# Townlight Station Manual

Windows x64 Beta 1 · MSI 1.7.0.1

The signed Windows x64 beta 1.7.0.1 passed the documented fresh Windows Sandbox checks with synthetic data.


- Townlight Station Manual
  - Using this manual
  - Beta identity
  - First run
  - Daily workflow
  - Connected workflows
  - Architecture
  - Scheduling playout and recovery
  - Technical specifications
  - Configuration and administration
  - API and integration examples
  - MCP and automation boundary
  - Security and permissions
  - Backup recovery and maintenance
  - Troubleshooting
  - Verified workflows and limits
  - Terms of service and license
  - Glossary
  - Alphabetical subject index
  - Source and document control
  - Completed recordings on YouTube and live CivicCast coverage
  - Delivery subject index


## Townlight Station Manual

The signed Windows x64 beta 1.7.0.1 passed the documented fresh Windows Sandbox checks with synthetic data. The Windows acceptance summary applies to this exact MSI.

### Using this manual

Townlight Station helps station operators, clerks, records staff and deployment administrators. Station adapts a pinned CivicCast media implementation into an independent Townlight boundary. It captures managed files or configured live UDP MPEG-TS, retains canonical meeting/agenda context, and hands verified recording evidence to Records. Scheduling, playout, approval and local playback have distinct controls.

Staff should begin with First run and Daily workflow. Administrators should also read Architecture, Technical specifications, Configuration, Security and Recovery. Developers should use the exact API/data dictionary and examples. The table of contents and alphabetical subject index link to the relevant sections in the HTML edition; the PDF preserves headings and searchable text. An editable Word edition is included with the same documentation companion.

## Beta identity

| Identity | Value |
|---|---|
| Documentation | Townlight naming, Access and Notice candidate |
| Intended installer | 1.7.0.1 |
| Desktop source | 1.7.0-1 |
| Runtime distribution | townlight-runtime 0.2.0 |
| Station distribution/import | townlight-station 0.3.0 / townlight_station |
| Acceptance composition | Fresh Windows x64; inspections-preview; Station configured separately |
| Candidate hashes/signature | SHA-256 checksums and Windows acceptance summary |
| Owned software license | AGPL-3.0-only |

This edition updates the paired-runtime instructions for the naming candidate. It does not rename historical Station artifacts or claim new Station software features. Use an empty database and fresh profile; no in-place legacy upgrade or old import shim is supported. Preserve existing city data and the prior release. The approved profile path remains `%LOCALAPPDATA%\CivicSuite` for that path only.

## First run

1. Download the signed MSI, matching source companion and checksums from the public Windows release.
2. Verify the MSI hash and Authenticode publisher before installing on a fresh Windows profile. See the Windows acceptance summary for the tested setup.
3. Launch Townlight under the intended named Windows account. Choose the storage folders. Select **Townlight Inspections candidate** (`inspections-preview`) for the bounded desktop composition, including Access and Notice. A narrower default selection will not expose all workspaces.
4. Enter the city profile, time zone and contacts. Create the first named local administrator, sign in and choose protected backup storage. **Continue without local AI** is valid for the deterministic workflows.
5. Run Health Check and wait for readiness. Open Townlight. In System Health select **Finish Records Migration** to activate the canonical Python/PostgreSQL runtime. Do not fabricate migration receipts or seed accounts.

6. Create named staff through Settings. Import authorized sources using each product's source library. All examples in this package are synthetic; they are not municipal authority.
7. Provision Station separately if it is part of the beta exercise. Selecting the desktop composition does not start or configure Station. Configure vision explicitly only if automatic photo drafting is selected.

## Daily workflow

### 1. Prepare meeting context

Create and save the actual meeting and agenda in Meetings. Assign each selected agenda item to that meeting. A recording request cannot substitute an editable meeting label for the canonical records.

### 2. Open the operator

Sign in as authorized staff. Use the configured local operator command with the explicit native profile root. It obtains a short-lived one-use browser ticket; the native proof is not placed in the URL.

### 3. Choose the input

Select the meeting, assigned agenda and administrator-configured live-source ID. For recorded-file work, place the source in the managed ingest directory and optionally supply an SRT sidecar. Callers cannot submit arbitrary pipelines or filesystem paths.

### 4. Capture and stop

Start capture and distinguish waiting for a signal from active recording. Stop requests pipeline drain. Wait for completed status and validated output; a partial, failed or interrupted attempt is not a completed recording.

### 5. Review recording evidence

Inspect the media and retained context/hash. Retry is explicit for failed/interrupted recorded-file jobs. A lost live signal cannot be reconstructed: begin a new live recording and preserve the failed attempt.

### 6. Hand off to Records

Choose an existing Records request and attach the reviewed recording. The receiver checks bytes/hash and retains a locked private original. Repeating the same handoff retains one canonical document.

### 7. Use playout deliberately

Scheduling, dry-run/commit playout, staff approval and local publication are distinct API operations. Public playback only serves the separately approved packaged output. Nothing in capture or Records handoff automatically publishes it.

### 8. Close and recover

Sign out of the operator. Back up Station's database and media evidence together, separately from Core's native profile backup. Restart interrupted work deliberately; do not automatically re-air a lost signal.

### Working safely

Use your own named account and save before leaving a screen. A saved draft is durable; unsaved form text may not be. If a write has an unknown result, reopen its record before repeating it.

Read source jurisdiction, dates, citation, version and visibility. A copied source hash identifies exact bytes, not their legal applicability. If a source changes or permission is removed, resolve the source problem rather than substituting stale cached content.

Keep originals and derivatives separate. Review the intended audience before export. A public summary excludes private fields only through the applicable copy contract and your review. A renamed private packet is still private content.

Drafting, review, approval, attachment, export, fulfillment and publication are separate operations. A Townlight workpaper does not itself adopt law, issue a permit, impose a fine or release a records response. The responsible staff member records the applicable decision.

## Connected workflows

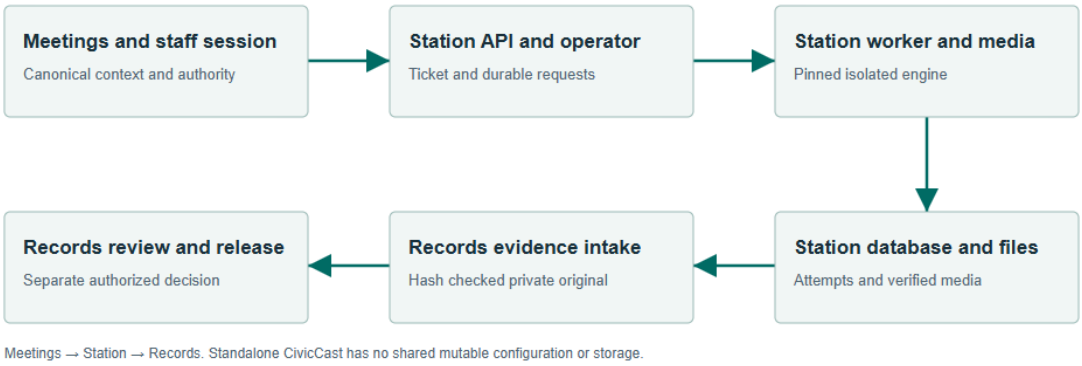
Meetings → Station → Records is the accepted recording workflow. Townlight owns current session authority; Station owns its database, storage and media children. Standalone CivicCast remains separate, with no shared configuration, service, data directory or automatic updater fallback.

| Direction | Material |
|---|---|
| Input | A real Townlight staff session, saved Meetings records, a dedicated Station database/data root, a managed ingest filename or configured live-source ID, and optional supplied SRT captions for recorded-file input. |
| Output | Completed verified recording derivatives, retained original inputs/provenance, separately approved local playback packages and locked private Records evidence. |

### Product boundary

There is no Station first-run wizard or automatic activation by the desktop composition. The accepted normal-profile path uses an explicitly configured paired Townlight HTTP process. Physical encoder/network, remote identity, SCM commissioning, CDN delivery and long-duration field quality are not certified. Station does not perform automatic meeting speech recognition.

# Architecture




Meetings → Station → Records. Standalone CivicCast has no shared mutable configuration or storage.

The drawing shows the principal staff/evidence path. Native commands carry trusted identity to Python business services; PostgreSQL and retained original files hold durable state. A downstream copy does not silently transfer decision authority. Station has its own configured authority/media boundary, database and mutable files. The connected-workflow text supplies the drawing's reading order and accessibility alternative.

# Scheduling playout and recovery

## Schedule a recording and interpret its state

Use **Schedules** after saving the actual meeting/agenda and placing a complete file in Station's managed `ingest` directory. Choose the meeting, **Start** time, **Source file in managed ingest**, and optional SRT **Caption sidecar**, then select **Schedule**. The browser form schedules files. The service's schedule API also supports a configured live-source ID; this is not a second input selector in that browser form.

The browser's datetime input is interpreted in the operator computer's local timezone and converted to an ISO UTC instant. Verify that computer's timezone before entering a city meeting time. A direct API client should send an explicit timezone offset or `Z`; a timezone-less API datetime is interpreted as UTC. A synthetic `POST /v1/schedules` body is:

```
{"meeting_id":"11111111-1111-4111-8111-111111111111","agenda_ids":
[],"source_file":"synthetic-program.mp4","captions_file":"synthetic-
captions.srt","start_at":"2026-11-05T18:00:00-
07:00","idempotency_key":"synthetic-schedule-001"}
```

Use a real synthetic meeting ID and intended future exercise time, with files already in managed ingest. The body selects exactly one file or configured live source; a live source cannot use a sidecar. Reusing an idempotency key with changed input/context is a conflict. The initial response includes `id`, `status: scheduled` and `start_at`; it does not mean recording has finished.

Refresh Schedules and inspect the returned identity/time. **Cancel schedule** is available while it remains scheduled; cancellation of an already dispatched or terminal entry is refused. The worker claims due work once and creates the linked recording. A start delayed by **more than ten**

**minutes** is `missed`, not replayed late. A start within that grace can dispatch on the next worker pass. Inspect the linked recording for its own queued/running/completed or failed state. Do not assume scheduling also commits that recording to air.

## Apply the caption profile

For a managed recorded file, an explicitly supplied SRT sidecar replaces embedded captions for this profile; otherwise embedded captions are retained where supported by the pinned media profile. Live capture preserves the embedded-caption path and does not accept an SRT sidecar. An absent caption stream is not automatic speech recognition. Check the actual captured/packaged captions for the selected synthetic input.

Original files are limited to 2,147,483,647 bytes and sidecars to 5,000,000 bytes. After publication, browser playback uses the packaged MP4 and a separate WebVTT track when produced. Preserve original inputs and recorded caption provenance separately from those derivatives.

## Check playout and commit deliberately

Open a **completed** recording. Select **Check playout** to queue `mode: dry-run`. Inspect its status and retained errors; a successful check does not change on-air state. To render and take that program to the scheduled on-air path, explicitly select **Commit to air**, which uses `mode: commit`. Wait for the playout to be `complete`; failed rendering is not a successful switch.

Equivalent `POST /v1/playouts` input is:

```
{"recording_id":"22222222-2222-4222-8222-222222222222","mode":"dry-run","idempotency_key":"synthetic-playout-check-001"}
```

Replace the recording ID with completed synthetic evidence. A separate deliberate commit uses `mode: commit` and a new key; do not reuse the check's key with a changed mode. Inspect the returned playout `id` and read `/v1/playouts/{id}` until terminal. The scheduled on-air program follows verified committed output, not every dry-run or every recording. Keep the worker and air roles distinct: one renders work, the other follows durable on-air state and emits the local program.

## Take a live override and return

In **On air**, choose an administrator-configured live input and enter a reason of at least three characters. Select **Take live override** and inspect the on-air mode/source. `POST /v1/on-air/live` uses `live_source_id` and `reason`; callers cannot supply an arbitrary network pipeline.

While live, select **Return to schedule** (`POST /v1/on-air/return-to-schedule` with no payload). Confirm the mode returns to scheduled program. The override supersedes the scheduled output; it does not delete its committed material. If no valid committed program is available, do not assume returning has produced a usable broadcast. Read actual output/state and any error. These are local synthetic on-air controls, not proof of a commissioned cable headend, physical receiver or public CDN.

## Approve and publish local playback

In **Playout and playback**, wait for a completed rendered playout. Review its actual media/context/captions, then enter the approval note and approve it. The API accepts a note of 3 –500 characters at `/v1/playouts/{id}/approve`. Approval is retained with the authenticated staff actor; it does not itself publish.

Select **Publish locally** only for that reviewed approved playout. `/v1/playouts/{id}/publish` packages local MP4/WebVTT playback. The response and subsequent playout view retain the publication details. Open the displayed local playback, verify it advances with the expected picture/sound, and inspect the caption track where supplied. Unpublished content is unavailable through the public playback route; guessing its ID does not authorize access. Repeating a completed publication returns the retained package rather than silently publishing changed media.

The local playback route `/v1/public/playback/{id}` serves the packaged program; `/captions` serves its available WebVTT. This approval/publication is separate from **Commit to air**, capture, attachment to Records and a Records release. Internet commissioning remains outside this local procedure.

## Recover without replaying unknown work

After a worker/API/air interruption, inspect durable schedules, recordings, attempts, playouts and on-air state before repeating any write. The worker marks interrupted in-flight work accordingly. Eligible failed/interrupted **file** recording retries are explicit; live loss requires a new live capture. Retain the prior attempt/logs and do not claim a missing live interval was reconstructed.

Future scheduled intent survives restart. A claimed/dispatched schedule must not be duplicated; one beyond the ten-minute grace becomes missed. Read its linked recording and create new future intent only when deliberately required. On-air recovery preserves attribution and does not automatically resume a stale live override as if the source had remained valid. Check the actual on-air mode and available committed program after restart before choosing a new override.

Visible failed/interrupted states and error text in the operator are the existing alert mechanism for this beta. Preserve the private process logs and attempt evidence. Reopen a normal operator ticket after API restart; do not replay an old ticket or bypass current staff authority. Restore Station’s database and its matching media as one recovery set, separately from Core’s native backup. Recheck original/output hashes and intended state after restoration. No automatic repair or database reset is implied by restart.

Source contracts: `townlight_station/operations.py`, `worker.py`, `airplay.py`, `operator.py`, the pinned media profile and `station/operator-ui/src/operator.tsx`. Service request models are in the Station API companion. The three suite-side Station paths in its OpenAPI download are not the full separate `/v1` service schema.

# Technical specifications

## Runtime and storage

The accepted package targets **Windows x64 fresh installation**. Its native shell uses **Rust 2021, Tauri 2 and Microsoft WebView2**; browser assets are JavaScript built with **Vite 8.1.6**. This describes the native desktop frontend; Station has a separate React operator, specified below or in its product manual. Canonical business workflows use bundled **CPython 3.13.5, FastAPI, Pydantic, SQLAlchemy and Alembic**, with **PostgreSQL 17 / PostGIS** and managed original files. The installed runtime tested here contains FastAPI 0.162.0. Package minimum Python requirements can differ from the bundled executable; the metadata reference preserves both. No stack rewrite is implied by the beta label.

The native transport is fixed to loopback `127.0.0.1:15480`; PostgreSQL uses the configured local runtime port (default `15432`). Optional local model service defaults to `127.0.0.1:15434`. Those are deployment defaults, not internet endpoints. The runtime has health, module and readiness endpoints; readiness is stronger than process existence. The task queue is the existing local Core task infrastructure, not an assertion that a separate Redis service is required for this desktop composition.

The default native profile is `%LOCALAPPDATA%\CivicSuite`; the historical directory name remains part of compatibility. Do not rename it to Townlight. A configured `TOWNLIGHT_PROFILE_ROOT` must be absolute. Local configuration and secrets live under that profile; protected managed files and database state must be backed up coherently. Source/runtime identity is distinct from a product's display label.

## Data and consistency

Configured live-source IDs; canonical meeting/agenda snapshots; queued recordings and immutable attempts; input/output hashes; schedules; playout jobs and approvals; durable on-air state; publication packages; Records attachment provenance.

Versioned saves use expected prior revision identifiers where defined. Reopen and reconcile on conflict. Exact exports and handoffs use source IDs, current visibility, revision and SHA256. Review belongs to the saved revision and does not transfer silently to edited text. Database changes use the owning product's migrations; a request or readiness check is not an authorization to mutate schema.

## Capacity and platform statements

This beta has synthetic single-profile workflow evidence. It does not have a published benchmark for concurrent staff capacity, minimum RAM/CPU, large-source limits or a field SLA. Do not turn a developer machine's specification into a supported minimum. Installer size is 2,092,122,112 bytes before installation; allow additional disk for extracted runtime, database, originals, exports, media and backups. Select equipment/storage after sizing those actual workloads.

The exact runtime payload lock records pinned files, sizes and hashes. The component metadata records shipped dependency declarations. These references are more precise than a manually

copied dependency list. Third-party and historical license notices must accompany their matching payloads.

## Station operator and media components

The native desktop frontend above is distinct from Station's browser operator. The pinned operator package `townlight-station-operator` is version **0.1.0**, using **React/react-dom 19.1.1**, **TypeScript 5.9.2**, **Vite 7.3.7** and the React Vite plugin 5.0.2. Station serves its built assets through its own operator API. Its Python control/API package remains **townlight-station 0.3.0**, while its retained storage identity remains **0.1.0**. These identities have different purposes and are not renumbered.

The separate pinned CivicCast-derived media engine uses Python/native GStreamer processing and bounded media children. It is not the standalone CivicCast application. Station's control database, data root, service/configuration and update identity remain independent. This describes the current mixed stack; no runtime rewrite or CivicCast migration is implied.

## Station media and lifecycle contract

The package/API version is `0.3.0`. Its retained storage identity has version `0.1.0`; that is a separate storage marker and must not be rewritten to match the package. Product/app IDs are `townlight-station` and `org.townlight.station`; service identity is `TownlightStation`, virtual account `NT SERVICE\TownlightStation`, credential namespace `townlight.station`, database schema `station`, and update-channel identity `townlight-station`. An identity name or SCM entry point does not establish that a Windows service was registered.

The control service is Python; its separate pinned CivicCast-derived engine uses native GStreamer media processing. There is no launch of the standalone CivicCast application. Default encoding is 1920×1080, 30 fps and 6000 kbps; explicit validated configuration may change it. The source lock is verified before use. Media children are isolated and bounded; unfinished attempts become interrupted on recovery.

Recorded-file requests accept a managed ingest filename, optional SRT sidecar and idempotency key. Live requests accept a configured source ID, not a path. Select exactly one file or live source; live capture cannot also supply a sidecar. Inputs are bounded by the configured two-hour default timeout and a 2,147,483,647-byte file limit. Supplied SRT sidecars are bounded to 5,000,000 bytes. The queue admits at most 100 pending/running jobs; live admission is further serialized. A repeated idempotency key must identify the same source bytes, canonical meeting context and encoding profile.

`queued` and `running` do not make content available as completed. Completion requires the worker's clean media teardown and stream validation. Failed/interrupted file jobs may be explicitly retried; a live gap requires a new capture. Records receives verified bytes and canonical provenance as a locked private original. Local publication is an explicit later operation on separately approved playout; its MP4/WebVTT package is distinct from the retained recording and from a Records release.

# Configuration and administration

## Prepare the normally activated synthetic profile

This procedure describes a future isolated beta installation. Documentation generation does not execute it. Complete normal Townlight Setup under the intended Windows account, select `inspections-preview`, sign in, and use **Finish Records Migration**. Keep the native database running and confirm the canonical runtime is ready. Use that account's actual profile path; the default is `%LOCALAPPDATA%\CivicSuite`. Do not create session files, seed accounts or force `domain_enabled=True`.

Station needs its own database and initially empty absolute data directory. It uses the same named Townlight staff authority through an additional configured loopback suite process. It does not launch or modify CivicCast. No SCM installation, registry edit or replacement of the desktop's fixed-origin service is needed.

The examples reserve suite port **15481**, Station port **8756** and synthetic UDP input port **19000**. Choose unused ports and change both configured origins together if needed. Never point these instructions at a city's live profile or data. Use the actual installation directory if Townlight was installed elsewhere.

## Store the independent test configuration

Run the following in a deployment PowerShell session under the same Windows user. Keep this session for startup and shutdown. The dedicated parent holds configuration/logs; its `data` child is Station's initially empty managed store.

```
$profilePath = Join-Path $env:LOCALAPPDATA 'CivicSuite'
$control = Join-Path $profilePath 'runtime\python\python.exe'
$payload = 'C:\Program Files\Townlight\_up_\runtime\payload\station'
$stationRoot = Join-Path $env:LOCALAPPDATA 'TownlightStation-Synthetic'
if (-not (Test-Path -LiteralPath $control)) { throw 'Complete normal
Townlight setup first.' }
if (Test-Path -LiteralPath $stationRoot) { throw 'Existing Station directory:
inspect it; do not overwrite.' }
New-Item -ItemType Directory -Path $stationRoot | Out-Null
$account = [System.Security.Principal.WindowsIdentity]::GetCurrent().Name
& icacls.exe $stationRoot /inheritance:r /grant:r "${account}:(OI)(CI)F"
'*S-1-5-18:(OI)(CI)F' | Out-Null
if ($LASTEXITCODE -ne 0) { throw 'Cannot protect Station configuration
directory.' }
New-Item -ItemType Directory -Path (Join-Path $stationRoot 'data'),(Join-Path
$stationRoot 'logs') | Out-Null
$rng = [System.Security.Cryptography.RandomNumberGenerator]::Create()
$randomBytes = New-Object byte[] 32
$rng.GetBytes($randomBytes)
$rng.Dispose()
$stationPassword = -join ($randomBytes | ForEach-Object { $_.ToString('x2')
})
$stationPassword | ConvertTo-SecureString -AsPlainText -Force | ConvertFrom-
```

```
SecureString |
  Set-Content -LiteralPath (Join-Path $stationRoot 'station-password.dpapi')
Remove-Variable stationPassword,randomBytes
```

The password file uses the current Windows user's DPAPI protection. It is not portable to a different user or machine without an explicit credential/recovery decision. Keep it out of source control, support bundles and public documentation. Reuse this protected configuration on restart rather than creating another database or password.

In the same session, establish the exact process environment. On a later restart, set `$profilePath`, `$control`, `$payload` and `$stationRoot` to these same existing locations, then repeat this environment/start section, **not** the first-time provisioning section.

```
$secret = Get-Content -LiteralPath (Join-Path $stationRoot 'station-
password.dpapi') -Raw |
  ConvertTo-SecureString
$stationPassword = [System.Net.NetworkCredential]::new('', $secret).Password
$nativePassword = (Get-Content -LiteralPath (Join-Path $profilePath
'config\secrets\postgres-password.txt') -Raw).Trim()
$encodedNative = [uri]::EscapeDataString($nativePassword)
$env:TOWNLIGHT_PROFILE_ROOT = $profilePath
$env:TOWNLIGHT_DATA_DIR = Join-Path $profilePath 'Data'
$env:TOWNLIGHT_FILE_STORAGE_DIR = Join-Path $profilePath 'Data\files'
$env:DATABASE_URL = "postgresql+asyncpg://townlight:${encodedNative}
@127.0.0.1:15432/townlight"
$env:TOWNLIGHT_PRODUCT_PROFILE = 'inspections-preview'
$env:PORTAL_MODE = 'private'
$env:TOWNLIGHT_RUNTIME_HOST = '127.0.0.1'
$env:TOWNLIGHT_RUNTIME_PORT = '15481'
$env:TOWNLIGHT_STATION_URL = 'http://127.0.0.1:8756'
$env:TOWNLIGHT_STATION_SUITE_URL = 'http://127.0.0.1:15481'
$env:TOWNLIGHT_STATION_DATA_ROOT = Join-Path $stationRoot 'data'
$env:TOWNLIGHT_STATION_DATABASE_URL =
"postgresql+psycopg2://townlight_beta_station_app:${stationPassword}
@127.0.0.1:15432/townlight_beta_station"
$env:TOWNLIGHT_STATION_MEDIA_SOURCE = Join-Path $payload 'media-source'
$env:TOWNLIGHT_STATION_MEDIA_PYTHON = Join-Path $payload 'runtime\python.exe'
$env:TOWNLIGHT_STATION_MEDIA_RUNTIME = Join-Path $payload 'runtime'
$env:TOWNLIGHT_STATION_FFPROBE = Join-Path $payload 'bin\ffprobe.exe'
$env:PYTHONNOUSERSITE = '1'
$env:PYTHONDONTWRITEBYTECODE = '1'
Remove-Item Env:PYTHONPATH -ErrorAction SilentlyContinue
Remove-Variable nativePassword,encodedNative,stationPassword,secret
```

Use the actual native PostgreSQL port/database/user if the deployment differs from the accepted defaults. Environment changes here apply to children of this shell, not the already running native supervisor. Keep secrets out of command arguments and output. The paired runtime deliberately uses normal `verify_activation` through its standard module entry point.

## Provision the Station database once

This first-time step creates only the dedicated synthetic role/database. Its role owns that database and its Station schema, with no superuser/global administration grant. It connects to the normally provisioned local database using the current profile's protected credential. Existing role/database names cause a refusal before creation; do not drop or repurpose them automatically.

```
@'
import os
from pathlib import Path
import psycopg2
from psycopg2 import sql
from sqlalchemy.engine import make_url

url = make_url(os.environ['TOWNLIGHT_STATION_DATABASE_URL'])
profile = Path(os.environ['TOWNLIGHT_PROFILE_ROOT'])
password = (profile/'config/secrets/postgres-
password.txt').read_text().strip()
conn = psycopg2.connect(host='127.0.0.1', port=15432, user='townlight',
                        password=password, dbname='postgres')
conn.autocommit = True
try:
    with conn.cursor() as cur:
        cur.execute('SELECT 1 FROM pg_roles WHERE rolname=%s',
(url.username,))
        role_exists = cur.fetchone() is not None
        cur.execute('SELECT 1 FROM pg_database WHERE datname=%s',
(url.database,))
        if role_exists or cur.fetchone() is not None:
            raise RuntimeError('Dedicated role/database already exists;
inspect before continuing')
        cur.execute(sql.SQL('CREATE ROLE {} LOGIN PASSWORD %s').format(
            sql.Identifier(url.username)), (url.password,))
        cur.execute(sql.SQL('CREATE DATABASE {} OWNER {}').format(
            sql.Identifier(url.database), sql.Identifier(url.username)))
finally:
    conn.close()
'@ | & $control -B -
if ($LASTEXITCODE -ne 0) { throw 'Station database provisioning did not
complete; inspect before retrying.' }
& $control -B -m townlight_station.cli migrate
if ($LASTEXITCODE -ne 0) { throw 'Station migration failed; retain logs and
inspect the dedicated schema.' }
```

If database creation fails after the role was created, the operation is partially complete: inspect that exact owned role/database before resuming. Do not rerun against guessed names or remove unrelated objects. On normal restarts, the database already exists; skip creation and use the existing identity. `migrate` is explicit; API requests do not substitute for schema migration.

## Start and check the paired processes

Use the same configured session. The standard suite entry point verifies the existing native activation. Check readiness of that exact paired origin, not only the desktop's canonical port. Startup below creates hidden children with private log files and records their identities; it does not focus windows or install services.

```
$owned = @()
function Start-StationRole([string]$name, [string[]]$arguments) {
  $p = Start-Process -FilePath $control -ArgumentList $arguments -WindowStyle
Hidden -PassThru `
    -RedirectStandardOutput (Join-Path $stationRoot "logs\$name.out.log") `
    -RedirectStandardError (Join-Path $stationRoot "logs\$name.err.log")
  $script:owned += [pscustomobject]@{ Name=$name; Id=$p.Id;
StartTime=$p.StartTime; Path=$p.Path }
  $script:owned | ConvertTo-Json | Set-Content -LiteralPath (Join-Path
$stationRoot 'owned-processes.json')
  return $p
}
function Wait-StationReady([string]$url, $process) {
  for ($attempt=0; $attempt -lt 40; $attempt++) {
    if ($process.HasExited) { throw "Owned process exited; inspect logs for
$url" }
    try {
      $response = Invoke-WebRequest -Uri $url -UseBasicParsing -TimeoutSec 2
      if ($response.StatusCode -eq 200) { return }
    } catch { }
    Start-Sleep -Milliseconds 250
  }
  throw "Readiness not established: $url; inspect the owned logs"
}
$suite = Start-StationRole 'paired-suite' @('-B','-
m','townlight_runtime.services')
Wait-StationReady 'http://127.0.0.1:15481/ready' $suite
$api = Start-StationRole 'station-api' @('-B','-
m','townlight_station.cli','serve','--port','8756')
Wait-StationReady 'http://127.0.0.1:8756/health' $api
$worker = Start-StationRole 'station-worker' @('-B','-
m','townlight_station.cli','worker')
# Only for the on-air profile:
$air = Start-StationRole 'station-air' @('-B','-
m','townlight_station.cli','air')
$owned | ConvertTo-Json | Set-Content -LiteralPath (Join-Path $stationRoot
'owned-processes.json')
```

Before launching, confirm ports 15481 and 8756 are unused. If readiness fails, inspect the relevant logs and retained profile/schema state; do not start another copy. A port that answers belongs to the intended process only when its ownership is established. A single worker owns a database. `worker --once` performs a bounded pass; it is not the normal continuous worker. The `air` role follows durable on-air state and emits the local program; it does not commission a

physical transmitter or CDN. Its currently configured local output is under Station's `air` directory. Back up the dedicated database and media together; Core's native backup does not cover them.

## Configure a live input and open the operator

Create `live-sources.json` under the Station **data** directory with the following fictional input. An actual sender must supply UDP MPEG-TS to that receiver; this configuration alone does not generate a signal.

```
[{"id":"synthetic-chamber","label":"Synthetic chamber feed","kind":"udp-mpegts","address":"127.0.0.1","port":19000}]
```

At most 32 rows are accepted. IDs are unique 1–64 letters/digits/underscore/hyphen; labels have 1–100 characters; IPv4 and ports 1024–65535 are validated. Each row has exactly the five shown keys. Protect the source network; UDP is not authenticated. Then, under the same signed-in native account:

```
& $control -B -m townlight_station.cli operator --profile-root $profilePath --port 8756
```

This normal operator command opens the operator for a future staff exercise. It reads the native session and exchanges a one-use launch ticket; it does not seed authority or put the native proof in a URL. Ticket expiry is 60 seconds, operator cookie lifetime 900 seconds, with at most 256 outstanding tickets/sessions each. Reopen normally after expiration, API restart or sign-out. Changing database/configuration identities does not repair a staff permission refusal.

## Bounds and shutdown

Default encoding is 1920×1080, 30 fps, 6000 kbps. Optional Station variables permit `MAX_SECONDS` 10–86400 (default 7200), even `WIDTH` 320–3840 and `HEIGHT` 180–2160, `FPS` 1–60 and `BITRATE_KBPS` 128–50000; each is prefixed `TOWNLIGHT_STATION_`. The media source lock and exclusive data identity are verified before use. The dedicated database name must end in `_station`; unowned nonempty storage and escaping/reparse paths are refused. The existing SCM entry point is not a claim of installed service acceptance.

To finish an exercise, stop live capture from the operator and wait for its completed or failed terminal state. Wait for any recording/playout work to finish, cancel unwanted future schedules, sign out and preserve the evidence/logs. The air role starts a separate media child; force-stopping only its parent does not run Python cleanup. Stop only the process trees you started, checking each root's identity before using a saved PID:

```
foreach ($entry in @($owned | Sort-Object StartTime -Descending)) {
  $process = Get-Process -Id $entry.Id -ErrorAction SilentlyContinue
  if ($process -and $process.StartTime -eq $entry.StartTime -and $process.Path -eq $entry.Path) {
    & taskkill.exe /PID $process.Id /T /F | Out-Null
    if ($LASTEXITCODE -ne 0) { throw 'Owned process tree did not stop; inspect before proceeding.' }
    $process.WaitForExit()
```

```
  }
}
```

Retain the ownership file if the shell must close; parse and verify its actual process paths/start times before acting. Do not terminate the native database/supervisor or CivicCast. Interrupting a worker during a job preserves an interrupted attempt, not completed media; recovery is described in the Station operating procedure. Clear secret-bearing environment values when closing the dedicated deployment shell. Do not delete the database/data root to “reset” an uncertain result.

# API and integration examples

## Native command example

The following JavaScript belongs in the existing trusted Townlight frontend/development environment with `invoke` imported from `@tauri-apps/api/core`. It is not a command for an arbitrary web page or an instruction for nontechnical staff. IDs must come from records created in an isolated synthetic exercise.

```
import { invoke } from '@tauri-apps/api/core';
const result = await invoke('meetings_domain_request', {request: {operation:
'get-meeting', id: meetingId}});
console.log(result);
```

## Trusted local HTTP example

An administrator/developer can inspect an already-configured isolated runtime from a trusted local process. This PowerShell example reads the signed-in synthetic profile’s existing proof into memory; it does not create a user, seed a session, reveal the proof in output or bypass a permission check. Do not use real city data in development examples. A read response does not prove approval or publication.

```
Invoke-RestMethod -Uri 'http://127.0.0.1:8756/health'
# Authenticated recording operations additionally require current native
proof.
```

## A synthetic write with subsequent readback

This example demonstrates the existing Meetings HTTP contract in an isolated exercise. It is intentionally fictional. It does not start a service. Its authentication setup is included so it can be run independently under the signed-in synthetic account. If a write times out, inspect current records before repeating it.

```
$profilePath = Join-Path $env:LOCALAPPDATA 'CivicSuite'
$sessionPath = Join-Path $profilePath 'config/local-session.json'
$session = Get-Content -LiteralPath $sessionPath -Raw | ConvertFrom-Json
$headers = @{ 'X-Townlight-Session' = $session.session_hash }
$body = @{
  title = 'Synthetic documentation exercise'
  scheduled_start = '2026-11-05T18:00:00-07:00'
  location = 'Synthetic room'
```

```
  meeting_type = 'regular'
} | ConvertTo-Json
$meeting = Invoke-RestMethod -Method Post `
  -Uri 'http://127.0.0.1:15480/desktop/v1/meetings/meetings' `
  -Headers $headers -ContentType 'application/json' -Body $body
Invoke-RestMethod `
  -Uri ('http://127.0.0.1:15480/desktop/v1/meetings/meetings/' + $meeting.id)
`
  -Headers $headers
Remove-Variable session, headers
```

This write requires Meetings to be enabled and the signed-in staff role to permit it. It is a cross-suite integration example; for this product's request models, use the complete technical reference below.

## Recording and Records handoff example

An authenticated local client sends `POST http://127.0.0.1:8756/v1/recordings` with `X-Townlight-Session` and this JSON, replacing the synthetic UUIDs with saved synthetic meeting/agenda IDs. Configure `synthetic-chamber` first.

```
{
  "meeting_id": "11111111-1111-4111-8111-111111111111",
  "agenda_ids": ["22222222-2222-4222-8222-222222222222"],
  "live_source_id": "synthetic-chamber",
  "idempotency_key": "synthetic-docs-capture-001"
}
```

The initial response is `202` with a queued/running identity rather than completed media. Read `/v1/recordings/{identity}`, explicitly stop live capture through `/v1/recordings/{identity}/stop`, and wait for completed status before reading `/content`. Then the authorized Records client calls `POST /desktop/v1/requests/{request_id}/station-recordings` on the paired Suite runtime with:

```
{
  "recording_id": "33333333-3333-4333-8333-333333333333",
  "expected_sha256":
"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
}
```

The shown hash is deliberately a placeholder. Use the actual completed recording's hash. A mismatched hash must fail. Repeating the identical completed handoff is idempotent; it does not create a release copy or fulfill a request.

## Complete technical contract

Read the Station API reference for every documented operation, request field, constraint, response declaration and source supplement. Download the OpenAPI JSON for tools. For Station this is the suite-side schema only; its separate service contract is documented in the API

supplement. The reference directory includes configuration/source inventories, dependency declarations, native command contracts, MCP signatures and candidate evidence.

## MCP and automation boundary

No direct product MCP server is exposed by the accepted Townlight desktop or Station boundary. Do not configure an invented `townlight mcp` command or assume an AI client has staff permissions. The integration surfaces for this beta are native IPC, the documented HTTP APIs, source bundle imports, exact exports and the existing Station CLI.

The bundled transcription code includes a **TextFlowKit MCP adapter** as a separate source capability. It offers job-based `transcribe_media`, `get_transcript`, `list_jobs`, `get_job_status` and `export_transcript`, and additional tools declared in that adapter. It uses stdio first and optional Streamable HTTP, with the MCP SDK as an optional extra. It is not automatically launched, authenticated as Townlight staff or accepted as a Townlight product MCP deployment. This documentation does not add such a service.

For a transcription client already provisioned separately, an illustrative tool call is:

```
{"name":"get_job_status","arguments":{"job_id":"synthetic-existing-job-id"}}
```

Use a job ID returned by that transcription service. This example does not create a Townlight record or bypass its review/Records handoff. The exact optional adapter tool signatures are supplied in the MCP source reference. Station meeting capture does not become live speech recognition because this optional donor adapter exists.

## Security and permissions

Use your own named account and save before leaving a screen. A saved draft is durable; unsaved form text may not be. If a write has an unknown result, reopen its record before repeating it.

Read source jurisdiction, dates, citation, version and visibility. A copied source hash identifies exact bytes, not their legal applicability. If a source changes or permission is removed, resolve the source problem rather than substituting stale cached content.

Keep originals and derivatives separate. Review the intended audience before export. A public summary excludes private fields only through the applicable copy contract and your review. A renamed private packet is still private content.

Drafting, review, approval, attachment, export, fulfillment and publication are separate operations. A Townlight workpaper does not itself adopt law, issue a permit, impose a fine or release a records response. The responsible staff member records the applicable decision.

The native session is validated and mapped to a persisted SQL identity. An editable actor/reviewer label is not authentication. Product/module permissions and source visibility apply at the server boundary. Revoked/unavailable authority fails closed for new Station operations; a previous permission result is not a perpetual authorization.

Use protected local folders, account permissions and credential storage. Never include the native session file, database passwords, provider keys or unredacted support bundles in a public issue or

generated public documentation. A local public view is not automatically a secured internet deployment. Keep network exposure and real equipment acceptance specific to the chosen beta site.

## Backup recovery and maintenance

Save work and use Core's canonical **Backup Now** before maintenance. That operation stops writers. Preserve the coherent profile/database/document backup and its receipt, then use **Start** to resume. A source export is not a full backup.

Use **Restore Latest Backup** through the supported controls. If the result says **Restore needs service start**, use **Start**, then **Check**, and reopen a known record. Do not repeat restore merely because services need to start. The prior accepted profile exercise verified request/marker recovery for its identified artifact. Repeat the required recovery proof for this changed signed candidate before calling it accepted.

Keep Station's dedicated database and media evidence together in its own backup. Core's native-profile backup must not be assumed to include an external Station data directory. Stop writers during a coordinated backup. Preserve failed media attempts and logs.

On failure, record the time, selected composition, component/installer versions and exact message. Check System Health for stopped, starting, ready or failed dependencies. Use the existing Retry control after readiness; do not create duplicate business records to refresh a stale view. Share only a redacted diagnostic bundle. Preserve secrets, original evidence, recovery material and unrelated installations.

## Troubleshooting

| Symptom | Likely meaning | Next action |
|---|---|---|
| Workspace absent | Wrong product composition or unavailable canonical runtime. | Check selected composition; activate through Finish Records Migration; wait for readiness. |
| Cannot sign in / 401 | Missing or expired named account session. | Sign in normally; reopen a Station launch ticket if applicable. |
| Permission denied / 403 | Account, module, source or browser-origin permission. | Ask the authorized administrator to inspect the intended permission; do not invent role headers. |
| Save conflict / 409 | The saved revision, expected source hash or lifecycle changed. | Keep edits, reopen current state and reconcile deliberately. |
| Invalid input / 422 | Required field, format, geometry, source envelope or state is invalid. | Read the field detail and correct the input/source; preserve originals. |
| Service unavailable / 503 | Database, schema, connected | Inspect System Health and |

| Symptom | Likely meaning | Next action |
|---|---|---|
| | source or configuration unavailable. | the named dependency. Preserve work; retry only after readiness. |
| Export refused | Missing exact review, approval, audience or source authorization. | Read the refusal and complete the owning product's explicit decision. |
| Restore needs service start | The snapshot was restored but services have not resumed. | Use Start, then Check and verify a saved record; do not blindly repeat restore. |
| Ticket expired or used | A launch is one-use and expires after 60 seconds. | Reopen from the signed-in native session; do not replay the old URL. |
| Recording interrupted | Input/worker failed or a live signal was lost. | Retain attempt/logs. Retry eligible file work explicitly; start a new live capture for a live gap. |
| Recording not downloadable | It is not complete or its retained bytes failed validation. | Inspect state and evidence; do not serve a partial attempt as completed media. |

## Verified workflows and limits

The original accepted Station manual and its workflow claims are preserved byte for byte. Those results identify earlier artifacts; they are not installed signoff for this naming candidate. The current package retains the same Beta Python service and pinned CivicCast donor. Source presence and a successful wheel import do not establish the required Windows Meetings → Station → Records handoff.

The new candidate must verify actual saved meeting/agenda context, authorized Station capture and stop, completed media/hash readback, private Records handoff, idempotency, authorization/logout, persistence and the agreed recovery path. Its ACCEPTANCE.md must distinguish new proof from justified reused evidence. This source documentation does not certify a signed MSI, live YouTube upload, physical PEG broadcast, city deployment or long-duration commissioning.

The separate provisioning/service boundary and existing external-service limits remain as described below. Staff remain responsible for official release and publication decisions. No additional provider or selected-model change is made.

## Terms of service and license

### Scope and status

These terms describe the Townlight Beta 1 software and this documentation package. This public Windows beta is for evaluation. No uptime commitment, support response time, indemnity, data-

processing agreement or paid service is created by these documents. A separate signed agreement, if any, governs services it actually provides.

## Software license and rights

Current Townlight-owned distributions identified in this candidate use **AGPL-3.0-only**. The full license is supplied with this documentation and with the source companion. The license governs rights to run, copy, modify and convey the covered software; these operational terms do not add restrictions to those rights. Historical grants and dependency licenses retain their own terms. CivicCast remains a separate product and is not relicensed by this documentation work.

When conveying covered software, retain applicable notices and provide the corresponding source and installation/build information required by the license. For a modified version supporting remote network interaction, follow section 13's source-offer requirement. The companion archive identifies the source for this candidate; a link to a different public branch is not a substitute for the matching source. Read the supplied license for the complete conditions. See the GNU license reference and GNU licensing FAQ.

## Data ownership and access

Importing a document does not transfer its ownership or grant permission to publish it. The operating organization controls its accounts, access decisions, lawful source use, retention and review/release policy. Keep resident information, credentials, private notes and restricted evidence in their authorized locations. Documentation examples contain fictional data. No real city account, credential or record is required to exercise these examples in an isolated test environment.

## Local operation and external processors

The tested Windows beta workflow uses named staff accounts. Station has separate storage and credentials. Configure backup destinations and optional model/network providers deliberately. A configured external provider may receive submitted prompts, images or audio according to that configuration and provider's terms; this documentation does not promise that every optional configuration is offline. An email outbox or local public view does not establish external delivery or an internet deployment. This static documentation site has no analytics, account collection or remote font dependency.

## Staff decisions and source applicability

Staff verify the jurisdiction, source currency, permissions, citations and proposed audience. Model output and generated guidance require review. No draft is automatically an official legal interpretation, zoning determination, permit approval, fee assessment, enforcement notice or records-release decision. Official permit fee authority remains with the adopted source and responsible organization; imported amounts do not become a new authoritative calculation.

## Beta operation and recovery

Use the identified candidate in its supported fresh-install scope. Preserve backups and original evidence. Test physical equipment and chosen network/service configuration before relying on

them. The supplied acceptance describes selected synthetic workflows, not every screen, vendor, field condition or upgrade. Operators decide whether a particular deployment meets their own operational obligations.

### Warranty and liability

The covered software's warranty and liability conditions are those in sections 15–17 of the supplied AGPL and any applicable separate written undertaking. This documentation creates no additional warranty or limitation beyond those terms. Third-party software is subject to its retained license notices. Nothing here removes rights that applicable law makes nonwaivable.

### Support and incident reporting

Use the project's existing SUPPORT and SECURITY guidance for its matching repository. Report reproducible behavior with the version, operation and redacted logs. Do not put credentials, resident records or unredacted evidence in a public issue. Coordinate sensitive reports through the channel designated in SECURITY. No new support address or service commitment is invented here.

### Ending a beta exercise and document changes

When an organization ends testing, it controls retention/export and removes access according to its policy. Uninstall may retain data; back up and inspect the selected removal options. Ending an exercise does not revoke rights already granted by AGPL. Later software/documentation versions must identify their changed candidate and scope; historical receipts retain their original identities. These terms do not claim that a city has accepted a contract merely by reading a manual.

## Glossary

| Term | Meaning |
|---|---|
| Approval | An explicit authorized decision on the saved text/artifact. It is separate from saving or attachment. |
| Canonical record | The owning product's retained identity and saved version, rather than an editable copied label. |
| Composition | The selected set of desktop modules; inspections-preview exposes the eight staff products. |
| Corresponding source | The source/build material matching the delivered covered program, subject to its license. |
| Derivative / Records copy | A separately prepared document with its own intended audience and hash. |
| Hash / SHA256 | A fingerprint of exact bytes. It detects change; it does not establish legal authority. |

| Term | Meaning |
|---|---|
| Idempotency | Repeating the same identified request has the same intended effect; changing its payload can be refused. |
| IPC | The native desktop command boundary between its browser interface and Rust host. |
| MCP | Model Context Protocol; no direct Townlight product server is enabled by this beta. |
| Original | Retained source bytes. A reviewed derivative does not replace them. |
| Provenance | Source, version, date, actor and relationship information retained with evidence. |
| Revision | An immutable saved version. Later edits do not inherit an earlier review automatically. |
| Snapshot | A fixed audience-specific copy tied to a source/reviewed revision. |
| Station | Townlight's separate media boundary using a pinned CivicCast-derived engine; standalone CivicCast remains separate. |

## Alphabetical subject index


- **Accounts and authentication** — Security and permissions.
- **AGPL and source** — Terms of service and license.
- **API contracts** — API and integration examples.
- **Architecture** — Architecture.
- **Backup** — Backup recovery and maintenance.
- **Beta version** — Beta identity.
- **Configuration** — Configuration and administration.
- **Evidence and acceptance** — Verified workflows and limits.
- **Fees and official decisions** — Terms of service and license.
- **First run** — First run.
- **Handoffs and audience** — Connected workflows.
- **MCP** — MCP and automation boundary.
- **Originals and provenance** — Security and permissions.
- **Recovery** — Backup recovery and maintenance.
- **Staff procedures** — Daily workflow.
- **Station configuration** — Configuration and administration.
- **Technical stack** — Technical specifications.
- **Troubleshooting** — Troubleshooting.

### Operating subject links

- **Apply the caption profile** — Apply the caption profile.
- **Approve and publish local playback** — Approve and publish local playback.
- **Bounds and shutdown** — Bounds and shutdown.
- **Check playout and commit deliberately** — Check playout and commit deliberately.
- **Configure a live input and open the operator** — Configure a live input and open the operator.
- **Prepare the normally activated synthetic profile** — Prepare the normally activated synthetic profile.
- **Provision the Station database once** — Provision the Station database once.
- **Recover without replaying unknown work** — Recover without replaying unknown work.
- **Schedule a recording and interpret its state** — Schedule a recording and interpret its state.
- **Scheduling playout and recovery** — Scheduling playout and recovery.
- **Start and check the paired processes** — Start and check the paired processes.
- **Store the independent test configuration** — Store the independent test configuration.
- **Take a live override and return** — Take a live override and return.

## Source and document control

This manual covers the Station service paired with Windows MSI 1.7.0.1. The Windows acceptance summary records the tested Meetings, Station and Records handoff.

### Workflow subject index

- **Capture and stop** — Daily workflow, step 4.
- **Choose the input** — Daily workflow, step 3.
- **Close and recover** — Daily workflow, step 8.
- **Hand off to Records** — Daily workflow, step 6.
- **Open the operator** — Daily workflow, step 2.
- **Prepare meeting context** — Daily workflow, step 1.
- **Review recording evidence** — Daily workflow, step 5.
- **Use playout deliberately** — Daily workflow, step 7.
- Source inventory and file hashes
- All component metadata
- Installed domain OpenAPI

- Workflow matrix
- Full AGPL text

## Completed recordings on YouTube and live CivicCast coverage

This source extension adds only completed-recording YouTube uploads and a live Station-to-CivicCast feed. It is **not in the published Beta 1 installer**. Existing component versions are retained. New Station code is AGPL-3.0-only. The pinned CivicCast source and third-party licenses remain unchanged. Standalone CivicCast is a separate product; no installer, account, database, service or default in it is changed by Station.

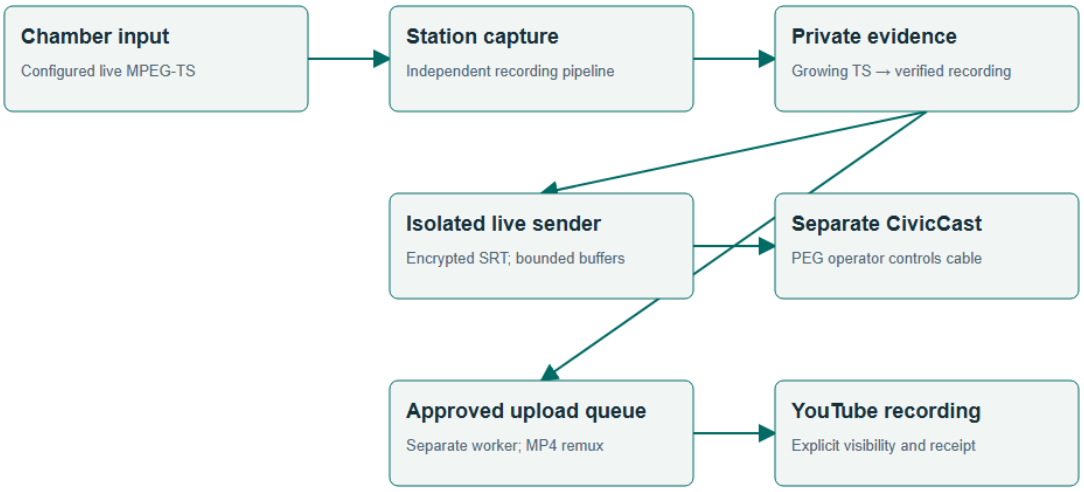
### Staff workflow

For live coverage, choose a meeting, its chamber input and the administrator's **CivicCast live feed** option, then start recording. Choose **Recording only** to capture privately without a network feed. A disconnected receiver does not stop the recording. Station sends the live MPEG-TS bytes as the recording grows; it does not wait for the completed file. The feed is available only during that capture. The CivicCast/PEG operator selects that source and controls when it is broadcast on local cable. Station does not automatically put the PEG channel on air. `sending` means transport bytes were sent, not proof of cable carriage.

After Station shows **complete**, use **Upload completed recording to YouTube**. Review the title, select private/unlisted/public visibility (default **private**), and enter the approval note. Click **Upload to YouTube**. This is an explicit external upload to the Station-configured OAuth account. No upload occurs just because recording stopped, a playout was committed or a Records attachment was made. Station does not create a YouTube livestream.

The upload receipt shows queued/running/uploaded/failed/needs-reconciliation, the chosen visibility and returned video link. `uploaded` is the YouTube upload acknowledgement; YouTube may still be processing the video. The canonical recording remains private and byte-identical for the existing Records handoff. The YouTube visibility is separate from local playback/disclosure approval.

## Architecture




The sender has bounded, leaky buffers and follows the live edge after congestion. Its network sink shares no pipeline with the recorder. A late join or connection loss can leave a live coverage gap, while the private recording continues. H.264/AAC and existing MPEG-TS captions are forwarded without another encode. The MP4 upload remux reuses `vod_job.py`; it does not generate transcripts or captions, and does not upload a separate caption track. Upload queue work runs in a separate process so a long upload never monopolizes the capture worker.

## Administrator setup

Use Station's existing dedicated data root and PostgreSQL database. Stop Station writers for a coordinated backup, retain that backup, and run the explicit `townlight-station migrate` before starting this source version. Schema **4** adds `station.youtube_uploads`, keeping existing recording/schedule/playout evidence. The data identity marker and component versions are unchanged. Old services expect schema 3; use a matched source/runtime and preserve the backup for rollback.

Create `civiccast-feeds.json` in the private Station data directory:

```
[
  {
    "id": "peg",
    "label": "CivicCast council feed",
    "address": "192.0.2.10",
    "port": 10080,
    "secret_env": "TOWNLIGHT_STATION_SRT_PEG"
  }
]
```

The address is a numeric IPv4 **local interface bind address on Station**, not the CivicCast computer. `192.0.2.10` is a documentation placeholder. Use the actual Station interface, an unused port (1024–65535), and appropriate network access between those two systems. Clients

select an administrator-defined ID, never an arbitrary destination or media pipeline. Keep configuration files private.

Set `TOWNLIGHT_STATION_SRT_PEG` in the capture worker's process environment to a 10–79 character SRT passphrase. Configure a separate CivicCast SRT live source with the caller URI `srt://<Station-address>:10080?mode=caller&latency=200` and the same passphrase through CivicCast's existing credential control. Do **not** put the passphrase in the URI. Station's listener uses 128-bit SRT encryption. Keep the two accounts and configuration stores separate; matching this specific transport credential is the deliberate pairing. Site-specific camera, headend and firewall commissioning remain deployment work.

For completed uploads, configure these **Station-only** environment variables in the upload worker process. Obtain OAuth authorization for the intended Station YouTube channel/account through the existing deployment's Google configuration; no OAuth onboarding framework is added here. Do not use CivicCast's credentials implicitly or copy its profile.

- `TOWNLIGHT_STATION_YOUTUBE_CLIENT_ID`
- `TOWNLIGHT_STATION_YOUTUBE_CLIENT_SECRET`
- `TOWNLIGHT_STATION_YOUTUBE_REFRESH_TOKEN` (authorized for YouTube video uploads)

Run the existing `serve` and capture `worker`, plus the separate upload consumer with the same Station settings and dedicated database:

```
townlight-station migrate
townlight-station serve
# In separate managed processes with the same explicit Station configuration:
townlight-station worker
townlight-station uploads
```

These are deployment commands, **not commands executed against installed apps by this development task**. Run only one capture worker and one upload worker per database. Each holds a distinct advisory lock; media children require the existing Windows job containment. The recorder child gets no provider credentials; the SRT child receives only its explicit transport passphrase plus the existing runtime allowlist. The upload worker reads only the Station OAuth variables. OAuth/upload HTTP is restricted to Google HTTPS hosts; redirects are not followed. The pinned donor import is checked against Station's declared media source.

## HTTP API and examples

All `/v1` operations below require the existing current Townlight staff session header (`X-Townlight-Session`) or the existing same-origin operator session. Read roles can inspect receipts. Starting live feed/upload/retry requires the existing Station `capture` capability. No new login, role system or MCP server is added.

| Method and route | Purpose |
|---|---|
| GET `/v1/civiccast-feeds` | Capture-authorized list of configured IDs and labels; no transport credentials |

| Method and route | Purpose |
|---|---|
| POST `/v1/recordings` | Existing capture contract plus optional `civiccast_feed_id`, allowed only with `live_source_id` |
| POST `/v1/recordings/{recording_id}/youtube` | Queue one explicit completed-recording upload |
| GET `/v1/youtube-uploads` | Read-authorized latest 100 durable upload receipts |
| POST `/v1/youtube-uploads/{upload_id}/retry` | Capture-authorized retry of a **preparation** failure only |

Station's generated `/openapi.json` includes the new routes and request schemas. Example bodies (use real canonical IDs from the selected deployment):

```
{
  "meeting_id": "11111111-1111-4111-8111-111111111111",
  "agenda_ids": [],
  "live_source_id": "chamber",
  "civiccast_feed_id": "peg",
  "idempotency_key": "council-live-2026-10-30"
}
```

```
{
  "title": "City council meeting — 30 October 2026",
  "privacy": "private",
  "approval_note": "Reviewed completed recording for upload",
  "idempotency_key": "council-vod-2026-10-30"
}
```

Upload titles are 1–100 characters; control characters and angle brackets are rejected. Visibility is private/unlisted/public; approval note 3–500 characters; key 1–100 characters. Repeating the same key returns the same intent, a changed request conflicts, and another key cannot duplicate an existing recording upload. A missing/uncompleted/tampered recording is rejected before upload admission; the worker independently verifies the recording hash before remuxing/sending. Retained receipts include approving actor, note/time, recording hash, MP4 path/ hash/size, chosen title/privacy and provider video ID/URL after acknowledgement. No OAuth token, passphrase or donor credential pointer is stored in receipts.

## Recovery and limits

A failed preparation (for example missing Station OAuth configuration or failed remux) is `failed`; correct the cause and explicitly call its retry route. The prior attempt files remain retained. Once any external request begins, an error is conservatively `needs-reconciliation`. A worker restart also moves interrupted `running` uploads to that state. There is no automatic retry that could create a duplicate YouTube video. Check YouTube Studio and the retained job before a human chooses another external action. This increment provides no automatic remote

reconciliation/delete/reupload control. A valid uploaded link is retained even if the capture worker is restarted. The existing recording/Records path does not depend on the external upload succeeding.

Live sender state is in the recording's `details.civiccast_delivery`; its final byte count and per-attempt status/log files stay with the private recording. On capture-worker restart, an interrupted feed is marked interrupted rather than left appearing to send. Stop/reap is bounded even if the network child hangs. No feed is automatically re-aired after recovery.

### Verification and release status

`tests/accept_deliveries.py` exercises the real Station API/store/media/remux, CivicCast's pinned SRT source adapter and real YouTube file-upload client using a disposable PostgreSQL cluster, generated audio/video, synthetic staff authority and `httpx.MockTransport` for Google. No city systems or external publication are involved. `tests/accept_delivery_ui.cjs` drives the actual compiled operator controls in headless Chromium with synthetic API responses. Existing boundary/ operations checks and TypeScript compilation also apply.

Example invocation from a development environment with existing dependencies:

```
$env:PYTHONPATH = (Resolve-Path ./beta-1).Path
$env:STATION_TEST_PG_BIN = 'C:/test-tools/postgres/bin'
$env:STATION_TEST_MEDIA_RUNTIME = 'C:/test-tools/station-media-runtime'
python -B ./beta-1/tests/accept_deliveries.py C:/test-output/station-deliveries-new
```

Use a **new** output directory; the script starts/stops only its owned disposable cluster and children. See the task's retained acceptance receipts for actual results. This evidence establishes the new source wiring synthetically. It does not replace CivicCast's already established real YouTube delivery evidence, nor claim a new real YouTube upload, physical cable broadcast, signed installer or Sandbox acceptance of this extension. Published Beta 1 artifacts stay unchanged.

## Delivery subject index

- CivicCast live feed
- YouTube: explicit completed recording upload; visibility; receipt and reconciliation.
- Schema4: durable upload intent; existing evidence retained.

The resident portal is deployed separately. See the Townlight Portal Guide for its workflow and acceptance limits.