Manage Twenty views, layouts, and webhook metadata

Use this guide to change one view, page-layout hierarchy, or provider webhook registration. Begin with the current workspace metadata schema and dependency graph, submit one critical mutation, then verify the provider result through a separ

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/twenty/guides/manage-views-layouts-and-webhook-metadata.md

The connector validates the selected resource, outer input, parent query, and webhook-secret boundary. Twenty owns resource-specific body fields, relationships, cascades, and the resulting user experience.

Prerequisites

Require:

- an admitted Twenty host bound to the intended tenant and workspace;
- authenticated access to metadata OpenAPI and the needed metadata operations;
- an integrator who owns the affected object, view, layout, or webhook;
- a trusted tenant approver for critical metadata changes;
- a current provider backup or recovery plan for the affected metadata;
- a durable place for schema, dependency, Action, approval, and result evidence;
- coordinated host and provider ownership for webhook registration or rotation.

Set only AIP client coordinates:

```
SH
export AIP_URL='https://aip.example.com'
export AIP_TOKEN_FILE='/run/secrets/aip-native-token'
```

Provider credentials and webhook secret bytes remain host-owned.

1. Confirm the admitted operations

Discover the metadata Resources and Tools used by the intended change:

```
SH
aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  capability list "$AIP_URL" \
  --capability-id cap:twenty:openapi.metadata \
  --limit 1

aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  capability list "$AIP_URL" \
  --capability-id cap:twenty:metadata.list \
  --limit 1
```

```
aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  capability list "$AIP_URL" \
  --capability-id cap:twenty:metadata.update \
  --limit 1
```

Use `metadata.create` or `metadata.delete` instead of update when that is the actual intent. Record catalog revision and every selected contract and schema digest.

Stop when an operation is absent. Do not widen the host allowlist during an active change merely to satisfy the procedure.

2. Capture the metadata schema

Read `cap:twenty:openapi.metadata` with `{}`. Retain the body under tenant-scoped controls and calculate its digest outside the connector.

Record connector artifact, pinned provider mapping, instance, tenant, workspace, credential revision, Action ID, observation time, HTTP status, and provider request ID.

Extract only the provider fields needed for the chosen resource. Freeze their required status, types, enums, identifiers, parent references, and delete behavior as the change's body contract.

The connector accepts an arbitrary object body with at most 2,048 top-level properties. It does not validate that body against the observed provider document.

3. Snapshot the current dependency graph

Use `metadata.list` and `metadata.get` before any mutation. The exact reads depend on the resource family.

Snapshot a view

Start with the owning object and view:

```
JSON
{
  "resource": "views",
  "query": {
    "objectMetadataId": "0190c42f-2d5a-7000-8000-000000000070"
  }
}
```

For each selected view, list these children using its UUID as `viewId`:

- `viewFields`;
- `viewFilters`;
- `viewGroups`;
- `viewSorts`;
- `viewFilterGroups`.

The connector permits an omitted `viewId`, but a change review should retain a bounded parent-specific snapshot.

Snapshot a page-layout hierarchy

List layouts by object and optional type:

```
JSON
{
  "resource": "pageLayouts",
  "query": {
    "objectMetadataId": "0190c42f-2d5a-7000-8000-000000000070",
```

```
    "pageLayoutType": "RECORD_PAGE"
  }
}
```

Allowed types are `RECORD_INDEX`, `RECORD_PAGE`, `DASHBOARD`, and `STANDALONE_PAGE`. The connector rejects a type without `objectMetadataId`.

List tabs under one layout:

```
JSON
{
  "resource": "pageLayoutTabs",
  "query": {
    "pageLayoutId": "0190c42f-2d5a-7000-8000-000000000071"
  }
}
```

Then list widgets under each tab:

```
JSON
{
  "resource": "pageLayoutWidgets",
  "query": {
    "pageLayoutTabId": "0190c42f-2d5a-7000-8000-000000000072"
  }
}
```

Both parent queries are mandatory and UUID-validated. Retain ordered child IDs and the fields that the planned body can change.

Snapshot webhook registrations

List `webhooks` with no query object. The connector rejects any query key for this resource and recursively removes every result field named `secret`.

Record registration IDs, target ownership, event selection, enabled state, and provider fields exposed by the current metadata document. Do not infer the configured secret from the redacted result.

4. Define one metadata intent

Choose exactly one operation and resource:

Intent	Capability	Required connector input
Add one resource	`metadata.create`	`resource`, provider-validated object `body`
Change one resource	`metadata.update`	`resource`, resource UUID, provider-validated object `body`
Remove one resource	`metadata.delete`	`resource`, resource UUID

Write the complete Action input from the frozen provider body contract. Retain its canonical hash and the expected before and after values.

Approval evidence should include:

- schema digest and resource body contract;
- parent and child IDs;
- current dependency snapshot;
- expected UI, automation, routing, and event effects;
- rollback or restoration procedure outside the connector;
- target tenant, workspace, credential, and configuration revisions.

The connector provides no dependency analysis or metadata transaction. Split multiple resources into ordered, independently verifiable intents.

5. Submit one governed change

Reserve a stable Action ID and external-account-scoped key:

```
SH
export CAPABILITY='cap:twenty:metadata.update'
export INPUT_FILE='metadata-update.json'
export ACTION_ID='act_twenty_metadata_update_001'
export IDEMPOTENCY_KEY='twenty-metadata-update-v1'
```

Submit once:

```
SH
aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  "$CAPABILITY" \
  --action-id "$ACTION_ID" \
  --idempotency-key "$IDEMPOTENCY_KEY" \
  --mode sync \
  --input @"$INPUT_FILE"
```

Without admitted authorization, the Action enters `requires_human`. Approval must bind the exact resource, ID, body, schema and dependency evidence, policy hash, workspace, and intended effect.

Approval resumes the same queued Action. Do not replace the Action or key while the original remains pending.

All metadata mutations advertise unsafe retry. A key is required for ownership and collision control, not automatic replay.

6. Apply the webhook-secret rules

Skip this section for non-webhook resources.

Create a provider registration

The create input must use `resource: webhooks`. Its body must omit both `secret` and `rotateConfiguredSecret`.

The connector injects the secret configured by `AIP_TWENTY_WEBHOOK_SECRET_FILE`. Create fails when host webhook security is not configured.

The host exposes `/webhooks/twenty` only when the webhook secret is present. It also requires an authorized `AIP_CONNECTOR_HOST_EVENT_ENDPOINT` for central event delivery.

The provider target URL, public ingress, TLS, routing, and secret file must describe the same controlled deployment. Registration creation alone does not prove that ingress or publication works.

Rotate a provider registration

Use `metadata.update` for the exact webhook UUID. The body may include only this connector control when requesting host-secret injection:

```
JSON
{
  "resource": "webhooks",
  "id": "0190c42f-2d5a-7000-8000-000000000073",
  "body": {
    "rotateConfiguredSecret": true
  }
}
```

The connector removes `rotateConfiguredSecret` and injects the current host-owned secret. A caller-supplied `secret`, Boolean `false`, or another value is invalid.

Provider update and host secret rollout are not atomic. Use a deployment-owned cutover plan that defines old and new ingress ownership, validation, rollback, and the point at which the old secret is revoked.

Never place either secret value in Action input, approval evidence, logs, or a read-back comparison.

7. Read the durable mutation result

Read the original Action:

```
SH
aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action status "$AIP_URL" \
  "$ACTION_ID" \
  --include-result \
  --include-receipts \
  --wait-ms 30000
```

Proceed only after terminal `completed`. Retain provider HTTP status, request ID, decoded redacted body, approval, and receipts.

For webhook metadata, every response field named `secret` is recursively removed. Absence of a secret in the result is expected and is not proof of a provider mismatch.

8. Verify the provider state

Use `metadata.get` on the validated result ID or the original update target. For a create whose result lacks a usable ID, perform a bounded parent-specific list and stop if the created resource cannot be uniquely identified.

Compare only reviewed fields from the frozen body contract. Re-list the affected parent children and confirm that unrelated IDs and ordering remain as expected.

For a delete, verify absence and re-read each known parent or dependent set. The connector does not state whether the provider cascades related resources.

For a webhook registration, also run a controlled signed delivery through the deployment ingress and verify durable AIP event publication. Metadata read-back alone does not exercise signature, replay, storage, or central delivery.

Recover without replaying a critical mutation

Observation	Response
Action is `requires_human`	Complete the original approval workflow
Action remains pending	Read the same Action and receipts
Definitive provider rejection	Refresh metadata schema and dependencies before a new intent
Temporary or transport failure	Preserve Action and key; inspect provider metadata before another write
Invalid or oversized provider response	Treat mutation outcome as uncertain and verify current state
Resource cannot be identified	Stop; do not guess an ID or repeat create
Read-back differs	Classify provider validation, automation, concurrency, or schema drift
Webhook metadata matches but delivery fails	Preserve registration and inspect ingress, signature, replay, storage, and publication separately

The connector exposes no rollback or reconciliation operation. A recovery change is a new critical intent with current schema, dependency, approval, and idempotency evidence.

Retain the change evidence

Keep:

- connector, artifact, provider mapping, tenant, workspace, and credential identities;
- catalog revision and selected capability digests;
- metadata OpenAPI digest and frozen body contract;
- bounded before-state and dependency snapshot;
- Action input hash, Action ID, key, approval, and policy hash;
- terminal result, receipts, and provider request ID;
- read-back and dependent-set comparison;
- for webhooks, redacted registration and separate signed-delivery evidence;
- recovery or rollback actions as independent chains.

This evidence supports one controlled metadata change. It does not establish general provider compatibility, atomic metadata migration, or qualification of another artifact or workspace.

Related documentation

- Twenty workspace metadata
- Twenty OpenAPI documents
- Manage standard and custom records
- Twenty connector configuration
- Approvals and policy