Twenty record create, update, and batch upsert

Use these four mutations to create or update standard and custom Twenty records. The connector binds each request to a fixed workspace, requires approval and idempotency, and adds deterministic create IDs when needed.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/twenty/capabilities/record-create-update-and-batch-upsert.md

The connector validates the outer request and safety guards. Twenty owns the current object schema, writable fields, relation semantics, automation, and the provider-shaped response.

Compare the operations

Operation	Provider request	Required input	Retry contract
`record.create`	`POST /rest/{object}?upsert=true`	`object`, object `body`	Safe with original key
`record.create_many`	`POST /rest/batch/{object}?upsert=true`	`object`, array `body`	Safe with original key
`record.update`	`PATCH /rest/{object}/{id}`	`object`, `id`, object `body`	Safe with original key
`record.update_many`	`PATCH /rest/{object}?filter=...`	`object`, object `body`, non-empty `filter`	Unsafe

All four are high-risk writes. They require tenant-policy approval and an idempotency key and support only synchronous, non-streaming execution.

No operation exposes cancellation, transaction planning, reconciliation, compensation, or provider rollback.

Apply the shared input bounds

Every operation requires `object` matching:

```
TEXT
^[A-Za-z][A-Za-z0-9_]{0,127}$
```

An object body may contain at most 2,048 top-level properties. Serialized provider request JSON cannot exceed 16 MiB.

Every supplied record `id` must be a canonical RFC 4122 UUID with version 1 through 5 and an RFC variant nibble. A valid identifier still does not prove that a record belongs to the configured workspace.

Create one deterministic upsert

`cap:twenty:record.create` accepts an object body and optional `depth` query:

```
JSON
{
```

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

The connector always adds provider `upsert=true`. If `body.id` is absent, it derives this UUID-v5 input string:

```
TEXT
aip:twenty:<workspace_uuid>:<idempotency_key>:0
```

The UUID namespace is `Uuid::NAMESPACE_URL`. The configured workspace and the original key therefore target the same create identity across equivalent attempts.

If the caller supplies `body.id`, the connector validates and preserves it. It does not compare an explicit ID with the deterministic ID.

Forced upsert and deterministic targeting reduce duplicate-create risk. They do not grant authorization, validate the provider body, or establish global exactly-once behavior.

Create a deterministic batch

`cap:twenty:record.create_many` accepts one through 200 object bodies:

```
JSON
{
  "object": "companies",
  "body": [
    {
      "name": "AIP Example Company One"
    },
    {
      "id": "0190c42f-2d5a-7000-8000-000000000020",
      "name": "AIP Example Company Two"
    }
  ],
  "query": {
    "depth": 0
  }
}
```

The connector preserves each valid explicit ID. For a body without `id`, it derives UUID v5 from workspace, key, and the zero-based array index.

```
TEXT
aip:twenty:<workspace_uuid>:<idempotency_key>:<index>
```

Array order is part of deterministic identity. Reordering, inserting, or removing members changes index ownership and requires a new governed intent and key.

The connector forces provider `upsert=true` for the complete batch. It does not promise that the provider commits all records atomically.

Update one record

`cap:twenty:record.update` requires the target ID separately from the body:

```
JSON
{
  "object": "companies",
  "id": "0190c42f-2d5a-7000-8000-000000000020",
  "body": {
    "name": "AIP Example Company Renamed"
  },
  "query": {
    "depth": 1
```

```
  }
}
```

`depth` is the only optional query key. The connector does not reject an empty body at its outer layer; deployment policy should require a meaningful change before approval.

Single update advertises retry safety only with the original idempotency key and unchanged canonical input. A new key represents a different AIP intent.

Update records selected by a filter

`cap:twenty:record.update_many` requires a non-empty provider filter:

```
JSON
{
  "object": "companies",
  "body": {
    "accountOwnerId": "0190c42f-2d5a-7000-8000-000000000030"
  },
  "query": {
    "filter": "id[in]:[0190c42f-2d5a-7000-8000-000000000020]",
    "depth": 0
  }
}
```

The connector rejects a missing filter, whitespace-only filter, literal `{}`, or literal `null`. It does not evaluate the provider filter or calculate the matching record count.

Bulk update advertises unsafe retry even though a key is required. Preserve the original Action and inspect provider state after any ambiguous result.

Use a preflight read to bound the target set. Approval should retain the exact filter, expected count, selected fields, and current workspace schema evidence.

Apply approval

Every operation requires a tenant-policy decision with:

- a reason;
- an input snapshot;
- a policy decision;
- a policy hash matching the frozen request;
- an authenticated approver accepted by the deployment.

The approval window is 900,000 ms. Approval resumes the same queued Action and does not authorize changed input or a replacement Action.

The policy reason states that Twenty mutations can change CRM records, schema, layouts, or webhook delivery policy. These four operations change CRM records, but provider automation can cause wider consequences.

Apply idempotency

Mutation keys must contain 1 through 1,024 ASCII graphic bytes. The contract scope is the external account, collision behavior revalidates the input hash, and the published TTL is 604,800,000 ms.

The connector forwards the key as `Idempotency-Key`. It also uses the key for missing create IDs, but not for explicit update target selection.

Keep these values together:

- Action ID;
- capability ID;
- canonical input and hash;
- configured external account and workspace;
- idempotency key;
- approval ID and policy hash;
- provider request ID and terminal result.

Never reuse a key for another object, body, array order, filter, target, or workspace intent.

Interpret successful results

A completed result has this connector-owned wrapper:

```
JSON
{
  "http_status": 200,
  "provider_request_id": "provider-request-id-or-null",
  "body": {}
}
```

The body is decoded provider JSON without record-shape normalization. The connector records its provider-effect-committed checkpoint after a successful response and before constructing the completed result.

A successful provider HTTP response establishes the returned provider outcome. It does not prove that every automation, notification, webhook, or integration triggered by the record change completed.

Handle failure and uncertainty

Condition	Retryable	Uncertain outcome
Invalid input, key, object, body, UUID, or filter	No	No
Provider `401` or `403`	No	No
Other provider `4xx`	No	No
Provider `429` or `5xx`, create/create-many/update	Yes	Yes
Provider `429` or `5xx`, update-many	No	Yes
Connect failure before provider connection	Create/create-many/update only	No
Other transport failure	Create/create-many/update only when source classifies it safe	Yes
Oversized or invalid provider response	No	Yes
Runtime cancellation branch	No	Yes

`retryable` and `uncertain_outcome` answer different questions. A retry-safe mutation still requires the original identity and key. An unsafe uncertain mutation requires reconciliation before another write.

The connector exposes no reconciliation operation. Read the original Action, retain provider request identity, and inspect the actual provider record or bounded target set through an authorized read.

Security and data boundaries

All four contracts classify input and output as confidential, possibly containing personal information, with redaction required. Arbitrary writable record bodies can include tenant-defined sensitive fields.

The connector fixes origin, workspace, token, and route templates. It does not apply field-level authorization, provider schema migrations, data residency, or retention policy.

Apply least-privilege provider scopes, object and field allowlists, target-count limits, approval separation, evidence redaction, and post-write verification at the deployment and application layers.

Related documentation

- Twenty capability index
- Record query and aggregation
- Twenty connector quickstart
- Approvals and policy
- Errors and retry decisions