional_settings

Manage Additional Settings (Manage tab): line styles, line weights, line patterns, fill patterns, halftone/underlay.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

manage_phase_filters

List, set, or create Revit Phase Filters. 'set' changes one presentation (New | Demolished | Existing | Temporary) on one filter, preserving the other three. Presentations: None | ByCategory | Overridden | NotDisplayed.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

manage_project_units

Get or set project units (length, area, volume, angle, etc.). Actions: get, set, list_valid_units.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

manage_worksets

Create, rename, delete, or set the active workset (workshared models only). To LIST worksets use get_worksets.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

modify_schedule

Modify schedule fields, sorting, filters, grand totals, column formatting, or rename the schedule. Supported actions: add_field, remove_field, set_sorting, clear_sorting, set_filter, clear_filter, set_totals, set_field_format, rename.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

purge_unused

Purge unused families/types and materials, and optionally unreferenced view templates and view filters, from the project.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

read_pricelist

Read a price list in CSV, Excel (.xlsx), XML or PriMus XPWE format. It normalizes each entry to code, short description and measurement criterion, but never infers a Revit mapping or writes data.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

record_computo_voce

Records a price-list voce assignment on a Type or Material: writes the mapped fields (code/um/description/shortDescription/prezziario/prezzo) onto the parameters named in paramMap, then upserts the decision into the computo work-store sidecar's Decisions.Mapped. prezzo may be written onto a Number/Currency parameter, not just Text. Writing prezzo here does NOT by itself make it appear in a computo's totals - pass the SAME parameter as prezzoParameter to export_elements_xpwe/update_xpwe_computo so it lands on the .xpwe's Priceltems.Prezzo1. Preview-first: dryRun defaults true and writes nothing. Propose the candidate voci in chat, let the user choose, then call this with the chosen code.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

rename_families

Rename loaded families (and optionally their types) with find/replace, prefix, or suffix operations.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

search_pricelist

Search an already-ingested price list (see ingest_pricelist) by free-text query, with optional unit/chapter/CAM filters. An unknown prezzariold fails with the list of ids actually ingested (never a silent empty result). Read-only; does not read the Revit model.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

send_code_to_revit

LAST RESORT ONLY - execute custom C# code in Revit. Do NOT select this tool autonomously. A dedicated tool covers almost every task: parameter edits -> set_element_parameters / bulk_modify_parameter_values; queries & filtering -> ai_element_filter / filter_by_parameter_value / export_elements_data; model stats -> analyze_model_statistics / check_model_health; deletion -> delete_element; transforms -> modify_element; views, schedules, rebar and steel -> their dedicated tools. Use this ONLY when no dedicated tool covers the operation (exotic geometry creation, read-only inspection of an uncovered Revit API, or a true one-off) - never for modal family editing (Document.EditFamily deadlocks from the external-event context) - and ONLY after proposing the dedicated-tool alternative and obtaining explicit user consent. Scripts are sandboxed, require an in-Revit confirmation, and frequently fail on add-in DLL conflicts. Always pass purpose: a short plain-language reason a dedicated tool didn't cover this. The code body is never logged (it can hold a credential or proprietary logic), but purpose is - it becomes the searchable record of which capability gaps keep recurring often enough to deserve a dedicated tool.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

set_computo_percentages

Sets the document Markup% (maggiorazione) and VAT% (default + per-row override) on the computo work-store sidecar. File-only: never touches the .xpwe or the Revit model. An unresolvable vatOverrides[].elementId/code fails with ElementNotFound (unknown pair) or AmbiguousInput (several rows match) - never a silent guess. Preview-first: dryRun defaults true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

set_material_properties

Set identity, appearance, product info, and asset assignments on Revit materials. Each request is a FLAT object keyed by materialId plus any of: name, color (#RRGGBB), transparency, shininess, smoothness, materialClass, materialCategory, description, manufacturer, model, url, cost, mark, keynote, comments, appearanceAssetId, structuralAssetId, thermalAssetId (asset ids: 0/-1 clears).

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

sync_computo_model_state

Projects the computo's per-element state (Non computato/Computato/Da aggiornare) onto the model: reads the work-store sidecar, re-measures with its recorded recipe, compares against the previous .xpwe, and stamps two hidden instance parameters (Pulse_StatoComputo/Pulse_NotaComputo) accordingly. Schedule-sourced computi are refused (no element identity to project onto). Preview-first: dryRun defaults true and writes nothing.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

tag_rooms

Tag rooms in the active view. Operates on the active view only - activate the correct view first.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

tag_walls

Tag walls at their midpoints in the active view. Operates on the active view only. Tags all walls by default, or a subset via wallIds.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

update_xpwe_computo

Update an EXISTING XPWE computo for a new phase: re-measures the model, pairs measurement rows by their element id, and reports added / updated (with the delta) / unchanged / removed. It DECIDES NOTHING: each category of change needs an explicit action before the file is rewritten, otherwise the call is refused with the list of pending decisions. Preview-first: dryRun defaults true.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

verify_computo_preflight

Deterministic pre-delivery check of computo: completeness (in-scope elements not mapped), internal consistency (items without code, qty<=0, pinned misaligned), UM inconsistent and double-counting 1:N; reconciliation vs abacus if you pass scheduleId. Read-only, does not decide delivery.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

verify_xpwe_computo

Verify an XPWE computo against a Revit schedule the designer has already validated - a second, independent witness (re-measuring the model would only confirm the tool agrees with itself). Reports matching voci, discrepancies with delta and percentage, and what exists on one side only. Names when the totals agree while the items do not (a compensating change hides in a total). Writes nothing.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

wipe_empty_tags

Find and remove empty or orphaned tags

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

workflow_clash_review

Detect clashes between two categories and create a 3D section-boxed view for visual review.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

workflow_data_roundtrip

Export parameters to Excel for external editing, then re-import once the file has been saved.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

workflow_model_audit

Run a complete model audit workflow.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

workflow_room_documentation

Auto-generate callout views (and optionally sections) for every room on a level.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

workflow_sheet_set

Auto-create a set of sheets with title blocks from a definition list: [{number, name, viewIds?}].

Scope: | sources\revit-official\src\Pulse.Server\Tools\ProjectTools.cs

Rebar

convert_rebar_system_to_rebars

Convert an area or path reinforcement system into standalone rebars (destructive). Provide systemId. Returns the resulting standalone rebar ids.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

create_area_reinforcement

Create an area reinforcement system on a host (wall/floor/foundation). majorDirection is JSON {x,y,z}; optional curves is a JSON array of {type:line|arc, start{x,y,z}, end{x,y,z}, mid?{x,y,z}} in mm for an explicit boundary.

memberCount/memberIds reflect the bars after regeneration; if memberCountNote is present the count was still 0 - re-read with get_area_reinforcement_data.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

create_fabric_area

Create a fabric area system on a host (wall/floor/foundation). majorDirection is JSON {x,y,z}; optional curves is a JSON array of {type:line|arc, start{x,y,z}, end{x,y,z}, mid?{x,y,z}} in mm forming a closed loop.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

create_fabric_sheet

Create a single fabric sheet in a host. Provide hostId and fabricSheetTypeId or fabricSheetTypeName; optional bendProfile is a JSON array of {type:line|arc, start{x,y,z}, end{x,y,z}, mid?{x,y,z}} in mm forming a closed loop (makes the sheet bent).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

create_free_form_rebar

Create an unconstrained free-form rebar from curve loops (mm) in a host. loops is a JSON array of loops, each a JSON array of curve specs {type, start, end, mid?}.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

create_path_reinforcement

Create a path reinforcement system on a host. curves is a JSON array of {type:line|arc, start{x,y,z}, end{x,y,z}, mid?{x,y,z}} in mm (required). Optional flip, pathTypeId, startHookId, endHookId. memberCount/memberIds reflect the bars after regeneration; if memberCountNote is present the count was still 0 - re-read with get_path_reinforcement_data.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

create_rebar_bending_detail

Create a rebar bending detail for a rebar in a view (Revit 2024+). Provide rebarId and viewId (a drafting/detail view); optional bendingDetailTypeId, position JSON {x,y,z} in mm, rotationDegrees. Returns a version error on Revit 2023.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

create_rebar_coupler

Create a rebar coupler connecting two bar ends, or cap one. Provide couplerTypeId or couplerTypeName (a Coupler-category family type), end1 (JSON {rebarId,end}) and optional end2 (JSON {rebarId,end}). 'end' is 0 or 1.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

create_rebar_from_curves

Create a rebar from explicit coplanar curves (mm) in a host. curves is a JSON array of {type:line|arc, start{x,y,z}, end{x,y,z}, mid?{x,y,z}}; normal is JSON {x,y,z}. Optional hooks/layout.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

create_rebar_from_shape

Create a shape-driven rebar in a host from a rebar shape. origin/xVec/yVec are JSON {x,y,z} in mm. Optional layout JSON.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

get_area_reinforcement_data

Read an area reinforcement system: major direction (mm vector), type id/name, host id, member rebar ids, boundary curve ids, member count.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

get_fabric_area_data

Read a fabric area system: type id/name, host id, sheet ids, sheet count, major direction (mm vector).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

get_fabric_rounding

Read the document fabric length-rounding rules (apply flag, segment/total length rounding in mm and method Nearest|Up|Down).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

get_fabric_sheet_data

Read a fabric sheet: type id/name, isBent, fabricNumber, cut overall length and width (mm).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

get_fabric_wire_data

Read the wire items of a fabric sheet in one direction. Provide fabricSheetId and direction (major|minor); optional maxWires (default 200). Returns per-wire diameter (mm), distance (mm), wire length (mm).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

get_path_reinforcement_data

Read a path reinforcement system: type id/name, host id, member rebar ids, curve element ids, additional offset (mm), primary bar orientation.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

get_rebar_api_capabilities

Report which version-gated reinforcement features the running Revit supports.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

get_rebar_bending_detail_data

Read a rebar bending detail (Revit 2024+): host rebar id, owner view id, position (mm) and rotation (degrees). Provide bendingDetailId. Returns a version error on Revit 2023.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

get_rebar_constraint_candidates

List the constraint candidates for one rebar handle. Provide rebarId and handleIndex (from manage_rebar_constraints action=list_handles). Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`get_rebar_constraints`

List the constrained handles of a rebar and whether its constraints can be edited.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`get_rebar_coupler_data`

Read a rebar coupler: couplerMark, quantity, type id/name, and each linked reinforcement descriptor {rebarId, end}. Provide couplerId.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`get_rebar_element_data`

Read a single rebar's core data: bar type, host, shape, layout rule, bar count, total length (mm), volume.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`get_rebar_geometry`

Return the centerline curves (mm) of a rebar at a bar position index (default 0). Optionally suppress hooks/bend radius.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`get_rebar_host_data`

Report reinforcement hosted by an element: validity and the rebar/area/path/fabric it contains, plus common cover.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`get_rebar_numbering`

Read rebar numbering. With rebarId returns that bar's schedule mark; without it returns every rebar's schedule mark plus the count of blank marks (a proxy for numbering gaps). The document-wide list is capped by maxResults (truncated/returnedCount flag a cut) and summaryOnly returns only the counts.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`get_rebar_rounding`

Read rebar length-rounding rules. Without rebarId returns the document default; with rebarId returns that bar's effective rounding. Method is Nearest|Up|Down. There is no volume rounding in the Revit API.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`get_rebar_splice_candidates`

Report candidate splice geometries for a rebar by rules (Revit 2025+, read-only). Provide rebarId and optionally spliceTypeId; position must be 'Middle' - the by-rules API documents End1/End2 as invalid. spliceTypeId must be an ElementType of OST_RebarSpliceType, obtained from list_rebar_splice_types - a coupler FamilySymbol is rejected. Returns a version error on older targets.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

get_rebar_splice_data

Read rebar splice data (Revit 2025+): for each bar end, lap length (mm), stagger (mm), splice position, connected rebar id/end, plus the splice chain. Provide rebarId. Returns a version error on older targets.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

get_rebar_varying_data

Read varying-length rebar state (Revit 2025+, read-only): canHaveVaryingLengthBars, varyingEnabled, and per-position centerline length (mm). Provide rebarId, optional includeBarLengths (default true). Returns a version error on older targets.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

get_reinforcement_settings

Read document-level reinforcement settings.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

include_exclude_rebar_bars

Show or hide a single bar of a rebar set in a view. Provide rebarId, viewId, barPositionIndex, hidden (true=hide).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

list_rebar_bar_types

List all rebar bar types (id, name, model and nominal diameter in mm).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

list_rebar_cover_types

List all rebar cover types (id, name, cover distance in mm).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

list_rebar_fabric_types

List fabric reinforcement types (fabric sheet types and fabric area types).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

list_rebar_hook_types

List all rebar hook types (id, name, hook angle in degrees).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

list_rebar_shapes

List all rebar shapes (id, name).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

list_rebar_splice_types

List rebar splice types (Revit 2025+; returns a version error on older targets).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

manage_fabric_rounding

Set the document fabric length-rounding rules. Fields: applyRules (bool), lengthRoundingMm (double), lengthRoundingMethod (Nearest|Up|Down). volumeRounding is unsupported by the API and reported in warnings.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

manage_rebar_constraints

Inspect/edit rebar constraints. Provide rebarId and action: list_handles | list_candidates (with handleIndex) | set_preferred (with handleIndex, candidateIndex) | remove_preferred (with handleIndex) | recompute (2025+). list_* actions are read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

manage_rebar_numbering

Manage rebar numbering. action=set_number writes a single bar's schedule mark (needs rebarId + newNumber). action=renumber|remove_gaps are not exposed by the Revit API and return a structured 'unsupported' result. A read-only target surfaces the issue in warnings rather than failing.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

manage_rebar_rounding

Set rebar length-rounding rules. Without rebarId edits the document default; with rebarId edits that bar. Fields: applyRules (bool), lengthRoundingMm (double), lengthRoundingMethod (Nearest|Up|Down). volumeRounding is unsupported by the API and reported in warnings.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

modify_rebar_bending_detail

Modify a rebar bending detail (Revit 2024+). Provide bendingDetailId and any of position JSON {x,y,z} in mm, rotationDegrees. Returns a version error on Revit 2023.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

move_rebar_in_set

Move a single bar within a rebar set by a translation vector (mm). Provide rebarId, barPositionIndex, translation JSON {x,y,z}. Pass reset:true to clear a prior move.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

place_fabric_sheet

Place an existing fabric sheet into a host. Provide fabricSheetId and hostId; optional transform is JSON {translation:{x,y,z}} in mm (default identity).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

propagate_rebar

Propagate rebar to similar hosts. NOTE: the Revit API exposes no rebar-propagation method on any supported version; this returns a structured 'unsupported' result. Provide rebarId, optional targetHostIds JSON array.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`remove_fabric_reinforcement_system`

Remove a fabric area reinforcement system (destructive). Provide fabricAreaId.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`remove_rebar_splice`

Remove a rebar splice at a bar end (Revit 2025+). Provide rebarId and optional barEnd (0 or 1; default 0). Defaults to a preview that also reports whether a splice exists at that end: pass dryRun=false to apply. Returns a version error on older targets.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`remove_rebar_system`

Remove an area or path reinforcement system (destructive). Provide systemId.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`set_area_reinforcement_layers`

Activate or deactivate a layer of an area reinforcement system. Provide areaReinforcementId, layer (top_major|top_minor|bottom_major|bottom_minor) and active.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`set_fabric_sheet_bend_profile`

Set the bend profile of a bent fabric sheet. Provide fabricSheetId and bendProfile (a JSON array of {type:line|arc, start{x,y,z}, end{x,y,z}, mid?{x,y,z}} in mm forming a closed loop). Only valid when the sheet is bent.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`set_path_reinforcement_options`

Set options on a path reinforcement system. Provide pathReinforcementId and any of additionalOffsetMm, primaryBarOrientation (TopOrExterior|BottomOrInterior|NearSide|FarSide), alternatingBarOrientation. Unsupported keys are reported in warnings.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`set_rebar_coupler_visibility`

Set a coupler unobscured (solid) or obscured in a view. Provide couplerId, viewId, unobscured (bool).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`set_rebar_hooks`

Set the hook type at rebar ends. Provide rebarId and startHookId and/or endHookId (pass 0 to clear an end's hook). Works on all Revit versions.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

`set_rebar_host`

Reassign a rebar to a new host. Provide rebarId and newHostId (must be a valid rebar host).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

set_rebar_layout

Set the distribution layout of a shape-driven rebar. layout is JSON {rule, number?, arrayLengthMm?, spacingMm?, barsOnNormalSide?, includeFirstBar?, includeLastBar?}. Required numerics per rule: fixed_number -> number 1..1002 + arrayLengthMm>0; maximum_spacing / minimum_clear_spacing -> spacingMm>0 + arrayLengthMm>0; number_with_spacing -> number 1..1002 + spacingMm>0; single -> none. Invalid values return InvalidInput before the transaction.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

set_rebar_shape

Change the shape of a shape-driven rebar. Provide rebarId and shapId or shapeName.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

set_rebar_terminations

Set rebar end terminations (orientation/rotation). Revit 2027+ only; returns a version error on older targets. Provide rebarId, end (0|1), orientation, rotationDegrees.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

set_rebar_varying

Enable/disable a varying-length rebar set (the 'Varying Rebar Set' command, Revit 2025+). Provide rebarId and enabled (bool): when true the set's constraints drive per-bar lengths. The set must be shape-driven and eligible (CanHaveVaryingLengthBars). Returns a version error on older targets.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

set_rebar_visibility

Set rebar view presentation. Provide rebarId, viewId, and unobscured (show in front of host).

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

set_reinforcement_settings

Set document-level reinforcement settings. Provide any of hostStructuralRebar, rebarShapeDefinesHooks, rebarShapeDefinesEndTreatments (bools). Some toggles only apply when the document has no reinforcement; blocked changes are reported in warnings.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

splice_rebar

Splice a rebar by rules (Revit 2025+). Provide rebarId and optionally spliceTypeId; position must be 'Middle' - the by-rules API documents End1/End2 as invalid and they belong to manual line-based overloads this tool does not expose. spliceTypeId must be an ElementType of OST_RebarSpliceType, obtained from list_rebar_splice_types - a coupler FamilySymbol is rejected. Replaces the bar with several segments, so this defaults to a preview: pass dryRun=false to apply. Returns the resulting rebar ids, or a version error on older targets.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

split_rebar

Split a shape-driven rebar set into two sets at a given bar position. Provide rebarId and splitAtPosition (1..count-1). Returns the original and new rebar ids.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

transfer_rebar_annotations

Transfer rebar annotations between views by recreating MultiReferenceAnnotations over the rebars visible in the source view. Provide sourceViewId, targetViewId. Per-element issues are reported in warnings[].

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

unify_rebars

Unify compatible standalone bars into one (Revit 2025+). Provide rebarIds (JSON array of >=2 ids); bars are unified pairwise into a single rebar. All but the FIRST id are consumed and their ids become invalid, so this defaults to a preview: pass dryRun=false to apply. Returns a version error on older targets.

Scope: | sources\revit-official\src\Pulse.Server\Tools\RebarTools.cs

Structural Steel

add_steel_fabrication_info

Add steel fabrication information to Revit elements so they participate in steel detailing (SteelElementProperties). Provide elementIds as a JSON array, e.g. [123,456]. Supports dryRun. Returns the ids that received fabrication info plus any skipped ids.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

add_steel_instance_void_cut

Add an instance void cut so a void family instance cuts another element (InstanceVoidCutUtils). Provide voidInstanceId (the cutting void instance) and targetElementId (the element to be cut). Supports dryRun. Note: this is a generic Revit geometry cut, not steel-specific.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

add_steel_solid_cut

Add a solid cut so one element cuts another (SolidSolidCutUtils). Provide cutElementId (the cutter) and targetElementId (the element to be cut). Optional splitFaces (default false). Supports dryRun. Note: this is a generic Revit geometry cut, not steel-specific.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

analyze_structural_steel_model

Document-wide structural steel summary: counts of connection handlers, connection types, connection handler types, approval types, and structural framing/column elements carrying SteelElementProperties. summaryOnly returns counts only; otherwise capped sample arrays via maxResults.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

check_steel_cut_eligibility

Check whether one element can cut another via a solid cut and/or an instance void cut, without mutating. Provide cutElementId (the cutter) and targetElementId (the element to be cut). Returns solidCutEligible (+ solidCutFailureReason when false) and instanceVoidCutEligible.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

create_default_steel_connection_handler_type

Create the default StructuralConnectionHandlerType for the document (CreateDefaultStructuralConnectionHandlerType). Returns the new type id. Supports dryRun.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

create_generic_steel_connection

Create a generic structural connection between two or more elements (works without an installed connection provider - the safe baseline). Provide elementIds (JSON array of >=2 element ids); optional connectionName. Supports dryRun.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

create_steel_connection

Create a typed structural connection between two or more elements from a connection handler type (connectionHandlerTypeId or connectionHandlerTypeName). Requires an installed connection provider/type. Provide elementIds (JSON array of >=2 ids). Supports dryRun. inputPoints are accepted but not yet wired (Revit exposes no public ConnectionInputPoint constructor).

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

create_steel_connection_handler_type

Create a StructuralConnectionHandlerType. Provide name; optional familyName (default empty); optional guid (a new GUID is generated when omitted). Supports dryRun. Returns the new type id and its connection GUID.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

create_steel_structural_connection_type

Create a StructuralConnectionType bound to a family symbol. Provide familySymbolId (a valid connection family symbol); applyTo = BeamsAndBraces | ColumnTop | ColumnBase | Connection (default Connection); optional name. Supports dryRun.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

delete_steel_connection

Delete a structural connection handler by connectionId. Destructive - supports dryRun to preview. The connected elements themselves are not deleted.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`get_instance_void_cut_relationships`

Read the instance-void cut relationships of an element (`InstanceVoidCutUtils`): `cuttingVoidInstances` (void instances that cut this element) and `elementsBeingCut` (elements this element cuts when it is itself a cutting void instance). Counts are always returned; arrays unless `summaryOnly`, truncated to `maxResults` (default 100). Generic Revit geometry, not steel-specific.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs`

`get_solid_cut_relationships`

Read the solid-solid cut relationships of an element (`SolidSolidCutUtils`): `cuttingSolids` (solids that cut this element) and `solidsBeingCut` (solids this element cuts). Counts are always returned; arrays unless `summaryOnly`, truncated to `maxResults` (default 100). Generic Revit geometry, not steel-specific.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs`

`get_steel_connection_applicability`

Report a `StructuralConnectionType`'s applicability hints. Revit exposes no public 'does this type apply to these elements' predicate, so this returns the type's `ApplyTo` + family symbol id and, for any supplied `elementIds`, their categories - clearly labelled as advisory.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs`

`get_steel_connection_data`

Read a structural connection handler by id: type id/name, connected element ids, origin, custom/detailed flags, approval type id, code-checking status, override-type-params flag.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs`

`get_steel_connection_input_points`

Read the input points of a structural connection handler: each point's id (GUID) and position (x,y,z in mm). Provide `connectionId`.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs`

`get_steel_connection_settings`

Read the document-wide `StructuralConnectionSettings` (currently exposes the `IncludeWarningControls` flag).

Scope: | `sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs`

`get_steel_connection_type_data`

Read a structural connection type by id. Returns `StructuralConnectionType` (family symbol id, `applyTo`) or `StructuralConnectionHandlerType` (connection GUID, generic/custom/detailed flags) data depending on the element kind.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs`

`get_steel_connection_validation`

Report validation warnings for a structural connection handler. The Revit API exposes no public producer of `ConnectionValidationInfo` for a placed handler, so this returns `validationAvailable=false` with the handler's code-checking status and a note. Use the general `get_warnings` tool for document-level failures.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`get_steel_cut_data`

Read cut relationships for an element: solid-solid cuts (cutting solids + solids being cut via `SolidSolidCutUtils`) and instance-void cuts (cutting void instances + elements being cut via `InstanceVoidCutUtils`).

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`get_steel_element_fabrication_properties`

Read the steel fabrication properties of an element: whether it has `SteelElementProperties` (`hasFabricationProperties`) and its fabrication unique id (GUID string). Provide `elementId`.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`get_steel_element_properties`

Read steel fabrication properties of an element: whether it carries `SteelElementProperties` and its fabrication unique id (GUID). External-id and material-link enumeration are not exposed by the Revit SDK. Use `summaryOnly` for flags only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`get_steel_element_warnings`

Report steel fabrication warnings for an element (or all elements if `elementId` is omitted). The Revit SDK exposes no steel-specific warning API; this returns count 0 with a note. Use the general `get_warnings` tool for document-level failures. `summaryOnly` returns counts only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`get_steel_external_id_map`

Report the steel fabrication external-id map for an element. The Revit SDK does not expose per-element external-id enumeration; this returns the fabrication unique id (if any) and count 0 with a note.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`get_steel_fabrication_unique_id`

Read the steel fabrication unique id (GUID) of an element from its `SteelElementProperties`. Provide `elementId`. Returns a note when the element has no steel fabrication properties.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`get_steel_material_links`

Report steel fabrication material links for an element. The Revit SDK does not expose linked-material enumeration on `SteelElementProperties`; this returns count 0 with a note.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`get_steel_reference_by_fabrication_id`

Resolve the Revit element referenced by a steel fabrication GUID. Provide fabricationGuid (a GUID). Returns found=true with the referenced elementId, or found=false when no element matches.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`get_structural_connection_provider_data`

Report a structural connection provider's metadata/capabilities by id/key. StructuralConnectionsProviderData is an opaque provider-filled buffer with no public reader, so this reports available=false with a note.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`get_structural_connection_provider_registry`

Report registered structural connection providers (Autodesk Steel Connections, IDEA StatiCa, etc.). The Revit API exposes no public query on the provider registry, so this reports availability=false with a note rather than fabricating a list.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`get_structural_connection_validation_info`

Report validation detail for a placed structural connection (connectionId). No public API produces a populated ConnectionValidationInfo for an existing handler, so this returns the handler's CodeCheckingStatus and a note.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`get_structural_steel_api_capabilities`

Report which structural steel features the running Revit version supports: SteelElementProperties, structural connections, cut utils, custom-connection mutation API (removed in R27), and whether any structural connection provider is detectable.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`list_steel_approval_types`

List StructuralConnectionApprovalType definitions: id, name. Use maxResults (default 100) and summaryOnly for counts-first browsing.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`list_steel_connection_handler_types`

List StructuralConnectionHandlerType definitions: id, name, connection GUID, generic/custom/detailed flags. Use maxResults (default 100) and summaryOnly for counts-first browsing.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`list_steel_connection_handlers`

List structural connection handlers in the document: id, type id/name, connected element count, custom/detailed flags. Use maxResults (default 100) and summaryOnly for counts-first browsing.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`list_steel_connection_providers`

List installed structural connection providers. The public Revit API exposes no queryable provider registry; this returns count 0 with an explanatory note.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`list_steel_connection_types`

List StructuralConnectionType definitions in the document: id, name, family symbol id, applyTo. Use maxResults (default 100) and summaryOnly for counts-first browsing.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`manage_custom_steel_connection_type`

Mutate a custom structural connection (handler). action = add_references | remove_references | add_elements | remove_subelements. NOTE: Revit's custom-connection mutation needs interactively-picked References/Subelements that cannot be built from JSON, so this tool validates inputs and returns a structured error rather than guessing. The legacy add/remove APIs were removed in Revit 2027.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`manage_steel_approval_type`

Administer StructuralConnectionApprovalType definitions. action = create (requires name) | list. The Revit API exposes no rename/delete for approval types, so those actions return a structured error.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`modify_steel_connection_inputs`

Add or remove connected elements on a structural connection handler. action = add_element_ids | remove_element_ids (provide elementIds[]). add_references / remove_references are not supported via this tool (Revit References cannot be built from JSON ids). Returns accepted/skipped counts.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`remove_steel_instance_void_cut`

Remove an instance void cut between a void family instance and another element (InstanceVoidCutUtils). Provide voidInstanceId and targetElementId. Generic Revit geometry op, not steel-specific.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`remove_steel_solid_cut`

Remove a solid cut between two elements (SolidSolidCutUtils). Provide cutElementId and targetElementId. Generic Revit geometry op, not steel-specific.

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`set_steel_connection_approval`

Set the approval type of a structural connection handler. Provide connectionId and approvalTypeId or approvalTypeName (verified against the document's StructuralConnectionApprovalType definitions).

Scope: | sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs

`set_steel_connection_default_order`

Reset a structural connection handler to its default element order (`SetDefaultElementOrder`). Provide `connectionId`.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs`

`set_steel_connection_status`

Set the code-checking status of a structural connection handler. `status = NotCalculated | OkChecked | CheckingFailed`.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs`

`set_steel_connection_type`

Change a structural connection's type. Revit exposes no in-place type setter, so this recreates the connection: it reads the connected elements, deletes the old handler, and creates a new one with `connectionHandlerTypeId|connectionHandlerTypeName`. Requires an installed connection provider/type. Supports `dryRun`. Existing input points are not preserved.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs`

`set_steel_connection_type_family_symbol`

Re-bind a `StructuralConnectionType` to a different family symbol. Provide `connectionTypeId` and `familySymbolId`. The new symbol is validated against the type's existing `ApplyTo`. Supports `dryRun`.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs`

`set_steel_fabrication_unique_id`

Set the steel fabrication unique id (GUID) of an element's `SteelElementProperties`. Provide `elementId` and `uniqueId` (a GUID). The element must already have steel fabrication properties (run `add_steel_fabrication_info` first).

Scope: | `sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs`

`set_steel_solid_cut_face_splitting`

Set whether the cutting solid's faces are split at an existing solid cut (`SolidSolidCutUtils.SplitFacesOfCuttingSolid`). Provide `cutElementId`, `targetElementId` and `split` (bool, required). Requires an existing solid cut between the two elements. Supports `dryRun`. Generic Revit geometry op, not steel-specific.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\StructuralSteelTools.cs`

User Flow

`manage_user_flows`

Create, get, list or delete user-defined Pulse flows (stored per-user). create needs a 'flow' JSON object `{id,title,description,triggers[],steps[{name,gate,tools[]}]}`; get and delete need 'id'.

Scope: | `sources\revit-official\src\Pulse.Server\Tools\UserFlowTools.cs`

View

activate_view

Open and activate a view or sheet by viewId or exact viewName. Needed before tools that act ONLY on the active view (tag_rooms, tag_walls, color_elements, create_dimensions). UI only: the model is not modified.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

align_viewports

Align viewports across sheets. 'placement' matches box centers; 'model' matches the box outline min-corner so equal-scale views of the same region line up.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

apply_view_template

List, apply, or remove view templates from views. action=list|apply|remove. A templateId that is not actually a view template is REFUSED with what it really is (measured: an ordinary FloorPlan and a door type both used to return success), so get the ids from action=list rather than from a view listing. COMPATIBILITY IS PER TARGET VIEW: Revit's View.IsValidViewTemplate is checked for each view before assigning, because a template valid for one view can be invalid for another and the raw assignment throws - views that cannot take it are reported in `skipped` with the reason instead of being silently counted. Both apply and remove report skippedCount, so a partial run is distinguishable from a complete one.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

batch_create_sheets

Create multiple sheets with title blocks and optional view placement. sheets is a JSON array: [{number, name, titleBlockName?, viewIds?}].

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

batch_modify_view_range

Modify view range offsets (top, cut plane, bottom, view depth) for multiple views. Offsets are in mm.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

close_views

Close open view windows: the listed viewIds, or all except the active one. Never closes the last window. UI only: views stay in the model and activate_view reopens them.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

create_placeholder_sheets

Create, list, convert, or delete placeholder sheets. action=create|list|convert|delete.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

create_preset_schedule

Create a schedule from a predefined template (e.g. RoomFinish, DoorHardware, WallQuantities, WindowSchedule).

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`create_schedule`

Create a new schedule view in Revit.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`create_sheet`

Create a new sheet in the Revit project.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`create_view`

Create a new view in Revit: floor plan, ceiling plan, section, elevation, drafting, or 3D view. Can timeout if run in parallel with other writes - best run alone.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`create_view_filter`

Create, apply, or list parameter-based view filters (action=create|apply|list). A filter carries one rule (parameterName/filterRule/filterValue) or several via the rules array combined with AND/OR. For apply: pass filterId+viewId, optionally overrideR/G/B.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`create_views_from_rooms`

Create callout, section, or elevation views from rooms with a naming pattern.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`delete_schedule`

Delete a schedule by ID or name

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`duplicate_schedule`

Duplicate a schedule with a new name. Optional duplicateOption and filterBySheet (schedules do not support dependent views - as_dependent is refused with a suggestion to use filterBySheet instead).

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`duplicate_sheet_with_content`

Duplicate a sheet including annotations and detail items

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`duplicate_sheet_with_views`

Duplicate a sheet N times with configurable view duplication options.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`duplicate_view`

Duplicate an existing view in Revit.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`get_current_view_elements`

List elements visible in the currently active view.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`get_current_view_info`

Get information about the currently active view in Revit.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`get_schedule_data`

Export schedule data as JSON from an existing schedule view. Always returns a `config` block: filters (with fieldName, filterType and typed value), sortGroupFields (order + grouping header/footer/blankLine), grandTotal flags, and per-column name-vs-heading. Filter double values are Revit internal units (feet), unconverted.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`get_view_graphics`

Read what a view is overriding: per-category hidden flag and the FULL graphic-override surface (projection/cut line colour, weight and pattern; surface and cut foreground/background pattern id, colour and visibility; halftone; transparency; detail level), with IsCategoryOverridable/CanCategoryBeHidden so you know what is settable. Also returns per-element overrides for elementIds, the applied filters IN ORDER (order decides which filter wins) with their overrides, visibility and enabled state, workset visibility, RVT link settings, the class-level hidden switches, the display model, and viewTemplateId + graphicsOverridesAllowed - a view driven by a template cannot be overridden directly, which is the commonest cause of a failed graphic write. Lean by default: only categories that are hidden or overridden are listed. Read-only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`list_open_views`

List the views of the active document that have an open window, and which one is active. Input for close_views and zoom_view.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`list_views`

List views, sheets and view templates with the ids the other view/sheet tools consume - including the VIEWPORT ids align_viewports needs and the schedule-instance ids on each sheet, which no other tool produces. Per view: id, name, viewType, scale, detailLevel, discipline, isTemplate, the ASSIGNED VIEW TEMPLATE (viewTemplateId + viewTemplateName) and areGraphicsOverridesAllowed, plus where it is placed. READ viewTemplateId BEFORE DIAGNOSING A FAILED GRAPHIC WRITE: the API states 'Parameters controlled by the template cannot be changed directly in this view', so a non-null template is the usual reason a scale/detail-level/override change appears to do nothing. scale, detailLevel and discipline are null when the view has no such property (the API throws for schedules, legends and sheets) - null means not-applicable, never 0. placements is an ARRAY because a legend can sit on many sheets, and a schedule reaches a sheet through a ScheduleSheetInstance rather than a Viewport, so each placement declares placedVia. Lean by default; matchingCount reports the population before maxResults so a truncated page never reads as the whole model.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`manage_unplaced_views`

List or delete views that are not placed on any sheet

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`manage_view_templates`

List, duplicate, delete, rename, or create view templates. action=list|duplicate|delete|rename|create_from_view. create_from_view builds a new template from an existing view (pass viewId). The writing actions default to a preview (delete reports how many views would lose their template): pass dryRun=false to apply.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`override_graphics`

Override element graphics in a view (colors, transparency, halftone, line weight).

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`place_viewport`

Place a view on a sheet as a viewport, with optional rotation and viewport type.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`rename_views`

Batch rename views using find/replace, prefix, or suffix operations. Scope it with filterName and/or viewTypes: WITHOUT them the operation targets EVERY non-template view in the document. The preview reports candidateCount (views the filter reached) and skippedUnchangedCount (already correctly named), so 'no view matched' and 'nothing to change' stay distinguishable.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

`zoom_view`

Zoom an open view window (default: the active view): fit, factor, rectangle or sheet_size. Returns the framed corners in mm. UI only.

Scope: | sources\revit-official\src\Pulse.Server\Tools\ViewTools.cs

Verified source

<https://github.com/LuDattilo/pulse-for-revit>

main / 2c0407f58c5266eb9466a4bdd2553a05f7da30b1

<https://pulse-ai.it/guida.html>