Manage standard and custom Twenty records

Use this guide to manage one standard or custom object through a current schema snapshot, bounded record discovery, duplicate review, one governed mutation, and a provider read that verifies the intended result.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/twenty/guides/manage-standard-and-custom-records.md

The connector applies the same outer object grammar to standard and custom objects. Twenty owns which objects and fields exist in the configured workspace and which fields are writable at the observed provider revision.

Prerequisites

Require:

- an admitted and healthy Twenty host for the intended tenant and workspace;
- a caller authorized to discover and invoke the selected capabilities;
- a trusted tenant approver for every mutation;
- an owner for the application's object and field allowlist;
- a durable store for schema, Action, approval, key, and provider evidence;
- a data-handling policy for confidential and personal record content.

Set only AIP client coordinates:

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

The operator owns the provider origin, workspace UUID, and API token. Do not add them to record input.

1. Confirm the admitted surface

Discover the capabilities needed for the intended path. A create path normally uses these operations:

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

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

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

Record the catalog revision and each contract and schema digest. Stop when an operation is absent; a syntactically valid ID cannot bypass admission.

For updates or lifecycle changes, discover that exact capability before constructing the Action.

2. Capture the current workspace schema

Read the core OpenAPI document with an empty input:

```
SH
export SCHEMA_ACTION_ID='act_twenty_records_schema_001'

aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  cap:twenty:openapi.core \
  --action-id "$SCHEMA_ACTION_ID" \
  --mode sync \
  --input '{}'
```

Retain the body under tenant-scoped access and calculate its digest outside the connector. Record the workspace, credential revision, timestamp, provider request ID, and connector artifact identity without retaining secret bytes.

Confirm that `OBJECT` exists and record:

- provider object name and label;
- required and writable fields;
- field types and relationship targets;
- unique and duplicate-sensitive fields;
- fields that trigger automation or notification;
- read and write permissions of the fixed provider credential;
- fields excluded from application input, output, logs, and evidence.

The document is an observation, not a compatibility guarantee. Stop when the observed object contract differs from the application's reviewed contract.

3. Freeze an application object contract

Create a versioned application record that binds:

```
JSON
{
  "workspace_identity": "deployment-owned",
  "object": "companies",
  "openapi_digest": "sha256-of-observed-document",
  "allowed_read_fields": ["id", "name"],
  "allowed_write_fields": ["name"],
  "duplicate_fields": ["name"],
  "maximum_bulk_records": 1
}
```

This is application policy, not connector input. Keep its revision with every governed mutation.

For a custom object, derive every listed field from the observed workspace schema. Do not copy a standard-object example into a custom object without reviewing types, required values, and relationships.

4. Read a bounded existing set

Write `record-list.json` using the exact object:

```
JSON
{
  "object": "companies",
  "query": {
    "limit": 25,
    "filter": "name[like]:AIP%",
    "depth": 0
  }
}
```

Submit the read:

```
SH
export LIST_ACTION_ID='act_twenty_records_list_001'

aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  cap:twenty:record.list \
  --action-id "$LIST_ACTION_ID" \
  --mode sync \
  --input @record-list.json
```

Validate that returned fields, object identity, pagination, and result size match the frozen application contract. Apply output minimization before storing or displaying provider records.

5. Review duplicate candidates

Before create, write one candidate body under `data`:

```
JSON
{
  "object": "companies",
  "body": {
    "data": [
      {
        "name": "AIP Managed Company"
      }
    ]
  },
  "query": {
    "depth": 0
  }
}
```

Invoke `cap:twenty:record.find_duplicates` with a new read Action. Review the provider candidates against the application's duplicate fields and current business ownership.

Duplicate detection provides candidates. It does not select a survivor, authorize merge, or prove that no duplicate exists.

Choose one path:

Observation	Next intent
No acceptable match	Create one deterministic record
One record is the intended entity	Update that exact ID when a change is needed
Several records need consolidation	Stop and prepare a separately approved merge
Schema or ownership is unclear	Stop and refresh the object contract

6. Freeze one mutation

For create, write the provider body once:

```
JSON
{
  "object": "companies",
  "body": {
    "name": "AIP Managed Company"
  },
  "query": {
    "depth": 0
  }
}
```

For update, use the validated provider ID and only approved writable fields:

```
JSON
{
  "object": "companies",
  "id": "0190c42f-2d5a-7000-8000-000000000060",
  "body": {
    "name": "AIP Managed Company Updated"
  },
  "query": {
    "depth": 0
  }
}
```

Freeze capability, input bytes, schema digest, object-policy revision, target ID, expected prior values, and expected result. A semantic change requires a new intent.

7. Submit one governed mutation

Reserve stable identities for the selected path:

```
SH
export MUTATION_CAPABILITY='cap:twenty:record.create'
export MUTATION_INPUT='record-create.json'
export MUTATION_ACTION_ID='act_twenty_records_create_001'
export MUTATION_KEY='twenty-records-create-v1'
```

Submit once:

```
SH
aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  "$MUTATION_CAPABILITY" \
  --action-id "$MUTATION_ACTION_ID" \
  --idempotency-key "$MUTATION_KEY" \
  --mode sync \
  --input @"$MUTATION_INPUT"
```

Without admitted authorization, the Action enters `requires_human`. The trusted approver reviews the frozen input, schema and policy identities, duplicate evidence, target, consequences, and required reason.

Approval resumes the same queued Action. Do not submit a replacement Action or change the key while the original remains pending.

Create forces provider upsert and derives a missing record ID from workspace, key, and index zero. Update preserves the supplied target ID. Both advertise retry safety only with the original key and unchanged canonical input.

8. Read the durable mutation result

Read the original Action:

```
SH
aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action status "$AIP_URL" \
  "$MUTATION_ACTION_ID" \
  --include-result \
```

```
  --include-receipts \
  --wait-ms 30000
```

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

A completed response does not prove that provider automation, webhooks, notifications, indexes, or integrations have completed.

9. Verify the provider record

Extract a validated provider record ID from the result. Never derive it from an Action ID or assume the application name is unique.

Write `record-get.json`:

```
JSON
{
  "object": "companies",
  "id": "0190c42f-2d5a-7000-8000-000000000060",
  "query": {
    "depth": 0
  }
}
```

Invoke `cap:twenty:record.get` under a new read Action. Compare only fields owned by the frozen application contract and retain the verification timestamp and provider request identity.

If the result differs, do not overwrite it automatically. Classify schema drift, provider validation, automation, concurrency, wrong target, or uncertain prior execution before preparing another mutation.

10. Choose a separate lifecycle intent

Use a new Action, key, approval, and current preflight for every lifecycle change.

Intent	Capability	Boundary
Hide one recoverable record	`record.soft_delete`	Restore depends on provider-retained state
Restore one soft-deleted record	`record.restore`	Uses a filtered collection route at this revision
Permanently remove one record	`record.destroy`	Critical; no connector rollback
Consolidate duplicates	`record.merge`	Two through nine unique IDs; provider chooses merge semantics
Change a bounded set	A `*_many` operation	Explicit non-empty filter and target evidence required

Do not treat soft delete and restore as one transaction. Do not use permanent destroy as duplicate cleanup without retention and dependency review.

Recover without duplicating work

Observation	Response
Action is `requires_human`	Complete the existing approval workflow
Action remains pending	Read the same Action; do not replace it
Idempotency collision	Stop; the key owns different canonical input
Definitive provider rejection	Refresh schema and state before a new intent
Retryable create or single-update failure	Reuse only the original Action and key according to runtime policy
Uncertain mutation outcome	Preserve all identities and inspect provider state before another write
Provider record cannot be identified	Stop; do not guess an ID or repeat create

The connector exposes no reconciliation capability. Bounded provider reads and retained evidence must establish current state before a separately authorized recovery mutation.

Retain the management evidence

Keep:

- catalog revision and selected capability digests;
- connector artifact, provider mapping, tenant, workspace, and credential revisions;
- observed OpenAPI body digest and application object-policy revision;
- bounded list and duplicate Actions and their minimized results;
- frozen mutation input and expected prior and resulting values;
- Action ID, idempotency key, approval, policy hash, result, and receipts;
- provider request and validated record IDs;
- post-mutation read and comparison;
- any lifecycle or recovery Action as a separate evidence chain.

This evidence demonstrates one controlled record workflow. It does not by itself establish live-provider qualification or general compatibility for every standard or custom object.

Related documentation

- Twenty OpenAPI documents
- Record query and aggregation
- Record create, update, and batch upsert
- Record delete, restore, and merge
- Twenty connector quickstart