SolutionCenter API Reference

REST API for managing FrameworX installations remotely — list/run/stop solutions, file ops, license activation, machine settings, server info. JWT-authenticated; activation-gated default-OFF in 10.1.5; pairs with the per-solution Runtime API on TServer ports 3101+.

Overview
Activation gate
Authentication
Scopes
Endpoints
Errors
Brand-neutrality and OEM hooks
OpenAPI 3.1 document
What is not in v1 (reserved for v2)
Deployment scenarios
Quick-start

Overview

The SolutionCenter API is the machine-wide management plane of FrameworX, exposed over HTTPS by TWebServices on port **10108**. Fleet operators, OEM administration tools, central monitoring solutions, and IT/DevOps automation use it to manage the solutions installed on a machine without touching the Designer or per-solution runtime ports.

Each FrameworX installation hosts the API once, regardless of how many solutions are installed or running on it. v1 is single-installation: each machine's API manages that machine's solutions. Cross-machine fleet aggregation is a v2 capability — orchestration today happens in the caller's tooling (Ansible, custom dashboards, PowerShell scripts) by fanning out to each machine's API.

The SolutionCenter API is one of two HTTP APIs FrameworX exposes:

API	Plane	Port	Server	Authentication
Runtime API	Data — tags, alarms, historian, datasets	3101 / 3201 / 3301 / 3401 (per profile)	TServer (per solution)	Bearer-GUID via `Security.LogOnAsync`
SolutionCenter API (this page)	Management — solutions, files, license, machine settings	10108	TWebServices (one per machine)	JWT (RFC 9068, `at+jwt`)

The two APIs are independent — different processes, different ports, different authentication. Integrating with one does not require the other.

Activation gate

The SolutionCenter API ships **activation-gated default-OFF** in 10.1.5 GA. With the gate off, every `/api/v1/...` call (except `/ping`, `/health`, and `/api/v1/openapi`) returns HTTP 503 with a `service-unavailable` Problem Details body. `/api/v1/openapi` stays reachable so the OpenAPI document can be inspected, but no functional endpoint can be called.

Enabling the API

Set `SolutionCenterApi.Enabled` to `true` in `MachineSettings/TWebServices.json` under the `appSettings` object, and restart the TWebServices service:

Error rendering macro 'code': Invalid value specified for parameter 'com.atlassian.confluence.ext.code.render.InvalidValueException'

```
{
  "appSettings": {
    "SolutionCenterApi": {
      "Enabled": true
    }
  }
}
```

Build-version safety net

Even with `Enabled = true`, the gate refuses to flip on if the running build is older than the constant `MinSafeBuildForSolutionCenterApi`. The constant ships at a deliberately impossible value in 10.1.5 GA so the API stays inert by default; it is bumped to a real build number in lockstep with a follow-up security workstream. When the gate refuses to flip, TWebServices logs a CRITICAL event to the Windows System event log and continues to return 503 on every API call.

Operators who accept the inherited risk on an older build can override the safety net by setting `SolutionCenterApi.AcceptRiskOnOldBuild = true` in the same `MachineSettings/TWebServices.json` file. This is a deliberate two-flag opt-in; flipping `Enabled` alone is insufficient.

Authentication

All API calls authenticate with a JWT (RFC 9068, header `typ: at+jwt`) presented as `Authorization: Bearer <jwt>`. Two issuance paths are supported.

OIDC primary path (humans)

Fleet operators authenticate against a corporate OIDC identity provider, and the SolutionCenter API mints a FrameworX-issued JWT on top of the IdP's identity claims:

1. Operator's browser hits `GET /api/v1/auth/oidc/login?provider=<name>&redirect=<url>`.
2. SolutionCenter API looks up the named provider in `MachineSettings/SolutionCenterApi-OidcProviders.json`, builds the IdP authorize URL with PKCE S256 + state, and returns a 302 to the IdP.
3. Operator authenticates at the IdP. The IdP redirects the browser back to `POST /api/v1/auth/oidc/callback` with the authorization code.
4. SolutionCenter API exchanges the code with the IdP, validates the resulting token, mints its own JWT carrying the operator's identity + scope set, and 302s the browser back to the original `redirect` URL with `?token=<jwt>` appended (or returns 200 with a JSON body containing `accessToken` if no `redirect` was supplied).
5. Subsequent API calls send `Authorization: Bearer <jwt>`.

OIDC providers are configured machine-wide in `MachineSettings/SolutionCenterApi-OidcProviders.json`. Client secrets live in that same file as plaintext, protected by file-system ACL (no separate secret store in v1; client-credentials grant and asymmetric signing are deferred to v2).

Service-account JWT (machine integrations)

For service-to-service callers (central monitoring solutions, OEM admin tools, scripted automation) there is no login round-trip. The OEM administrator generates a long-lived JWT signed with a key the SolutionCenter API trusts, and drops the credential file at `MachineSettings/SolutionCenterApi-Keys.json` on the target machine. From that point the caller presents the pre-issued JWT in every request:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6ImF0K2p3dCIsImtpZCI6Im9...
```

This mirrors the existing FrameworX AccessKey deployment shape that has been in production for embedded and IPC scenarios for years — the credential is a file on the box, rotated by replacing the file. JWT lifetime is OEM-controlled; days or weeks is typical.

JWT shape

Tokens are signed HS256 (symmetric, kid-scoped). The `kid` header lets OEMs rotate signing keys without an algorithm change. `iss` and `aud` are OEM-configurable (see Brand-neutrality and OEM hooks below).

Error rendering macro 'code': Invalid value specified for parameter 'com.atlassian.confluence.ext.code.render.InvalidValueException'

```
{
  "header": {
    "alg": "HS256",
    "typ": "at+jwt",
    "kid": "fleet-2026-q2"
  },
  "claims": {
    "iss":   "https://api.example.com",
    "aud":   "solutioncenter",
    "sub":   "alice@example.com",
    "iat":   1714324800,
    "exp":   1714328400,
    "scope": "installation:read installation:control",
    "groups": ["FleetOps"]
  }
}
```

Refresh

`POST /api/v1/auth/refresh` with an expiring or recently-expired JWT returns a new JWT with extended `exp`, provided the original is still inside the configurable refresh window (default 24h after expiry). Beyond that, humans re-run the OIDC dance and machine integrations rotate their service-account JWT.

Scopes

Authorization is JWT scope membership. Each endpoint declares the scope it requires; the validator middleware checks it before dispatching to the handler. Scopes do **not** subsume each other — holding `installation:admin` does not implicitly grant `installation:control`; callers that need both must hold both literally.

Scope	Grants
`installation:read`	Read-only queries: list solutions, solution info, server info, license info, installed protocols, installed product versions.

`installation:control`	Start and stop solutions; read and set the auto-start flag.
`installation:files`	File operations within the configured path allowlist: chunked upload/download, single-shot upload, existence checks, allowlist inspection.
`installation:license`	License operations: site code retrieval, license-key set, activation/deactivation by code, license info.
`installation:machine`	Read named MachineSettings files within the configured filename allowlist; list the allowlist.
`installation:admin`	Privileged operations: write MachineSettings files, list active runtime connections, drop a connection, delete a solution.

Insufficient scope returns HTTP 404 (not 403). This is intentional — it prevents the API from leaking the existence of resources the caller is not authorized to see (BOLA defence).

Endpoints

All endpoints live under `/api/v1/`. Times are ISO 8601 UTC with millisecond precision and a `Z` suffix on output; inputs must carry an explicit `Z` or numeric offset (local-naive timestamps are rejected). List responses use a truncation envelope; cursor pagination is deferred to v2.

Solution lifecycle

Method	URL	Scope	Purpose
GET	`/api/v1/solutions/`	`installation:read`	List solutions installed on this machine, one row per (solution, profile) pair.
GET	`/api/v1/solutions/{name}/info`	`installation:read`	Full solution metadata snapshot including the running-state predicate.
GET	`/api/v1/solutions/{name}/isrunning`	`installation:read`	Cheap boolean predicate: is this solution running, yes or no.
POST	`/api/v1/solutions/{name}/run`	`installation:control`	Start TServer for this solution under the requested profile.
POST	`/api/v1/solutions/{name}/stop`	`installation:control`	Stop TServer for this solution.
GET	`/api/v1/solutions/{name}/autostart`	`installation:read`	Read the auto-start flag for this solution.
PUT	`/api/v1/solutions/{name}/autostart`	`installation:control`	Set the auto-start flag for this solution.
GET	`/api/v1/solutions/{name}/routing`	`installation:read`	Multi-version routing resolution — which installed product version this solution will start under.

Example — `GET /api/v1/solutions/`

Error rendering macro 'code': Invalid value specified for parameter 'com.atlassian.confluence.ext.code.render.InvalidValueException'

```
{
  "solutions": [
    { "name": "PlantSCADA", "buildVersion": "build-127", "profile": "Production",  "status": "running", "port":
3101 },
    { "name": "PlantSCADA", "buildVersion": "build-127", "profile": "Development", "status": "stopped", "port":
null }
  ]
}
```

Example — `GET /api/v1/solutions/PlantSCADA/info`

Error rendering macro 'code': Invalid value specified for parameter 'com.atlassian.confluence.ext.code.render.InvalidValueException'

```
{
  "name": "PlantSCADA",
  "description": "Plant 1 SCADA",
  "productVersion": "fx-2020.1.5.2000",
  "productFamily": "Enterprise",
  "productModel": "Unlimited",
  "targetFramework": ".net 10.0",
  "buildVersion": "build-127",
  "dateModified": "2026-04-28T11:14:02.115Z",
  "dateLastOpen": "2026-04-28T08:01:30.044Z",
  "path": "C:\\Solutions\\PlantSCADA.dbsln",
  "autoStart": false,
  "isRunning": true,
  "runningProfiles": ["Production"]
}
```

Example — `GET /api/v1/solutions/PlantSCADA/isrunning`

Error rendering macro 'code': Invalid value specified for parameter 'com.atlassian.confluence.ext.code.render.InvalidValueException'

```
{ "name": "PlantSCADA", "isRunning": true }
```

Example — `POST /api/v1/solutions/PlantSCADA/run`

Error rendering macro 'code': Invalid value specified for parameter 'com.atlassian.confluence.ext.code.render.InvalidValueException'

```
Request body:
{ "profile": "Production" }

Response (HTTP 202):
{ "status": "starting", "statusUrl": "/api/v1/solutions/PlantSCADA/isrunning" }
```

Profile values map: `Production` = 0, `Development` = 1, `Validation` = 2, `Custom` = 3. Numeric values are accepted for forward compatibility.

Files

File operations are confined to a configurable path allowlist. The default allowlist covers the solutions and backup directories; operators add additional paths in `MachineSettings/TWebServices.json`. Path traversal (`..`), absolute paths outside the allowlist roots, and symbolic links are rejected with HTTP 400 `path-not-allowed`.

Method	URL	Scope	Purpose
GET	`/api/v1/files/allowed-paths`	`installation:files`	Returns the active path allowlist.
POST	`/api/v1/files/allowed-paths`	`installation:admin`	Add a search path to the persistent allowlist. Body: `{ "path": "..." }`. Idempotent — re-adding an existing path is a no-op. Returns 200 with `{ "path": "...", "status": "added" }`.
DELETE	`/api/v1/files/allowed-paths`	`installation:admin`	Remove a search path from the persistent allowlist. Body: `{ "path": "..." }` (path is in the body, not the URL, to mirror the POST shape and avoid URL-encoding hell with Windows paths). Idempotent — removing a path not in the list still returns 200 with `{ "path": "...", "status": "removed" }`.
GET	`/api/v1/files/exists`	`installation:files`	Existence check for a file or directory under the allowlist.
POST	`/api/v1/files/upload/start`	`installation:files`	Begin a chunked upload; returns a transfer ID.
POST	`/api/v1/files/upload/chunk`	`installation:files`	Upload one chunk of an in-progress transfer.
POST	`/api/v1/files/upload/finish`	`installation:files`	Finalize a chunked upload.
POST	`/api/v1/files/upload/cancel`	`installation:files`	Cancel an in-progress chunked upload.
POST	`/api/v1/files/upload/small`	`installation:files`	Single-shot upload for files under 4 MB.
POST	`/api/v1/files/download/start`	`installation:files`	Begin a chunked download; returns a transfer ID.
POST	`/api/v1/files/download/chunk`	`installation:files`	Read one chunk of an in-progress download.
POST	`/api/v1/files/download/finish`	`installation:files`	Finalize a chunked download.
POST	`/api/v1/files/download/cancel`	`installation:files`	Cancel an in-progress chunked download.
GET	`/api/v1/files/download/size`	`installation:files`	Pre-flight size check for a downloadable file.
POST	`/api/v1/files/delete-solution`	`installation:admin`	Delete a solution from the installation. Privileged.

The chunked upload/download dance is the standard four-call sequence: `start` returns a transfer ID, then any number of `chunk` calls reference that ID, then a single `finish` call commits, or a `cancel` call discards. Per-transfer caps and rate limits are enforced server-side; chunks arriving out of order produce HTTP 400 `chunk-out-of-order`; transfers that exceed the configured byte cap produce HTTP 413 `too-many-bytes`; an unknown transfer ID produces HTTP 404 `transfer-not-found`.

License

Method	URL	Scope	Purpose
GET	`/api/v1/license/site-code`	`installation:license`	Retrieve the machine's site code (needed for offline activation).
PUT	`/api/v1/license/key`	`installation:license`	Set the license key.
POST	`/api/v1/license/activate`	`installation:license`	Activate the license by activation code.
POST	`/api/v1/license/deactivate`	`installation:license`	Deactivate the license by activation code.
GET	`/api/v1/license/info`	`installation:license`	License metadata: edition, dates, limits.

Example — `POST /api/v1/license/activate`

Error rendering macro 'code': Invalid value specified for parameter 'com.atlassian.confluence.ext.code.render.InvalidValueException'

```
Request body:
{ "activationCode": "EXAMPLE-12345-67890-ABCDE-FGHIJ" }

Response (HTTP 200):
{ "status": "activated", "result": "..." }
```

A rejected key or activation code returns HTTP 400 with the `license-key-rejected` Problem Details type and the licensing service's reason in the `detail` field.

Machine settings

Reads and writes against files under `MachineSettings/` are confined to a configurable filename allowlist. Filenames outside the allowlist return HTTP 404 (not 403) so the API does not leak the existence of files the caller is not authorized to see. Writes require the higher-privilege `installation:admin` scope; reads only require `installation:machine`.

Method	URL	Scope	Purpose
GET	`/api/v1/machine/settings`	`installation:machine`	List the active filename allowlist.
GET	`/api/v1/machine/settings/{filename}`	`installation:machine`	Read the named MachineSettings file (returned as `application/json` for `.json` files, otherwise `text/plain`).
PUT	`/api/v1/machine/settings/{filename}`	`installation:admin`	Write the named MachineSettings file. Body is the file contents.

Server info and connections

Method	URL	Scope	Purpose
GET	`/api/v1/server/info`	`installation:read`	Hostname, OS, current UTC time, running-process count, server-info detail rows.
GET	`/api/v1/server/connections`	`installation:admin`	List active runtime client connections across all solutions on this machine.
DELETE	`/api/v1/server/connections/{guid}`	`installation:admin`	Force a runtime connection off. Logged for audit.
GET	`/api/v1/server/protocols`	`installation:read`	List installed communication-protocol drivers.
GET	`/api/v1/server/versions`	`installation:read`	List installed product versions on this machine (multi-version routing source).

Example — `GET /api/v1/server/info`

Error rendering macro 'code': Invalid value specified for parameter 'com.atlassian.confluence.ext.code.render.InvalidValueException'

```
{
  "hostname": "plant1-srv07",
  "os": "Microsoft Windows NT 10.0.20348.0",
  "processCount": 12,
  "processInfo": "...",
  "currentTimeUtc": "2026-04-28T11:42:11.003Z",
  "serverInfoRows": [
    { "Name": "Edition", "Value": "Enterprise" },
    { "Name": "Build",   "Value": "fx-2020.1.5.2000" }
  ]
}
```

The host's IP addresses are intentionally **not** exposed in this response. The caller already knows the address it connected to, and enumerating the host's interfaces would leak internal network topology to anyone holding `installation:read`.

Authentication endpoints

Method	URL	Scope	Purpose
GET	`/api/v1/auth/oidc/login`	*(none — pre-auth)*	Kick off the OIDC dance for a configured provider; returns 302 to the IdP's authorize URL.

POST	`/api/v1/auth/oidc/callback`	*(none — pre-auth)*	IdP callback target; exchanges the authorization code for a FrameworX-issued JWT.
POST	`/api/v1/auth/refresh`	*(uses the supplied JWT)*	Refresh an expiring or recently-expired JWT within the refresh window.

OpenAPI document

Method	URL	Scope	Purpose
GET	`/api/v1/openapi`	*(none — remains accessible even when the activation gate is OFF)*	OpenAPI 3.1 document describing every endpoint, scope, and response shape. Suitable for code generation.

Errors

All error responses use the RFC 7807 Problem Details envelope with `Content-Type: application/problem+json`:

Error rendering macro 'code': Invalid value specified for parameter 'com.atlassian.confluence.ext.code.render.InvalidValueException'

```
{
  "type":     "https://errors.example.com/solutioncenter-api/permission-denied",
  "title":    "Permission denied",
  "status":   403,
  "detail":   "JWT scope insufficient for this resource.",
  "instance": "/api/v1/server/connections"
}
```

The `type` URL base is OEM-configurable. Standard error type slugs:

HTTP status	Error type slug	When
400	`bad-request`	Malformed request, missing required parameter, wrong HTTP verb for the URL.
400	`path-not-allowed`	File operation references a path outside the allowlist or contains traversal segments.
400	`chunk-out-of-order`	Upload chunks arrived out of sequence.
400	`license-key-rejected`	Licensing service refused the supplied key or activation code.
401	`authentication-required`	Token missing, malformed, or signature invalid.
401	`session-expired`	JWT `exp` claim is in the past, or refresh window has been exceeded.
404	`not-found`	Resource does not exist, OR caller's scope set is insufficient (BOLA defence — intentional collapse to 404 instead of 403).
404	`solution-not-found`	Named solution is not installed on this machine.
404	`file-not-found`	Requested file does not exist within the allowlist.
404	`filename-not-allowed`	MachineSettings filename is outside the configured allowlist.
404	`transfer-not-found`	Chunked-transfer ID is unknown or expired.
413	`too-many-bytes`	Chunked transfer exceeded the configured per-transfer byte cap.
500	`internal-error`	Unhandled server-side condition; details in the response `detail` field and TraceLog.
503	`service-unavailable`	Activation gate is OFF, or the machine's build is older than `MinSafeBuildForSolutionCenterApi` without an explicit risk-acceptance override.

Brand-neutrality and OEM hooks

The SolutionCenter API is designed to be re-branded by OEMs without source edits. With default configuration, no response body contains the literal strings `FrameworX` or `Tatsoft` — this is verified by an automated test in the build. The following knobs are configurable per installation:

Surface	Default	Override
OpenAPI document title	`SolutionCenter API`	`SolutionCenterApi.OpenApiTitle`
Problem Details `type` URL base	`https://errors.example.com/solutioncenter-api/`	`SolutionCenterApi.ErrorTypeBase`
JWT `iss` claim	`https://api.example.com`	`SolutionCenterApi.Issuer`
JWT `aud` claim	`solutioncenter`	`SolutionCenterApi.Audience`
Service-account JWT signing keys	(none — ephemeral key generated on first run)	`MachineSettings/SolutionCenterApi-Keys.json`

OIDC providers + client secrets	(none — OIDC unavailable until at least one provider is enrolled)	`MachineSettings/SolutionCenterApi-OidcProviders.json`
File-operation path allowlist	Solutions and backup directories	`SolutionCenterApi.AllowedPaths[]`
MachineSettings filename allowlist	`TWebServices.json`, `TWebServicesRunPrograms.json`, `Logging.json`	`SolutionCenterApi.MachineSettingsAllowlist[]`
Activation gate	OFF	`SolutionCenterApi.Enabled` (with `AcceptRiskOnOldBuild` as the build-version override)
OpenAPI `servers[]` list	empty	`SolutionCenterApi.PublicBaseUrls[]`

The following are **not** configurable (they are part of the API contract):

- The `/api/v1/...` URL prefix.
- The JWT model (RFC 9068 `at+jwt`, HS256 in v1).
- The Problem Details envelope structure (RFC 7807).
- The set of scopes (additions are possible in v2; existing scopes will not lose meaning).
- The activation gate mechanism itself.

OpenAPI 3.1 document

`GET /api/v1/openapi` returns the OpenAPI 3.1 document describing every endpoint, scope, and response. All operation IDs are namespaced under `solutioncenter.*` to avoid collision with the Runtime API in shared tooling. The document is suitable as input to standard client generators (`openapi-generator-cli`, `NSwag`, `autorest`).

The OpenAPI endpoint stays reachable even when the activation gate is OFF, so downstream tooling can introspect the surface before operators flip the gate.

The OpenAPI document advertises base URLs via its `servers[]` array. By default this array is empty — client generators and Swagger UI then fall back to the host that served the document, which is correct in most local-network deployments. If the OpenAPI document will be consumed away from the API host (developer portals, partner SDK distributions, or reverse-proxy fronts where the externally-reachable hostname differs from the listener), populate `SolutionCenterApi.PublicBaseUrls` in `TWebServices.json` with the canonical external URL(s) — one entry per URL — so generated clients and rendered docs target the right host.

What is not in v1 (reserved for v2)

Two URL families are intentionally reserved and return HTTP 404 in v1, so callers that depend on them cannot mistake an unimplemented capability for a network failure or a permission issue:

- `/api/v1/installations/...` — reserved for multi-installation aggregation (a centralized SolutionCenter API instance proxying queries across many remote machines).
- `/api/v1/fleet/...` — reserved for fleet-wide rollups, agent registry, and heartbeat ingestion.

v1 is single-installation by design: each machine's API manages that machine's solutions. Operators with multiple installations call each machine's API directly from their own orchestration layer (Ansible, custom dashboards, PowerShell scripts). Multi-installation aggregation is a v2 capability when a real fleet operator drives the requirements.

Other capabilities deferred to v2: asymmetric JWT signing (RS256 / ES256) with a JWKS endpoint, OIDC client-credentials grant for service accounts, cursor pagination, audit-log query API, WebSocket push for live state, bulk multi-installation rollout.

Deployment scenarios

The SolutionCenter API ships in **both** TWebServices binaries on every FrameworX 10.1.5 installation, on the same port, with the same surface:

Binary	Path	Target framework	Use when
TWebServices (Windows .NET Framework)	`FactoryStudio\TWebServices.exe`	.NET Framework 4.8	Default Windows install. Runs on every supported Windows host without needing a separate runtime.
TWebServices (.NET 10)	`FactoryStudio\net10.0\TWebServices.exe`	.NET 10	Cross-platform deployments where the host already runs the .NET 10 runtime (Linux, container, or Windows hosts standardized on .NET 10).

Both binaries listen on port **10108** by default and expose identical `/api/v1/...` behaviour. An installation runs one or the other — not both at once on the same machine.

Quick-start

The examples below assume the activation gate is ON, a service-account JWT is in place, and TLS is terminated either by TWebServices or by an upstream reverse proxy. Replace `plant1-srv07` with the target host and `$jwt` with the actual token.

PowerShell — list solutions

```
$jwt = Get-Content -Path "C:\secure\fleet-jwt.txt" -Raw

$headers = @{ Authorization = "Bearer $jwt" }

$resp = Invoke-RestMethod `
    -Uri    "https://plant1-srv07:10108/api/v1/solutions/" `
    -Method GET `
    -Headers $headers

$resp.solutions | Format-Table name, profile, status, port
```

curl — check whether a solution is running

```
curl --silent \
     --header "Authorization: Bearer $JWT" \
     "https://plant1-srv07:10108/api/v1/solutions/PlantSCADA/isrunning"

# {"name":"PlantSCADA","isRunning":true}
```

Python — start a solution

```
import os, requests

jwt  = os.environ["FLEET_JWT"]
host = "plant1-srv07"

resp = requests.post(
    f"https://{host}:10108/api/v1/solutions/PlantSCADA/run",
    headers={"Authorization": f"Bearer {jwt}"},
    json={"profile": "Production"},
    timeout=30,
)

resp.raise_for_status()
print(resp.json())
# {"status": "starting", "statusUrl": "/api/v1/solutions/PlantSCADA/isrunning"}
```

In this section…