TANGODOWN

SECURITY

Entwicklerhilfe

Version 0.87.0 · 06. November 2026

Kostenloses Hobbyprojekt von Lars Stahl – ohne Gewähr. Aktuelle Fassung und Download: https://tangodown.eu

Inhalt

1 Überblick: Welche Anbindung wofür? 5

Entscheidungshilfe 5
Richtung der Daten 5
Gemeinsame Grundsätze 5

2 Plugins – Entwicklungshilfe 5

Was ist ein Plugin? 5
Sicherheit zuerst: die Plugin-Sandbox 6
Rechte nach Bedarf: Netz, Schlüssel, Dateien 6
Wo liegen Plugins? 8
Schritt für Schritt: dein erstes Plugin 8
plugin.json – Referenz 8
Eingabe 9
Ausgabe 9
Vorlage: Python 10
Vorlage: JavaScript 10
Vorlage: lokaler Webdienst mit FastAPI 10
Wo erscheinen Plugins? 11
Testen und Fehlersuche 11
Freigabe und Protokoll 11
Gute Praxis 11
Plugins im Marktplatz veröffentlichen 12

3 Register & geschützte Datenbanken anbinden 12

Die Einrichtungsmaske („ + Quelle einrichten“) 13
Beispiel 1: Fachverfahren mit Client-Zertifikat (mTLS) 13
Beispiel 2: POST mit JSON-Body und Token 13
Beispiel 3: interne PostgreSQL-Datenbank (nur lesend) 13
Was TangoDown erzwingt 14

4 REST-API 14

Was bringt sie? 14
Einschalten 14
Adresse und Anmeldung 15
Endpunkte 15
Eigene Datenbanken (benannte, fallübergreifende Listen) 16
Beispiele 16
Fehlercodes 18
Schutz 18

5 MCP – KI-Assistenten anbinden 18
Was ist MCP und was bringt es? 18
Einrichten 18
Werkzeuge 20
Beispiel-Aufträge an den Assistenten 20
Schutz und Datenschutz 20

6 Cyber-Intelligence: Schlüssel der Datenquellen 21

7 Eigene API-Konnektoren (ohne Programmierung) 21
Was bringen sie? 21
Einrichten 21
Schlüssel richtig eintragen 21
Konnektor oder Plugin? 22

8 Weitere Erweiterungswege 22

9 Updates 22

10 Sicherheit und Recht – Kurzfassung für Entwickler 22

TangoDown – Entwickler-Dokumentation

Diese Dokumentation beschreibt, wie man TangoDown mit eigenen Programmen erweitert und anbindet: **Plugins** (eigene Transformationen), die **REST-API** (eigene Skripte und Programme), **MCP** (KI-Assistenten wie Claude Code) und **eigene API-Konnektoren** (Webdienste ohne Programmierung). Alle Wege laufen über dieselben Schutzmechanismen wie die Oberfläche: Fallbezug, Zweckbindung, Freigaben und Audit-Log.

> 🍼 **Noch in Entwicklung und Erprobung:** TangoDown ist ein kostenloses Hobbyprojekt, das eine einzelne Person in ihrer Freizeit entwickelt. Die Software wird noch erprobt – es gibt sicher noch Fehler und vieles zu verbessern, und manche Bereiche oder Funktionen arbeiten womöglich noch nicht vollständig. Hinweise sind sehr willkommen und helfen, gezielt nachzubessern: am besten mit Version, den genauen Schritten bis zum Fehler (reproduzierbar), was du erwartet hast und was stattdessen passiert ist, gern mit Bildschirmfoto und dem Diagnosebericht („Hilfe & Info → Kontakt → 🩺 Diagnosebericht erstellen“) – an info@tangodown.eu oder über das Kontaktformular auf tangodown.eu. Wer die Arbeit unterstützen möchte, kann das gern per Krypto oder PayPal tun (tangodown.eu → Unterstützen).

Überblick: Welche Anbindung wofür?

Entscheidungshilfe

- **Du willst einen Wert (Domain, IP, Name …) mit eigener Logik anreichern und das Ergebnis im Graphen sehen? → Plugin.** Per Rechtsklick auf ein Element, im Terminal und in der Untersuchungs-Automatik. Python oder JavaScript in der abgeschotteten Sandbox – oder ein lokaler Webdienst (FastAPI).
- **Du willst einen fremden Webdienst abfragen, der JSON liefert (z.B. eine Firmen-API mit Schlüssel), ohne zu programmieren? → Eigener API-Konnektor.** URL-Vorlage, Schlüssel aus dem Vault, Felder per JSON-Pfad.
- **Du willst von außen auf den geöffneten Fall zugreifen – lesen, Elemente anlegen, verknüpfen – z.B. aus einem Skript, einer Pipeline oder Excel/Power Query? → REST-API.**
- **Du willst, dass ein KI-Assistent (Claude Code, Claude Desktop oder ein anderer MCP-fähiger Client) den Fall lesen und – wenn erlaubt – ergänzen kann? → MCP.**
- **Du willst wiederkehrende Abläufe innerhalb von TangoDown automatisieren? → Terminal** (Pipes, Variablen, Makros, `replay`) und **Untersuchungs-Automatik**.

Richtung der Daten

- **Plugins und Konnektoren:** TangoDown ruft *dein* Programm bzw. *einen* Dienst auf → Ergebnis kommt in den Fall.
- **REST-API und MCP:** *Dein* Programm bzw. *ein KI-Assistent* ruft TangoDown auf → liest oder ergänzt den geöffneten Fall.

Gemeinsame Grundsätze

- Alles läuft lokal auf diesem PC; Plugin-Code nur in der abgeschotteten Sandbox, Webdienste und API nur über 127.0.0.1.
- Nur mit ausdrücklicher Freigabe: Plugins einzeln (mit Prüfsumme), die API mit Token, Schreibzugriff gesondert.
- Jeder Zugriff steht im Audit-Log des Falls – was die Anwendung verlässt oder hineinkommt, ist nachvollziehbar.
- VS-NfD-Fälle sind über den normalen Token gesperrt. Für Programme und KI im eigenen Haus (On-Prem) gibt es einen eigenen **On-Prem-Token**, mit dem auch VS-NfD-Fälle abfragbar sind (siehe „REST-API" → On-Prem-Zugang).
- Ergebnisse aus Plugins, Konnektoren und KI sind Hinweise, keine Feststellungen. Neue Elemente erhalten die Einstufung F6 (nicht beurteilbar), bis jemand sie bewertet.

Plugins – Entwicklungshilfe

Was ist ein Plugin?

Ein Plugin ist ein Ordner mit einer Beschreibung **plugin.json** und Code in **Python** oder **JavaScript** – oder der Adresse eines **lokalen Webdienstes**. TangoDown übergibt den Wert eines Elements und bekommt neue Elemente, Verknüpfungen und eine Notiz zurück.

Sicherheit zuerst: die Plugin-Sandbox

Python- und JavaScript-Plugins laufen nicht als eigenes Programm auf deinem PC, sondern abgeschottet:

- **kein Dateisystem**, **kein Netz**, **keine Programme**, keine Fenster, keine Geräte
- kein Zugriff auf TangoDown, den Fall, den Vault, die lokale Schnittstelle oder andere Plugins
- jeder Lauf startet frisch; das Zeitlimit beendet auch Endlosschleifen
- TangoDown startet keine fremden Programme
- die Antwort wird streng geprüft: nur bekannte Elementtypen, Text mit begrenzter Länge; unzulässige Feldnamen und Steuerzeichen werden entfernt

Python steht mit der Standardbibliothek bereit – ohne Installation und ohne Zusatzpakete.

Rechte nach Bedarf: Netz, Schlüssel, Dateien

Braucht ein Plugin mehr – etwa eine Web-API abfragen oder eine Datei auswerten –, meldet es genau das in `plugin.json` unter `permissions` an. Bei der Freigabe siehst du die Liste und bestätigst sie; ändert sich etwas daran, ist die Freigabe weg.

```
"permissions": {
  "net": ["api.beispiel.org", "*.beispiel.net"],
  "secrets": ["shodan", { "name": "beispiel", "hosts": ["api.beispiel.org"] }],
  "files": "read"
}
```

- **net**: Hostnamen (ohne `https://` und Pfad; `*.` für Subdomains). Das Plugin bleibt selbst ohne Netz – es bittet TangoDown mit `td.fetch(...)`. TangoDown prüft den Host, erlaubt nur `https`, folgt keinen

Weiterleitungen, läuft über das eingestellte Routing (Tor/Proxy, Not-Aus) und schreibt jede Abfrage ins Audit-Log. Höchstens 50 Abfragen je Lauf, je 5 MB.

- **secrets**: Einträge aus „Vault & Konnektoren". Im Code steht nur der Platzhalter `{secret:name}` in Pfad, Abfrage, Kopfzeile oder Body; TangoDown setzt den Schlüssel erst beim Senden ein. Das Plugin sieht ihn nie, auch in Antworten wird er geschwärzt.
  - **Jeder Schlüssel geht nur an seinen Dienst.** Bekannte Einträge sind fest gebunden, z.B. `shodan` → `api.shodan.io`, `virustotal` → `www.virustotal.com` – auch wenn das Plugin weitere Hosts anmeldet. So kann ein Plugin einen Schlüssel nicht an einen eigenen Server schicken.
  - **Eigene Einträge** brauchen eine Host-Bindung: `{ "name": "beispiel", "hosts": ["api.beispiel.org"] }` (die Hosts müssen auch unter `net` stehen). Ohne Bindung setzt TangoDown den Schlüssel nirgends ein.
  - Im Hostnamen ist `{secret:…}` nicht erlaubt.
  - Bei der Freigabe sieht der Nutzer je Schlüssel, an welche Hosts er geht, und einen Warnhinweis: Das Plugin nutzt den Zugang im Namen des Nutzers (Kontingent, Kosten).
- **files**: `"read"` – das Plugin darf mit `td.readFile()` um eine Datei bitten; du wählst sie jeweils im Dialog aus (bis 20 MB, höchstens 5 je Lauf). Das Plugin kann keine Pfade nennen und nichts schreiben.
- **Kennung und Name:** Die `id` muss eindeutig sein. Liegen zwei Plugins mit derselben Kennung im Plugin-Ordner, läuft nur das erste; das zweite wird als fehlerhaft angezeigt. Heißen zwei Plugins gleich, zeigt das Rechtsklick-Menü Autor bzw. Kennung dazu.

Nutzung im Code:

```
// JavaScript
async function transform(input) {
  const r = await td.fetch('https://api.beispiel.org/lookup?q=' + encodeURIComponent(input.value), {
headers: { Authorization: 'Bearer {secret:beispiel}' } })
  if (!r.ok) return { entities: [], message: 'HTTP ' + r.status }
  const d = r.json()
  return { entities: [], note: JSON.stringify(d).slice(0, 2000) }
}
```

```
# Python: transform als async def, dann await td.fetch / td.fetch_json / td.read_file (Ergebnisse
als dict)
async def transform(inp):
    r = await td.fetch("https://api.beispiel.org/lookup?q=" + inp["value"], headers=
{"Authorization": "Bearer {secret:beispiel}"})
    f = await td.read_file()   # nur mit "files": "read" – {"name", "size", "text", "base64"}
    return {"entities": [], "note": r["text"][:2000]}
```

Ohne angemeldete Rechte wird jede Bitte abgelehnt. **Volle Rechte für Plugins gibt es bewusst nicht**: Ein Programm mit vollen Rechten hätte Zugriff auf alle Daten des Arbeitsplatzes. Wer das wirklich braucht, betreibt seinen Code als **lokalen Webdienst** (siehe unten) – der läuft sichtbar außerhalb von TangoDown in deiner Verantwortung und tauscht nur JSON aus.

Die Abschottung ist darauf ausgelegt, dass ein Plugin höchstens eine falsche Antwort liefern kann – die du wie jeden Hinweis prüfst. Eine absolute Garantie gibt es bei keiner Software; gib deshalb nur Plugins aus vertrauenswürdiger Quelle frei.

⚠ **Sicherheitsgrenze – lokale Webdienste:** Lokale Webdienste (z.B. FastAPI) sind eigene Programme, die du selbst startest – außerhalb von TangoDown. TangoDown schickt ihnen nur den Wert und liest JSON zurück; was der Dienst sonst auf deinem PC tut, liegt in deiner Verantwortung. Nur vertrauenswürdige Dienste anbinden.

Wo liegen Plugins?

- In der App: **Werkzeuge** → **Plugins** → 📂 **Plugin-Ordner öffnen**.
- Je Plugin ein eigener Unterordner.
- ＋ **Beispiele anlegen** erzeugt drei Vorlagen und die Kurzanleitung `LIESMICH.md`:
  - `domain-teile` – JavaScript
  - `text-extrakt` – Python (findet E-Mails, Webadressen, IPs und Rufnummern im Text)
  - `fastapi-beispiel` – lokaler Webdienst mit FastAPI

Schritt für Schritt: dein erstes Plugin

1. Im Plugin-Ordner einen Unterordner anlegen, z.B. `mein-plugin`.
2. Darin `plugin.json` und `main.py` (oder `main.js`) anlegen – Vorlagen unten.
3. In TangoDown **Werkzeuge** → **Plugins** → ↻ **Neu einlesen**.
4. ✓ **Prüfen & freigeben**: Art, Datei und Prüfsumme kontrollieren und bestätigen.
5. ▶ **Ausprobieren**: Wert eingeben, Ergebnis als JSON ansehen – oder gleich ⇄ **In den Fall**.
6. Danach erscheint das Plugin im Rechtsklick passender Elemente.

Jede Änderung an einer Datei im Ordner hebt die Freigabe auf – nach dem Bearbeiten erneut prüfen und freigeben.

plugin.json – Referenz

```
{
  "id": "mein-plugin",
  "name": "Mein Plugin",
  "version": "1.0",
  "description": "Was es tut – erscheint in der Plugin-Liste",
  "author": "Name oder Team",
  "inputs": ["domain", "ip"],
  "kind": "python",
  "main": "main.py",
  "timeout": 30,
  "icon": "⚙"
}
```

- **id** (Pflicht): eindeutig; Buchstaben, Ziffern, `.`, `_`, `-`.
- **name**, **version**, **description**, **author**: Anzeige; Name und Version stehen als Quelle an jedem Ergebnis.
- **inputs**: Elementtypen, bei denen das Plugin angeboten wird, z.B. `domain`, `ip`, `email`, `person`, `username`, `phone`, `company`, `location`, `vehicle`, `btc`, `webcam`, `note`, eigene Typen – oder `"*"` für alle.
- **kind**: `"python"` oder `"js"` (Sandbox) oder `"http"` (lokaler Webdienst).
- **main** (python/js): die Code-Datei im Plugin-Ordner (Standard `main.py` bzw. `main.js`).
- **url** (http): z.B. `http://127.0.0.1:8765/transform` – nur `127.0.0.1`, `localhost` oder `[::1]`.
- **network** (nur http): `true`, wenn dein Dienst selbst ins Netz geht; im Air-Gap-Modus dann gesperrt.
- **timeout**: Sekunden (2–300, Standard 30). Der erste Python-Lauf braucht etwa 1–2 Sekunden zum Start der Laufzeit.
- **icon**: ein Zeichen für das Menü.
- **permissions** (python/js, optional): Rechte für Netz, Schlüssel, Dateien – siehe „Rechte nach Bedarf“.

Eingabe

Dein Code bekommt ein Objekt `input` (Python: ein `dict`):

```
{
  "value": "mail.shop.example.org",
  "type": "domain",
  "label": "mail.shop.example.org",
  "props": { "registrar": "Beispiel-Registrar", "url": "https://mail.shop.example.org/" }
}
```

- **value**: der Wert (bei Elementen mit `url`-Angabe die URL, sonst die Bezeichnung).
- **type**: Elementtyp; **label**: Bezeichnung; **props**: die Angaben des Elements (Feldname → Text).
- Bewusst nicht enthalten: Vault-Schlüssel, andere Elemente, Fallmetadaten.

Ausgabe

Gib ein Objekt zurück:

```
{
  "entities": [
    { "type": "domain", "label": "example.org", "props": { "art": "registrierbare Domain" }, "link":
"gehört zu" },
    { "type": "location", "label": "Hamburg", "geo": [53.55, 9.99], "link": "Standort" }
  ],
  "note": "Freitext – wird mit Zeitstempel und Quelle als Notiz an das Element gehängt",
  "message": "kurze Meldung für die Statuszeile"
}
```

- **entities**: höchstens 200. `type` (muss ein vorhandener Elementtyp sein) und `label` Pflicht; `props` (Text), `geo` ([Breite, Länge]) und `link` (Beschriftung der Verknüpfung) optional.
- Gibt es ein Element mit gleichem Typ und gleicher Bezeichnung schon im Fall, wird es nur verknüpft, nicht doppelt angelegt.
- **note** (bis 5000 Zeichen) und **message** sind optional.

Vorlage: Python

```
import re

def transform(inp):
    wert = inp["value"]
    ergebnisse = []
    if "." in wert:
        ergebnisse.append({"type": "domain", "label": wert.split(".", 1)[1], "link":
"übergeordnet"})
    return {"entities": ergebnisse, "note": f"geprüft: {wert}", "message": f"{len(ergebnisse)}
Treffer"}
```

Verfügbar ist die Python-Standardbibliothek (re, json, datetime, hashlib, ipaddress, urllib.parse, …). Zusatzpakete und Netzzugriff gibt es in der Sandbox nicht – dafür einen lokalen Webdienst verwenden. Alternativ zu `transform` darf das Skript auch `json.load(sys.stdin)` lesen und das Ergebnis mit `print(json.dumps(...))` ausgeben.

Vorlage: JavaScript

```
function transform(input) {
  const teile = String(input.value).split('.')
  return { entities: [], note: `Labels: ${teile.length}`, message: 'fertig' }
}
```

`transform` darf auch `async` sein. `print(...)` bzw. `console.log(...)` schreiben in die Ausgabe. Kein `require`, kein `fetch` ins Netz, keine Dateien – reines JavaScript.

Vorlage: lokaler Webdienst mit FastAPI

```
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Eingabe(BaseModel):
    value: str
    type: str
    label: str | None = None
    props: dict = {}

@app.post("/transform")
def transform(e: Eingabe):
    return {"entities": [{"type": "note", "label": f"geprüft: {e.value}", "link": "Ergebnis"}],
"message": "ok"}
```

Starten: `py -3 -m pip install fastapi uvicorn` (einmalig), dann `py -3 -m uvicorn server:app --host 127.0.0.1 --port 8765`. plugin.json dazu: `"kind": "http", "url": "http://127.0.0.1:8765/transform"`.

Wann ein Webdienst statt Sandbox-Code? Wenn das Plugin ins Netz muss (eigene Abfragen), Zusatzpakete oder Modelle braucht, Zustand halten muss oder schon als Dienst im Team existiert. Der Dienst kann in jeder Sprache geschrieben sein – Hauptsache, er nimmt JSON per POST an und antwortet mit JSON.

Wo erscheinen Plugins?

- **Rechtsklick** auf passende Elemente (Graph, Eigenschaften, Tabellen) – mit Icon und Name.
- **Terminal:** `plugin` listet alle, `plugin <id> <wert> [--type domain]` führt aus; die neuen Elemente kommen als Tabelle zurück und lassen sich weiter verarbeiten, z.B. `plugin domain-teile mail.example.org | graph new "Plugin-Ergebnis"`.
- **Untersuchungs-Automatik:** als abwählbarer Schritt bei passenden Startwerten (nicht vorausgewählt).
- **Plugin-Bereich:** „▶ Ausprobieren“ zum Testen, „⇄ In den Fall“ zum Übernehmen.

Testen und Fehlersuche

- **Am schnellsten:** im Plugin-Bereich „▶ Ausprobieren“ – Fehlermeldungen aus Python (Traceback) und JavaScript erscheinen direkt.
- **Python lokal testen** (falls Python installiert ist): in einer Datei `test.py` `from main import transform; print(transform({"value": "example.org", "type": "domain", "props": {}}))`.
- **FastAPI lokal testen:** `curl -X POST http://127.0.0.1:8765/transform -H "Content-Type: application/json" -d "{\"value\":\"example.org\",\"type\":\"domain\"}"`
- **„kein gültiges JSON“:** `transform` gibt kein Objekt zurück bzw. das Skript druckt kein JSON.
- **„… mit unbekanntem Typ verworfen“:** Elementtyp gibt es nicht (Schreibweise prüfen, eigene Typen erst unter Typen & Symbole anlegen).
- **„Zeitlimit überschritten“:** `timeout` erhöhen oder die Arbeit beschleunigen.
- **„Dienst nicht erreichbar“:** Läuft der Webdienst auf genau der Adresse aus `url`?
- **„Plugin wurde geändert“:** nach jeder Dateiänderung erneut freigeben.

Freigabe und Protokoll

- Die Freigabe gilt für den genauen Stand aller Dateien des Plugins (Prüfsumme). Jede Änderung → neue Freigabe.
- Ein Plugin-Ordner darf höchstens **200 Dateien** enthalten (ohne `__pycache__`, `.venv`, `node_modules`, `.git`) – sonst lässt er sich nicht freigeben. Große Abhängigkeiten gehören nicht in den Plugin-Ordner.
- **Audit-Log** je Lauf: Plugin, Version, Prüfsumme, Elementtyp und Wert, Anzahl Ergebnisse, SHA-256 der Antwort, Laufzeit.
- Freigeben und Entziehen erfordert die Administrator-Rolle (Behörden-Edition).

Gute Praxis

- Ein Plugin, eine Aufgabe; klare Namen und Versionen (Version erhöhen bei Änderungen).
- Ergebnisse mit Herkunft anreichern (`props.quelle`, Link zur Fundstelle).
- Keine personenbezogenen Daten erfinden oder „schätzen“; im Zweifel nichts zurückgeben.
- Nur rechtmäßig zugängliche Quellen verwenden; Nutzungsbedingungen beachten.
- Geheimnisse nie in plugin.json oder den Code schreiben. Braucht eine Abfrage einen Schlüssel, einen **eigenen API-Konnektor** (Schlüssel im Vault) oder einen lokalen Dienst mit eigener Konfiguration

verwenden.

Plugins im Marktplatz veröffentlichen

Der Marktplatz auf **tangodown.eu** verteilt Plugins als `.tdplugin` – eine JSON-Datei mit `plugin.json` und den Textdateien des Plugins (nur `.js`, `.py`, `.json`, `.md`, `.txt`, `.csv`, flach, höchstens 50 Dateien und 2 MB):

```
{
  "format": "tdplugin/1",
  "manifest": { "id": "beispiel.whois", "name": "Whois-Kurzinfo", "version": "1.0", "kind": "js",
"main": "main.js", "inputs": ["domain"] },
  "files": { "main.js": "async function transform(i) { … }", "README.md": "# Whois-Kurzinfo" }
}
```

So kann ein Plugin in den Marktplatz kommen:

1. Plugin-Ordner fertigstellen, **selbst prüfen** (Code, angemeldete Rechte, Funktion) und in TangoDown testen.
2. Den Ordner mit einer kurzen Beschreibung und Anleitung (Deutsch, gern auch Englisch) an **info@tangodown.eu** schicken. Mit der Einsendung bestätigst du, dass du das Plugin selbst geprüft hast, dafür verantwortlich bist, die Rechte daran hast und es **kostenlos** bereitstellst.
3. Ob und wann ein Plugin aufgenommen wird, entscheidet der Herausgeber frei – es gibt **keinen Anspruch** auf Aufnahme. Aufgenommene Plugins werden gepackt und **signiert**.

Verantwortung: Für Code, Rechte, Funktion und Folgen eines Plugins ist allein sein **Autor** verantwortlich. Der Herausgeber prüft eingereichte Plugins nicht und übernimmt keine Verantwortung dafür – auch nicht bei Aufnahme in den Marktplatz. Die Signatur bestätigt nur, dass das Plugin aus dem Marktplatz stammt und unverändert ist; sie sagt nichts über Sicherheit oder Funktion.

Nur kostenlos: TangoDown ist kostenlos, und das gilt auch für Plugins: Plugins für TangoDown dürfen nur **unentgeltlich** weitergegeben werden – kein Verkauf, keine Bezahlschranke, keine kostenpflichtige Lizenz. Der Marktplatz nimmt nur kostenlose Plugins auf.

Offener Code: Nutzer sehen den Code jedes Plugins in TangoDown („⟨⟩ Code“) und im Marktplatz. Lege im README fest, unter welcher Lizenz andere dein Plugin verbessern und weitergeben dürfen (z.B. MIT) – so kann die Community mitprüfen und weiterentwickeln.

Sperren: Der Herausgeber kann Plugins jederzeit aus dem Marktplatz entfernen und in der Anwendung sperren, wenn er es für nötig hält.

Selbst anbieten: Du kannst dein Plugin auch ohne Marktplatz weitergeben, z.B. auf deiner eigenen Website oder bei GitHub – als Ordner oder als `.tdplugin`-Datei, ebenfalls kostenlos und in eigener Verantwortung.

TangoDown installiert aus dem Marktplatz nur Plugins, deren **Prüfsumme** stimmt und deren **Signatur** zum eingebauten Herausgeber-Schlüssel passt. Jede neue Fassung braucht wie immer eine neue Freigabe. Eine `.tdplugin`-Datei lässt sich auch ohne Marktplatz unter „Plugins“ → „🛒 Marktplatz“ → „📄 Aus Datei installieren …“ einspielen – zum Beispiel, um ein eigenes Plugin im Team weiterzugeben.

Register & geschützte Datenbanken anbinden

Für nicht öffentliche Quellen, für die eure Organisation berechtigt ist (Fachverfahren, interne Datenbanken). Einrichtung im Modul ⛨ **Register & Datenbanken** – ohne Programmierung. Passwörter, Tokens und Zertifikate werden verschlüsselt im Vault gespeichert, nie in der Konfiguration.

Die Einrichtungsmaske („ + Quelle einrichten“)

- **Art:** Fachverfahren (REST/HTTPS) oder Datenbank (SQL: PostgreSQL, Microsoft SQL Server, MySQL/MariaDB).
- **Adresse** mit `{q}` für den Suchwert; `http://` nur im eigenen Netz.
- **Anmeldung:** Client-Zertifikat (mTLS), Token, Benutzer/Passwort oder keine. Zertifikat (PFX/P12) und Passwort über „📜 Zertifikat …“ – sie landen im Vault, nie in der Konfiguration.
- **SQL:** eine lesende Abfrage mit `:q`; schreibende Befehle, mehrere Anweisungen und Systemprozeduren werden abgewiesen.
- **Felder übernehmen:** je Zeile `Bezeichnung = Pfad` – bei REST ein JSON-Pfad (`treffer[0].halter.name`), bei SQL der Spaltenname.
- **Für Elementtypen:** bei welchen Elementen die Quelle im Rechtsklick erscheint.
- **Verbindung testen:** prüft Erreichbarkeit, TLS und Zertifikat ohne Suchwert – es werden keine Personendaten abgefragt.

Beispiel 1: Fachverfahren mit Client-Zertifikat (mTLS)

- Adresse: `GET https://fachverfahren.intern/api/v2/halter?kennzeichen={q}`
- Anmeldung: „Client-Zertifikat (mTLS)“ → „📜 Zertifikat …“ (PFX/P12) und „Zert.-Passwort …“
- Kopfzeilen (optional): `X-Dienststelle: Musterstadt`
- Felder übernehmen:

```
Halter = treffer[0].halter.name
Anschrift = treffer[0].halter.anschrift
Fahrzeug = treffer[0].fahrzeug.typ
```

Erwartete Antwort (Beispiel): `{"treffer": [{"halter": {"name": "…", "anschrift": "…"}, "fahrzeug": {"typ": "…"}}]}`. TangoDown nimmt als Zeilen das erste Array der Antwort (oder das Objekt selbst).

Beispiel 2: POST mit JSON-Body und Token

- Adresse: `POST https://register.beispiel.org/suche`, Anmeldung „Token (Bearer)“ → „🔑 Token …“
- Body: `{"suchwert": "{q}", "art": "person"}` – {q} wird JSON-sicher eingesetzt

Beispiel 3: interne PostgreSQL-Datenbank (nur lesend)

```
SELECT p.name, p.geburtsdatum, a.strasse || ', ' || a.ort AS anschrift
FROM personen p JOIN anschriften a ON a.person_id = p.id
WHERE p.kennung = :q
```

Felder: `Name = name`, `Geburtsdatum = geburtsdatum`, `Anschrift = anschrift`. Datenbankkonto mit reinen Leserechten anlegen, z.B.:

```
CREATE ROLE tangodown_leser LOGIN PASSWORD '…';
GRANT CONNECT ON DATABASE register TO tangodown_leser;
GRANT SELECT ON personen, anschriften TO tangodown_leser;
```

Microsoft SQL Server: gleiche Abfrage mit :q (TangoDown setzt sie als Parameter @q ein, höchstens 200 Zeilen); MySQL/MariaDB ebenso.

Was TangoDown erzwingt

- Verschlüsselte Datenbank-Verbindungen prüfen das **Server-Zertifikat** (Standard). Nur für Server im eigenen Netz (z.B. `db.intern`, 10.x, 192.168.x) lässt sich „Zertifikat nicht prüfen (intern)“ anhaken – etwa bei selbst signierten Zertifikaten. Bei Servern im Internet wird das abgewiesen.
- Suchwerte gehen nur als Parameter in die Abfrage (keine SQL-Injektion), nur SELECT/WITH, eine Anweisung, Nur-Lese-Transaktion, 20 s Zeitlimit, 200 Zeilen.
- Nicht erlaubt (wird beim Speichern abgewiesen): Kommentare (`--`, `/* */`, `#`), `INTO`, Systemprozeduren und -funktionen (z.B. `xp_…`, `pg_read_file`, `OPENROWSET`, `dblink`), `SLEEP` / `WAITFOR`, `SET`, `LOCK`, `FOR UPDATE`. Text in Anführungszeichen wird dabei nicht mitgeprüft – `WHERE notiz = 'a;b'` geht.
- Beispiel, das geht: `SELECT halter, marke FROM fahrzeuge WHERE kennzeichen = :q` · Beispiel, das abgewiesen wird: `SELECT * INTO kopie FROM fahrzeuge WHERE kennzeichen = :q`.
- Jede Abfrage: Rechtsgrundlage + Begründung, im Fall mit Prüfsumme gespeichert, im Audit-Log; optional Vier-Augen-Freigabe je Quelle (Behörden-Edition).
- Schlüssel im Klartext in Kopfzeilen werden abgewiesen (`{key}` bzw. die Anmeldeart verwenden).

REST-API

Was bringt sie?

Eigene Skripte, Pipelines oder Werkzeuge können den **geöffneten Fall lesen** (Fallinfo, Elemente, Verknüpfungen, Audit-Log) und – wenn freigegeben – **Elemente und Verknüpfungen anlegen**. Typische Einsätze: Ergebnisse aus eigenen Werkzeugen in den Fall schieben, Falldaten für Auswertungen exportieren, Berichte in anderen Systemen erzeugen, Massenimporte.

Einschalten

1. **Werkzeuge → Vault & Konnektoren → Lokale Schnittstelle · REST & MCP**.
2. „aktiv“ einschalten; Port (Standard **47821**) bei Bedarf ändern.
3. ⧉ **curl-Beispiel** kopiert einen fertigen Aufruf mit Token in die Zwischenablage.
4. **Schreibzugriff** nur einschalten, wenn Programme Daten anlegen sollen (eigene Bestätigung).

Der Token wird verschlüsselt gespeichert und lässt sich dort erneuern („Erneuern“). Er gehört nicht in Skripte, die weitergegeben werden – besser aus einer Umgebungsvariable lesen.

Schutz der Schnittstelle:

- Erreichbar nur auf diesem Rechner (`127.0.0.1`), nicht aus dem Netz. Andere Benutzer und Programme auf demselben Rechner erreichen den Port aber – entscheidend ist deshalb der **Token**, nicht die Portnummer.
- Jede Anfrage braucht den Token (zufällig, 192 Bit); er wird in konstanter Zeit verglichen.
- Anfragen mit fremdem Host-Namen (Schutz gegen DNS-Rebinding) und aus Webseiten im Browser (`Origin`-Kopfzeile) werden abgewiesen.
- **Sperre nach Fehlversuchen:** Nach 10 Anfragen mit falschem oder fehlendem Token innerhalb einer Minute weist die Schnittstelle eine Minute lang **alle** Anfragen ab (HTTP 429) – auch mit richtigem Token.

TangoDown meldet das, und es steht im Audit-Log des offenen Falls. Ein Neustart der Schnittstelle in den Einstellungen hebt die Sperre auf. Im Zweifel den Token erneuern.

- In den Einstellungen steht, wie viele Anfragen abgewiesen wurden.
- „⧉ kopieren“ legt den Token in die Zwischenablage und leert sie nach 60 Sekunden wieder, sofern dort noch derselbe Text steht.
- Lesen nur mit Token; Schreiben zusätzlich nur mit eigener Freigabe; VS-NfD-Fälle nur mit dem On-Prem-Token. Jede Abfrage steht im Audit-Log.
- Die Verbindung ist unverschlüsseltes HTTP, verlässt aber den Rechner nicht (Loopback).

Adresse und Anmeldung

- Basis: `http://127.0.0.1:47821/api/v1`
- Kopfzeile: `Authorization: Bearer <TOKEN>`
- Antworten: JSON, UTF-8.

Endpunkte

- `GET /api/v1/case` – Aktenzeichen, Titel, Zweck, Rechtsgrundlage, Umfang des geöffneten Falls.
- `GET /api/v1/entities?type=domain&q=suchtext&limit=200` – Elemente auflisten (Typ und Suche optional, höchstens 500).
- `GET /api/v1/entities/<id oder Bezeichnung>` – ein Element mit Angaben, Quelle, Einstufung, Notizen, Verknüpfungen.
- `GET /api/v1/links` – alle Verknüpfungen.
- `GET /api/v1/audit?limit=50` – letzte Einträge des Audit-Logs.
- `GET /api/v1/types` – alle Elementtypen (eingebaut und eigene) mit Feldvorlagen (Schlüssel, Beschriftung, Feldart).
- `GET /api/v1/search?q=suchtext&limit=50` – im Fall suchen (Bezeichnung, Eigenschaften, Quelle, Notizen), mit kurzem Auszug.
- `GET /api/v1/export` – **Daten geben:** der ganze Fall als Paket (alle Elemente mit Eigenschaften/Koordinaten und alle Verknüpfungen) zum Weiterverarbeiten.
- `POST /api/v1/entities` (Schreibzugriff) – ein Element anlegen. Body: `{ "type": "domain", "label": "example.org", "rel": "C3", "source": "mein Skript", "props": { "notiz": "…" }, "notes": "…" }`
- `PATCH /api/v1/entities/<id>` (Schreibzugriff) – ein Element ändern. Body (alles optional): `{ "label": "…", "rel": "B2", "props": { "port": "443", "altesFeld": null }, "geo": [53.55, 9.99], "notes": "…" }`. In `props` setzt ein Wert das Feld, `null` entfernt es; `geo: null` entfernt die Koordinaten. (Auch per `POST` auf denselben Pfad.)
- `POST /api/v1/links` (Schreibzugriff) – Body: `{ "source": "<id>", "target": "<id>", "label": "gehört zu" }`
- `POST /api/v1/ingest` (Schreibzugriff) – **Daten empfangen:** mehrere Elemente und Verknüpfungen auf einmal (Massenimport aus einem anderen Werkzeug). Verknüpfungen verweisen per `"#0"`, `"#1"` … auf die neu angelegten Elemente (Reihenfolge) oder per vorhandener ID bzw. exakter Bezeichnung.

Angelegte Elemente tragen als Quelle „API · <Programm>“ und ohne Angabe die Einstufung F6; jede Änderung erscheint sofort in der Oberfläche und im Audit-Log.

> **Wohin importiert wird:** `POST /entities`, `POST /links` und `POST /ingest` schreiben in den **geöffneten Fall** (das Element-Netz im Graphen). Die fallübergreifenden **Eigenen Datenbanken** (z.B. eine benannte Fahrzeug-Liste) sprichst du getrennt über die `databases`-Endpunkte an (unten) – dort gezielt eine

> bestimmte Datenbank als Ziel/Quelle. Geschützte Quellen bindest du über **Register & Datenbanken** (Konnektor) an. Beim Abgleich (z.B. Fahrzeug) werden die Eigenen Datenbanken automatisch nach dem Suchwert durchsucht.

Eigene Datenbanken (benannte, fallübergreifende Listen)

Getrennt vom Fall-Graphen – ideal, um z.B. eine Fahrzeug- oder Personenliste per Skript zu füllen und gezielt abzufragen. Die Datenbank wird **per Name oder ID** angesprochen (Name ohne Beachtung der Groß-/Kleinschreibung).

- `GET /api/v1/databases` – alle eigenen Datenbanken (Name, Art, Felder, Anzahl).
- `GET /api/v1/databases/<Name|id>/entries?q=suchtext&limit=200` – Einträge einer bestimmten Datenbank, optional Teilstringsuche über Schlüssel und Felder.
- `POST /api/v1/databases/<Name|id>/entries` (Schreibzugriff) – Eintrag aufnehmen. Body: `{ "key": "HH-AB 000", "fields": { "Halter": "Max Muster" }, "note": "…" }` (`key` = Schlüsselwert: Kennzeichen, Name, Rufnummer …).
- `GET /api/v1/databases/search?q=wert` – einen Wert über **alle** eigenen Datenbanken suchen (wie der Abgleich), mit Angabe der Datenbank und exakt/Teiltreffer.

Beispiel (Python): in die Fahrzeug-Liste schreiben und gezielt abfragen

```
BASE = "http://127.0.0.1:47821/api/v1"
requests.post(f"{BASE}/databases/Fahrzeuge/entries", headers=H,
    json={"key": "HH-AB 000", "fields": {"Halter": "Max Muster", "Typ": "PKW"}})
treffer = requests.get(f"{BASE}/databases/Fahrzeuge/entries", headers=H, params={"q": "HH-
AB"}).json()
print(treffer["db"]["name"], len(treffer["entries"]), "Treffer")
```

Beispiele

PowerShell:

```
$h = @{ Authorization = "Bearer $env:TD_TOKEN" }
Invoke-RestMethod -Uri http://127.0.0.1:47821/api/v1/entities?type=domain -Headers $h
```

Python:

```
import os, requests

H = {"Authorization": f"Bearer {os.environ['TD_TOKEN']}"}
BASE = "http://127.0.0.1:47821/api/v1"

print(requests.get(f"{BASE}/case", headers=H).json())

# Ein Element anlegen und eine Eigenschaft nachtragen
neu = requests.post(f"{BASE}/entities", headers=H,
    json={"type": "ip", "label": "192.0.2.10", "source": "Import"}).json()
requests.patch(f"{BASE}/entities/" + neu["id"], headers=H,
    json={"props": {"offene ports": "443, 8443"}, "geo": [53.55, 9.99]})

# Daten empfangen: Treffer aus einem eigenen Werkzeug als Paket übernehmen
requests.post(f"{BASE}/ingest", headers=H, json={
    "entities": [
        {"type": "company", "label": "Beispiel AG", "rel": "B2"},
        {"type": "domain",  "label": "beispiel.example", "props": {"quelle": "mein crawler"}},
    ],
    "links": [{"source": "#0", "target": "#1", "label": "Website"}],  # #0 -> #1
})

# Daten geben: den ganzen Fall exportieren
fall = requests.get(f"{BASE}/export", headers=H).json()
print(len(fall["entities"]), "Elemente,", len(fall["links"]), "Verknüpfungen")
```

Massenimport mit curl:

```
curl -X POST http://127.0.0.1:47821/api/v1/ingest \
  -H "Authorization: Bearer $TD_TOKEN" -H "content-type: application/json" \
  -d '{"entities":[{"type":"ip","label":"198.51.100.7","props":{"asn":"AS64500"}}]}'
```

curl:

```
curl -H "Authorization: Bearer <TOKEN>" http://127.0.0.1:47821/api/v1/case
```

Fehlercodes

- **401** Token fehlt oder falsch · **403** Host nicht erlaubt, Browser-Zugriff, VS-NfD-Fall ohne On-Prem-Token oder kein Schreibzugriff · **404** unbekannter Pfad · **409** kein Fall geöffnet.

Schutz

- Nur auf diesem PC erreichbar (`127.0.0.1`).
- Anfragen aus Webbrowsern werden abgewiesen – keine Webseite kann den Fall auslesen.
- Standardmäßig aus; Schreibzugriff gesondert; VS-NfD-Fälle nur mit On-Prem-Token; jeder Zugriff im Audit-Log.

Vorhandene Elemente (nie still überschreiben): `create_entity` bzw. `POST /entities` und der Import legen ein Element nur neu an, wenn es noch keines mit gleichem Typ und Namen gibt. Sonst wird das vorhandene weiterverwendet: leere Felder werden ergänzt, abweichende Werte **nicht** überschrieben. Die Antwort meldet das mit `existing: true` , `filled` (ergänzte Felder) und `conflicts` (`field` , `old` , `next`); der Import meldet `entitiesExisting` und `conflicts` . Abweichungen stehen auch im Audit-Log.

On-Prem-Zugang (VS-NfD): Unter Vault & Konnektoren → Lokale Schnittstelle „On-Prem-Zugang" einschalten. TangoDown erzeugt einen **zweiten Token**. Wer mit diesem Token fragt, darf auch VS-NfD-Fälle lesen (und bei freigegebenem Schreibzugriff schreiben). Der normale Token bleibt für VS-NfD gesperrt – an ihm hängen oft Assistenten, die Daten in die Cloud schicken.

- Den On-Prem-Token nur in Programme eintragen, die vollständig im eigenen Haus laufen, z.B. ein lokales Sprachmodell (Ollama, vLLM) mit eigenem MCP-Client oder eigene Skripte.
- „⧉ MCP-JSON (On-Prem)" bzw. „⧉ curl (On-Prem)" kopieren die Konfiguration mit diesem Token.
- Jeder Zugriff steht im Audit-Log mit dem Vermerk `[On-Prem-Token]` . Ausschalten macht den Token sofort ungültig.

```
# Beispiel: VS-NfD-Fall mit dem On-Prem-Token abfragen
curl -H "Authorization: Bearer <On-Prem-Token>" http://127.0.0.1:47821/api/v1/case
```

MCP – KI-Assistenten anbinden

Was ist MCP und was bringt es?

Das **Model Context Protocol (MCP)** ist ein offener Standard, mit dem KI-Assistenten Werkzeuge nutzen. Über die MCP-Schnittstelle kann z.B. **Claude Code** oder **Claude Desktop** den geöffneten Fall lesen, Fragen dazu beantworten, Zusammenhänge finden, Berichte entwerfen – und, wenn freigegeben, Elemente, Verknüpfungen und Notizen anlegen. Der Assistent arbeitet dabei immer über TangoDown: mit Zweckbindung und Audit-Log.

Einrichten

1. Lokale Schnittstelle einschalten (siehe REST-API).
2. ⧉ **Claude Code-Befehl** kopiert den fertigen Befehl, z.B.:

```
claude mcp add --transport http tangodown http://127.0.0.1:47821/mcp --header "Authorization: Bearer
<TOKEN>"
```

1. Für andere Clients **MCP-JSON** kopieren und in deren Konfiguration eintragen:

```
{
  "mcpServers": {
    "tangodown": {
      "type": "http",
      "url": "http://127.0.0.1:47821/mcp",
      "headers": { "Authorization": "Bearer <TOKEN>" }
    }
  }
}
```

Transport: Streamable HTTP (JSON-Antworten).

Werkzeuge

Nur lesend (immer):

- `case_info` – Fall, Zweck, Rechtsgrundlage (der Assistent soll es zuerst aufrufen: der Zweck begrenzt die Recherche).
- `list_entities` – Elemente, optional nach Typ und Suchtext.
- `list_entity_types` – Typen mit Feldvorlagen.
- `get_entity` – ein Element vollständig.
- `list_links` – Verknüpfungen.
- `audit_log` – letzte Einträge.
- `search` – im Fall suchen (Bezeichnung, Eigenschaften, Quelle, Notizen), kurze Treffer mit Auszug.
- `export_case` – **Daten geben:** den ganzen Fall als Paket (Elemente + Verknüpfungen) zum Weiterverarbeiten.
- `list_databases` – eigene (fallübergreifende) Datenbanken auflisten.
- `query_database` – eine **benannte** eigene Datenbank durchsuchen (z.B. „Fahrzeuge").
- `search_databases` – einen Wert über **alle** eigenen Datenbanken suchen.

Schreibend (nur mit Schreibzugriff):

- `create_entity` – neues Element (Einstufung F6, Quelle „MCP").
- `create_link` – zwei Elemente verknüpfen.
- `add_note` – Notiz anhängen.
- `update_entity` – Element ändern: Eigenschaften setzen/entfernen (`props` , Wert `null` entfernt ein Feld), Bezeichnung, Einstufung, Koordinaten `[lat, lng]` (`null` entfernt sie), Notizen.
- `ingest` – **Daten empfangen:** mehrere Elemente und Verknüpfungen auf einmal. Verknüpfungen verweisen per `"#0"` , `"#1"` … auf die neu angelegten Elemente oder per vorhandener ID/Bezeichnung.
- `add_database_entry` – einen Eintrag in eine **benannte** eigene Datenbank aufnehmen (z.B. ein Fahrzeug in „Fahrzeuge"): `key` = Schlüsselwert, `fields` = weitere Spalten.

Beispiel-Aufträge an den Assistenten

- „Fasse den Fall zusammen und nenne offene Fragen."
- „Welche Domains und IPs hängen zusammen? Zeichne mir die Wege."
- „Prüfe die Notizen auf Widersprüche."
- „Lege für die drei genannten Firmen Elemente an und verknüpfe sie mit der Person." (Schreibzugriff)
- „Übernimm dieses JSON als Elemente und Verknüpfungen in den Fall." (ingest, Schreibzugriff)
- „Exportiere den Fall und erstelle mir daraus eine Tabelle der IPs mit offenen Ports." (export_case)

Schutz und Datenschutz

- Was der Assistent liest, geht an dessen Anbieter (bei Cloud-KI). Jeder MCP-Zugriff steht im Audit-Log.
- VS-NfD-Fälle nur mit dem On-Prem-Token – den nur einem Assistenten geben, der vollständig im eigenen

Haus läuft (lokales Modell, keine Cloud).
- KI-Aussagen sind Hinweise; Belege prüfen.

Cyber-Intelligence: Schlüssel der Datenquellen

Das Modul ◎ **Cyber-Intelligence** fragt zu einer IP/Domain mehrere Dienste passiv ab. Die Schlüssel liegen im Vault (Werkzeuge → Vault & Konnektoren), je Anbieter einer; „Testen“ prüft ihn:

- **GreyNoise** – Community-API (frei, ~50/Woche): Schlüssel aus dem GreyNoise-Konto.
- **AbuseIPDB** – frei bis 1000 Prüfungen/Tag: API-Schlüssel aus dem Konto.
- **VirusTotal** – Public API (4/min, 500/Tag, nicht kommerziell): API-Schlüssel.
- **urlscan.io** – Suche frei (öffentliche Scans sind sichtbar); Schlüssel optional für mehr Kontingent.
- **Netlas** – frei mit Limit; für Security-Integrationen bietet Netlas eine **Developer License** an (anfragen).
- **Censys** – Schlüssel als „API-ID:Secret“ hinterlegen (TangoDown bildet daraus die Basic-Auth).
- **Shodan InternetDB** – ohne Schlüssel (nur IPv4).

Alle Abfragen sind rein lesend (die Dienste haben bereits gescannt); TangoDown sendet nichts an die Zielsysteme. Ergebnisse werden zusammengeführt, Widersprüche markiert; jede Abfrage steht mit Prüfsumme im Audit-Log. Hinweise zur Weitergabe: Prüfe je Anbieter, ob die kostenlose API nur persönlich oder auch in weitergegebener Software genutzt werden darf.

Eigene API-Konnektoren (ohne Programmierung)

Was bringen sie?

Ein Konnektor bindet einen Webdienst an, der JSON liefert – z.B. eine Firmen-, Telefon- oder Bedrohungsdaten-API. Er erscheint als Rechtsklick-Transformation bei den gewählten Elementtypen; die Felder aus der Antwort werden als Angaben übernommen.

Einrichten

- **Werkzeuge → Vault & Konnektoren → Eigene Konnektoren → + Neuer Konnektor**.
- **URL-Vorlage** mit `{q}` für den Wert, z.B. `https://api.example.org/v1/lookup?q={q}`.
- **Kopfzeilen** dürfen `{key}` enthalten – der Schlüssel liegt im Vault, nie im Konnektor.
- **Felder:** Beschriftung = JSON-Pfad, z.B. `Land = data.country`, `Ports = ports`.
- „Testen“ zeigt die Antwort, bevor der Konnektor aktiv wird.

Schlüssel richtig eintragen

Im Header steht nur der Platzhalter, z.B. `Authorization: Bearer {key}`; der eigentliche Schlüssel kommt in das Feld „Schlüssel“ und wird verschlüsselt im Vault gespeichert. Ein Schlüssel im Klartext im Header wird abgewiesen (er stünde sonst in Konfiguration und Bildschirmfotos) – dann den Schlüssel beim Anbieter erneuern.

Konnektor oder Plugin?

- **Konnektor:** ein Webdienst, eine Abfrage, Felder übernehmen – ohne Code, Schlüssel im Vault, Netzweg und Kill-Switch greifen.
- **Plugin:** eigene Logik, mehrere Schritte, neue Elemente und Verknüpfungen, Offline-Auswertungen, eigene Bibliotheken oder Modelle.

Weitere Erweiterungswege

- **Terminal:** Befehle mit Pipes über typisierte Objekte, Variablen, Hintergrund-Aufträge, Makros, `replay`, geschwärzter Mitschnitt. Ideal für wiederkehrende Abläufe innerhalb von TangoDown.
- **Untersuchungs-Automatik:** fester Ablauf je Startwert, Schritte nach Freigabe – inklusive passender Plugins.
- **Eigene Typen und Felder:** Werkzeuge → Typen & Symbole; eigene Typen können in `inputs` von Plugins stehen.
- **Eigene Datenbanken:** benannte, fallübergreifende Listen mit eigenen Feldern, Wertsuche; automatisch im Abgleich (z.B. Fahrzeug, Person). Befüllen im Modul (auch CSV-Import) oder über die `databases`-Endpunkte der REST-API bzw. die MCP-Werkzeuge.
- **Fallpaket (.tdpack):** verschlüsselter Export und Import ganzer Fälle zwischen Arbeitsplätzen.

Updates

TangoDown sucht Updates nur auf Knopfdruck („Hilfe & Info“ → „⇪ Nach Updates suchen“) und installiert nie selbst. Die Versionsdatei kommt von **tangodown.eu** und ist vom Herausgeber **signiert**; TangoDown prüft die Signatur mit dem fest eingebauten öffentlichen Schlüssel. Passt sie nicht, wird kein Download angeboten.

Nach dem Herunterladen „✓ Download prüfen“ (SHA-256) und erst bei „Prüfsumme stimmt“ starten. Portable: neue EXE an dieselbe Stelle legen – der Datenordner daneben bleibt.

Warum Signatur statt nur Prüfsumme: Wer eine Website übernimmt, könnte Datei und Prüfsumme gemeinsam austauschen. Eine gültige Signatur kann er ohne den privaten Schlüssel des Herausgebers nicht erzeugen. MD5 wird bewusst nicht verwendet (gilt als gebrochen).

Sicherheit und Recht – Kurzfassung für Entwickler

- Nur rechtmäßig öffentlich zugängliche Quellen und Daten im Rahmen des Fallzwecks.
- Keine Geheimnisse in Code oder Konfiguration; Schlüssel gehören in den Vault.
- Nichts, was Zugangsschutz umgeht (kein Login-Umgehen, kein Scraping gegen Nutzungsbedingungen, keine ungesicherten Privatkameras).
- Ergebnisse sind Hinweise, keine Feststellungen – kennzeichnen, woher sie stammen.
- Alles, was in den Fall kommt oder ihn verlässt, steht im Audit-Log – auch Zugriffe über Plugins, REST-API und MCP. Einzige Ausnahme: Fälle, die in der PI-Edition ausdrücklich „ohne Protokollierung“ angelegt wurden; sie sind gekennzeichnet und haben keinen Prüfpfad. In der Behörden-Edition gibt es diese Ausnahme nicht.