



DATABLAU · DAM

REST API integration guide

Product module user manual

Version 9.1.2

Contents

01	DAM developer API reference
02	Authentication and request conventions
03	Metadata APIs
04	Standards and metrics APIs
05	Quality and job APIs
06	Assets and security APIs
07	Lineage and graph APIs
08	REST API integration scenarios

DAM developer API reference

Use DAM REST APIs to integrate enterprise portals, development platforms, catalogs, quality operations, and AI applications. Begin with read-only discovery, definitions, and governance status before adding updates, approvals, or execution.

This guide selects interfaces by business object and scenario. Test endpoints, telemetry, icons, cache operations, service restarts, internal synchronization, permission bypasses, and legacy compatibility endpoints are not recommended integration entry points. Inclusion does not constitute an independent public API stability commitment.

Start with a task

Goal	Guide	Outcome
Explain tables and fields in a portal	Conventions, metadata	Definitions traceable to systems and standards
Synchronize standards and reference values	Standards and metrics	Versioned definitions and reference code snapshots
Integrate quality dashboards or tickets	Quality and jobs	Rules, runs, issues, and verification
Display catalogs and ownership	Assets and security	Authorized assets and classification information
Assess changes before release	Lineage and graphs	Dependencies, transformations, impact
Combine multiple services	Integration scenarios	Call sequence, identity mappings, validation

API base URLs

Combine the deployment entry point with the correct service prefix. For example:
`https://dam.example.com/metadata/entities/123/get`.

Variable	Example base URL	Responsibility
BASE_BASE_URL	https://<host>/base	Systems, tags, shared services
METADATA_BASE_URL	https://<host>/metadata	Sources, tables, fields, reports, metamodels
DOMAIN_BASE_URL	https://<host>/domain	Standards, reference codes, metrics
QUALITY_BASE_URL	https://<host>/quality	Rules, validation tasks, issues
ASSETS_BASE_URL	https://<host>/assets	Catalogs and asset details
DATASECURITY_BASE_URL	https://<host>/datasecurity	Classification, discovery, policies
SQLINK_BASE_URL	https://<host>/sqlink	Lineage, parsing, impact
GRAPH_BASE_URL	https://<host>/graph	Cross-object relationships and asset access
JOB_BASE_URL	https://<host>/job	Scheduling and execution records
USER_BASE_URL	https://<host>/user	Users, organizations, credentials

Versions and compatibility

The baseline is the [current documented release](#). A service's `info.version` alone does not identify the product release. Availability depends on deployed Swagger/OpenAPI and permissions. Archive definitions, service build identifiers, and integration records.

Graph v2 and unified asset access are deployment-dependent capabilities. Confirm availability before including them. Before upgrades, compare methods, paths, schemas, permissions, and enums, then regress representative scenarios. Similar screens do not establish API compatibility.

Examples use fictional hosts and IDs and configured authentication variables. Tables list key parameters; the customer deployment's Swagger defines the complete contract.

Implementation sequence

1. Create a dedicated integration identity with necessary domain and module access.
2. Confirm entry point, credentials, and validity through [request conventions](#).
3. Query a small sample and record IDs, pagination, and response structures.
4. Combine services using [scenarios](#) and compare with DAM pages.
5. Assign incremental synchronization, retries, access changes, and upgrade regression ownership before expanding.

Governance REST APIs query or manage governance objects. [Data Services](#) publishes approved business data APIs. Reading a definition does not authorize access to the underlying data.

Authentication and request conventions

Prepare an identity

Confirm license, functional permissions, and object scope for each target module. Systems, sources, standards catalogs, asset catalogs, and quality issues have separate access scopes. An administrator's visibility does not establish an integration identity's access.

Deployments can use gateways, SSO, or tokens. The definitions do not declare a common `securitySchemes`, so this guide does not assume Bearer, Basic, or a fixed custom header. Have the administrator confirm credential placement, lifetime, and revocation. Personal browser sessions are unsuitable as a permanent unattended integration design.

Current identity and tokens

In **Personal Workspace > Token Management**, generate a short-lived token for temporary testing or a permanent token for a controlled service account. Save it in enterprise secrets management; the complete credential may not be displayed again. Assign an owner, purpose, rotation cycle, and revocation record.

These are authenticated identity-management endpoints, not anonymous password-to-token exchanges:

Method and complete path	Purpose	Inputs or response
GET /user/usermanagement/users/details/self	Verify current identity	Current user details; retain only needed identifiers
POST /user/token/getTokens	List current user's tokens	id, name, expireHours, expiresAt; may include sensitive credentials
POST /user/token/generateTokens	Create token	Body name String, expireHours Long; policy controls availability and lifetime
POST /user/token/updateToken	Update token configuration	Deployed UserTokenUpdateDto
POST /user/token/deleteToken	Revoke token	Deployed UserTokenUpdateDto; check affected consumers

Tokens do not expand account permissions. Keep credentials out of source code, request logs, screenshots, and documents.

URLs, methods, and media types

Use the deployment entry point plus complete path. Swagger servers can contain internal nodes; obtain the approved external entry point. Many queries use **POST**. A method alone does not identify a write, and historical multi-method declarations do not make DELETE or PUT safe query aliases.

Use `application/json` for JSON, preserve Query/Path/Body locations, and follow multipart upload contracts. Exports may return binary data or asynchronous identifiers.

This metadata search uses an approved cookie file for session-based testing. Replace authentication according to deployment. `DAM_BASE_URL` is the actual entry point.

`_bash`

```
curl --fail-with-body --request POST \  
  --cookie "$DAM_COOKIE_FILE" \  
  --header 'Content-Type: application/json' \  
  --data '{"currentPage":1,"pageSize":20,"keyword":"customer","types":["TABLE"]}' \  
  "$DAM_BASE_URL/metadata/entities/searchMetadata"
```

The example preserves HTTPS certificate validation. Use a trusted enterprise certificate chain.

Pagination and responses

Interface	Request fields	List and total
Metadata, sources, published standards	Body <code>currentPage</code> , <code>pageSize</code>	<code>content</code> , <code>totalItems</code>
Reference codes	Body <code>currentPage</code> , <code>pageSize</code>	<code>data</code> , <code>total</code>
Technical quality rules	Body <code>currentPage</code> , <code>pageSize</code>	<code>ruleList</code> , <code>total</code>
Assets	Body <code>pageNum</code> , <code>pageSize</code>	<code>status</code> , <code>msg</code> , open data structure
Published security assets	Body <code>pageNo</code> , <code>pageSize</code> ; schema defaults 1 and 10	<code>status</code> , <code>msg</code> , <code>data</code>
Quality issue rows	Query <code>currentPage</code> , <code>pageSize</code> , starting at 1	<code>rowNames</code> , <code>rowData</code> , <code>totalRow</code>
Generic jobs	Conditions and nested page	<code>content</code> , <code>totalElements</code> , <code>number</code>
Optional unified asset search	Body <code>cursor</code> , <code>limit</code>	<code>items</code> , <code>nextCursor</code> , <code>hasMore</code>

Confirm page origins rather than inferring them from names. Sample open Maps before adapting them; do not invent fields such as `data.list`.

Object identity

- Retain system `categoryId`, source `modelId`, object `objectId`, source system, and schema.
- Metadata search returns `id`; detail and column paths use `objectId`. Verify their mapping.
- Catalog `catalogId`, structure `structureId`, asset `assetId`, and underlying `objectId` differ.
- Standard `domainId` differs from `domainCode`; metrics and reference codes have separate identifiers.
- SQLink node `id` is `String`, not DAM's numeric `objectId`; establish a verified mapping.
- Parse and store Long identifiers without JavaScript safe-integer precision loss.

Errors, retries, and logging

Condition	Action
401 or login redirect	Reauthenticate and detect HTML instead of treating it as JSON
403 or invisible object	Check identity, module, and scope
400 or 404	Check prefix, method, types, and existence
429, busy gateway, temporary 5xx	Bounded backoff for reads; 429 support depends on deployment
200 with business failure	Inspect <code>status</code> , <code>msg</code> , and actual error structure
Custom status such as 599	Retain a redacted summary for the service administrator

No common idempotency key has been verified for writes or job starts. After timeout, query actual status before resubmitting. Log time, method, path, redacted conditions, IDs, and HTTP/business state. Remove credentials, passwords, and raw issue data.

Acceptance checks

1. Query an authorized and an unauthorized sample with the integration identity.
2. Compare names, sources, states, counts, and pagination termination with the UI.
3. Test expired credentials, missing input, missing objects, and service failures.
4. Record deployment build and contract; repeat scenarios after upgrades.

Metadata APIs

Use METADATA_BASE_URL=https://<host>/metadata and /base for system lookup. Locate systems and sources before searching objects and reading definitions, fields, and history. Paths below include service prefixes.

Core interfaces

Method and complete path	Purpose	Key inputs
POST /base/modelCategory/getModelCategories	Systems	No required body; array with categoryId, categoryName
POST /base/modelCategory/getModelCategoryById	System detail	Required Long query categoryId
POST /metadata/models/fromre/page	Harvested sources	Body Long categoryId, modelName, currentPage, pageSize; query Boolean showLogicalModel, default false
GET /metadata/models/{modelId}/schemas	Schemas	Required Long source ID; returns strings
POST /metadata/entities/searchMetadata	Objects	Body modelIds, schemas, types, keyword, currentPage, pageSize
GET /metadata/entities/{objectId}/get	Definition	Required Long object ID
GET /metadata/entities/{objectId}/columns	Table fields	Table ID; optional query keyword
GET /metadata/entities/{objectId}/history	Object revisions	Required Long object ID
POST /metadata/dataReport/search	Reports	Body keyword, String array path, currentPage, pageSize
GET /metadata/dataReport/{dataReportId}	Report and associations	Required Long report ID

Search criteria

modelIds is a Long array; schemas is a String array. Common types are TABLE, COLUMN, VIEW, STORED_PROCEDURE, FUNCTION. Files, APIs, and metamodels also use SHARE_FILE, API, META_MODEL, and typeId. Follow actual metamodel types.

□ json

```
{
  "currentPage": 1,
  "pageSize": 20,
  "modelIds": [1001],
  "schemas": ["demo_sales"],
  "types": ["TABLE", "VIEW"],
  "keyword": "customer"
}
```

Replace fictional source and schema. Responses contain `totalItems`, `currentPage`, `pageSize`, and `content`. Records include `id`, `name`, `type`, `definition`, and `path`; use the returned `id` for detail lookup rather than a display name as a unique key.

Detail fields

Fields	Meaning
<code>objectId</code> , <code>modelId</code> , <code>schema</code> , <code>typeId</code>	Source, granularity, type
<code>physicalName</code> , <code>logicalName</code> , <code>definition</code>	Physical and business names, description
<code>domains</code> , <code>standardCodes</code> , <code>tags</code>	Applicable standards, reference codes, tags
<code>owner</code> , <code>udps</code> , <code>properties</code>	Ownership, custom and technical properties
<code>Field type</code> , <code>notNull</code> , <code>defaultValue</code> , <code>ordinal</code>	Data type, nullability, default, order

The schema spells precision percision; preserve that actual key. History includes `version`, `createTime`, `changeHistory`, and source details. It describes metadata changes, not database row change logs.

Select the query by source

Query reverse-engineered tables, fields, procedures, and functions as dataset objects; use actual types for metamodel instances. DDM dictionaries and DDL produce table/field metadata, not attachment records. Harvested file-server resources are file objects. BI and custom reports share report queries, with possible `relatedTable`, `path`, and `url` fields.

See [Metadata Management](#). Metadata, catalog details, and business data samples are separate; `/entities/data/sample` is not a default integration endpoint here.

Synchronize a data dictionary

1. Map external systems to authorized `categoryId` and `modelId` values.
2. Page through sources, schemas, and types; read details and fields.
3. Store IDs, semantics, technical properties, standards, and synchronization time.
4. Compare history and snapshots. Without a suitable incremental cursor, use controlled scans and differences.
5. Before treating a missing object as retired, exclude permission changes, failed harvesting, and incomplete pagination.

Validate customer objects and record pagination, permissions, and errors.

Create, schedule, and execute harvesting tasks

Source creation and scheduling are separate. Confirm account access, plugins, and schema scope before execution.

Method and complete path	Purpose	Inputs
POST /metadata/jobs/createJob	Create harvest/sync task	DatablauJobDescriptor: name, typeName, boundResourceId, boundResourceType, runCycle, schedule, parameters, disabled, @class
POST /metadata/jobs/startJob	Start now	Long query name described below; String task parameter body
POST /metadata/jobs/stopJob	Stop	Deployed ID and parameter format
POST /metadata/jobs/enableJob /disableJob	Enable or disable schedule	Task ID
POST /metadata/jobs/rescheduleJob	Change schedule	Same query name; schedule body
POST /metadata/jobs/query/jobs/byDto	Find tasks	Name, type, state, resource, ID, cycle, creator, time
POST /metadata/jobs/query/jobResults/byCriteria	Results	Task, instance, state, time
POST /metadata/jobs/query/jobEvents/byCriteria	Events	Instance, event type, time
POST /metadata/simplejobs/status	Simple job state	Long query jobId

The source definition's Chinese query name is \u4efb\u52a1ID in Unicode escape notation. Decode it to the literal parameter name if required; confirm gateway handling. Do not translate wire identifiers.

Example task structure:

json

```
{
  "name": "metadata-sync-demo",
  "typeName": "META_DATA_SYNC",
  "boundResourceId": 1001,
  "boundResourceType": "MODEL",
  "runCycle": "CRON",
  "schedule": "0 0 2 * * ?",
  "parameters": {},
  "disabled": false,
  "@class": "com.datablau.job.dto.DatablauJobDescriptor"
}
```

Reconfirm @class, typeName, schedule, and ID parameter names after upgrades or gateway transformations. After start timeout, check instances before retrying.

Compare models with DDM

Use harvested database metadata as the baseline and a DDM design model as the target: reverse-engineer, edit in DDM, compare, review, and apply approved changes.

Method and complete path	Purpose
GET /metadata/models/{modelId}/compareMappings	Available mappings
GET /metadata/models/{modelId}/{modelVersionId}/compareResult/{mappingId}/{ddmModelId}/{ddmModelVersionId}	Versioned differences
GET /metadata/models/{modelId}/{modelVersionId}/compareResultFile/{mappingId}/{ddmModelId}/{ddmModelVersionId}	Export differences
GET /metadata/models/{modelId}/{mappingId}/compareResultHistory	Comparison history
POST /metadata/models/{modelId}/{modelVersionId}/updateCompareResultFile	Apply reviewed comparison file
POST /metadata/models/model/{baseModelId}/compare/{targetModelId}	Compare models
POST /metadata/models/model/{baseModelId}/compare/{targetModelId}/download	Download report

Comparison does not replace DDM approval and publication. Retain revisions, mappings, reports, and rollback snapshots.

Controlled updates and bulk maintenance

Method and complete path	Contract	Guidance
PUT /metadata/entities/{dataObjectId}/summary	Long ID; DataObjectDto body explicitly requires the complete modified object	Not a partial PATCH with only logicalName
POST /metadata/models/{modelId}/exportAsync	Query tableIds, String tagIds, isSimple; DataObjectExportDto body including schemas	String response requires confirmed job/download handling
POST /metadata/models/{modelId}/updateMetadata/{ignoreError}	Long model ID, Boolean ignoreError, binary file	Use same-environment export template; editable columns follow template

Read and snapshot the complete object, modify only approved fields, submit, and read back. `writable` is an indicator; server authorization remains decisive. Re-read before concurrent updates rather than testing overwrite behavior with empty or omitted fields.

For batches, export, enrich, import, and inspect outcomes. Retain IDs, columns, and failed rows. Validate partial-success behavior for `ignoreError`; it does not guarantee atomicity.

The current definition declares binary upload under `application/json` and only String responses for import/export. Confirm actual media type, task tracking, and downloads with the service owner. Until then, use [UI bulk maintenance](#).

Standards and metrics APIs

Base: DOMAIN_BASE_URL=https://<host>/domain.

Provide published standards, reference values, and metric definitions to modeling, development, reporting, and analytics tools. Check state, revision, and effective scope before treating content as an enterprise standard.

Core interfaces

Method and complete path	Key inputs	Result
POST /domain/domains/domain/getPublicPage	Body currentPage, pageSize, categoryId, keyword; optional domainCode, lastModificationStart, lastModificationEnd	Published standards: content, totalItems
POST /domain/domains/domain/getDomainById	Body String domainId	Definition, domainCode, chineseName, description, dataType, state, version
POST /domain/domains/code/getPage	Body currentPage, pageSize; optional code, name, categoryId, state	Codes: data, total
POST /domain/domains/code/getCode	Body String code	Code and values, state, revision
POST /domain/metricManagement/mapping/details	Required query String metricId	Dimensions, formula, source/table, metricMappings
POST /domain/querytheme/page	Required body currentPage, pageSize; optional keyword, categoryId	Metric query topics: content, totalItems

Time filters use date-time; follow supported format and timezone. Obtain catalog categoryId values from the deployment.

Read a definition

Submit to `/domain/domains/domain/getDomainById`:

📄json

```
{
  "domainId": "example-domain-id"
}
```

`domainId` is an internal identifier, distinct from the business code. Retain `version`, `state`, `lastModification`, and applicable `takeEffectDate`, `expireDate`, and `referenceCode`.

Example body for `/domain/domains/code/getPage`:

📄json

```
{
  "currentPage": 1,
  "pageSize": 20,
  "name": "Customer status"
}
```

The name is illustrative. Read values with `getCode` using the actual code. List entries are not necessarily published. Confirm deployment meanings for state values A, C, D, and X rather than inferring them.

Supply standards to development tools

1. Read applicable published definitions and details.
2. Read associated codes and values into a consistent revision snapshot.
3. Display definition, type, length/precision, nullability, and value constraints.
4. Use [metadata APIs](#) to verify field mappings; reading a definition does not create them.
5. Compare revisions and references after changes and have owners assess model, interface, and rule impacts.

For modification-time scans, overlap windows and deduplicate by `domainId` + `version`. Assess retirement, withdrawal, and lost visibility separately; absence from one incremental response does not establish deletion.

Explain metric sources

Read the definition, then query mappings with `metricId` to check `modelId`, `tableId`, dimensions, and formula. Use [lineage](#) when needed. Definitions, mappings, and actual metric values are separate.

Dynamic SQL metric queries are not a general-purpose data retrieval API in this guide. Deliver values through an approved method such as [Data Services](#), with access, scope, performance, and audit requirements.

Integration validation

Code pagination returns data, total, currentPage, and pageSize. Validate standards pagination, permissions, business errors, definitions, values, and metric results with known customer samples rather than fabricated success responses.

Quality and job APIs

Use `QUALITY_BASE_URL=https://<host>/quality` for quality and the relevant `/base` or `/job` service for cross-module execution state. Retrieve rules, run instances, issue lists, and verification separately. Successful execution does not mean issue-free data.

Quality queries

Method and complete path	Key inputs	Result
POST <code>/quality/quality/rules/tech/page</code>	Body <code>currentPage</code> , <code>pageSize</code> ; optional <code>keyword</code> , <code>modelCategoryId</code> , <code>buRuleId</code> , <code>hasProblem</code>	<code>ruleList</code> , <code>total</code>
GET <code>/quality/quality/rule/tech/{ruleId}</code>	Long rule ID	Targets, type, publication state
POST <code>/quality/quality/jobs</code>	Body <code>currentPage</code> , <code>pageSize</code> ; optional <code>name</code> , <code>modelCategoryId</code> , <code>jobId</code>	Definitions: <code>content</code> , <code>totalElements</code>
GET <code>/quality/quality/jobs/{jobInstanceId}/events</code>	Long instance ID	<code>timestamp</code> , <code>message</code> , <code>status</code> , <code>percent</code>
GET <code>/quality/quality/rule/tasks</code>	No required inputs	Issue lists with rule, result, and verification; array unsuitable for frequent full polling
POST <code>/quality/quality/rules/tasks/{taskId}/data</code>	Path ID; required query <code>currentPage</code> , <code>pageSize</code> , starting at 1; filter body	<code>rowNames</code> , <code>rowData</code> , <code>totalRow</code>

Both `quality` path segments are required by the definition.

Build a quality dashboard

1. Retrieve rules and tasks by system and retain `ruleId`, `jobId`, and `scope`.
2. Read events for actual run instances and confirm completion or failure.
3. Associate issues by `ruleId`, `jobResultId`, `modelCategoryId`, and `batchNum`.
4. Display execution, issue, and verification state separately. Do not mix counts from different batches.

5. Compare one batch's rules, instances, issues, and verification with the UI.

Issue status includes INIT, NOT_START, CONFIRM, ACCEPT, FIXED, VERIFIED, CLOSED, END. verifyStatus includes NOT_VERIFIED_YET, VERIFIED_ALL_PASSED, VERIFIED_PARTIALLY_PASSED, VERIFIED_NO_PASSED, NOT_AVAILABLE. Retain both; closed issues are not necessarily fully verified.

Synchronize tickets

Example detail body for a confirmed taskId:

json

```
{
  "assignStatus": "ALL",
  "solveStatus": "NOT_SOLVED"
}
```

Example path: /quality/quality/rules/tasks/2001/data?currentPage=1&pageSize=20. The ID is fictional. Interpret rowData using rowNames order rather than fixed row object fields.

Retain issue list, rule, batch, and available record key to prevent duplicate tickets. Prefer summary, owner, and a DAM link. Synchronize raw issue data only with explicit data authorization and business need.

Ticket closure does not close or verify a DAM issue. Design bidirectional mappings, ownership, and audit separately. See [remediation and verification](#).

Job systems and asynchronous results

Method and complete path	Job system	Inputs and result
POST /base/instantJob/getJobStatus	Base instant jobs	String query jobId; stage, percent, msg
POST /base/instantJob/getJobResult	Base instant jobs	String query jobId; jobStatus, errorMessage, resultType
GET /metadata/simplejobs/status	Metadata simple jobs	Long query jobId; jobStatus, result, errorMessage
POST /sqlink/job/getJobResult	SQLink instance	Long query jobInstanceId, minimum 1; state, times, result
POST /job/main/query/jobs/byCriteria	Generic definitions	JSON criteria; content, totalElements
POST /job/main/query/jobResults/byCriteria	Generic results	Criteria linked to task and instance
POST /job/main/query/jobEvents/byCriteria	Generic events	Criteria linked to instance

Generic criteria require @type and nested page with currentPage and pageSize. Obtain a confirmed DTO example; the schema does not enumerate every concrete type. Prefer module-specific queries until the contract is known.

Base stages are IN_QUEUE, RUNNING, FAILED, FINISHED; metadata and SQLink enums differ. Use the system returned by the initiating endpoint rather than sending every jobId to /job.

Scheduling controls

Call an administrator-approved business entry point, retain the instance ID, poll to a terminal state, then verify business output. Store definition and instance IDs separately.

The /job/main/startJob definition uses a Chinese query name represented in Unicode escape notation as \u4efb\u52a1ID, with a String body. Decode that name if required by the verified contract. This guide provides no untested one-click harvesting command and does not recommend restart, forced stop, or state reset as routine governance APIs.

Use bounded backoff, total timeout, and alerts. Verify actual instance state after failure before repeating execution. Validate read responses in the target deployment; integration checks should not inadvertently run tasks, modify issues, or download business data.

Assets and security APIs

Use `ASSETS_BASE_URL=https://<host>/assets` for assets and `DATASECURITY_BASE_URL=https://<host>/datasecurity` for classification and security assets. Catalog browsing, asset details, and underlying data access have separate permissions. Preserve publication and authorization workflows.

Catalogs and assets

Method and complete path	Key inputs	Purpose
GET /assets/catalog/structures	Required query String fromPage; confirm value for the entry point	Visible structures with id, name, access and publishing settings
GET /assets/catalog/browse/{structureId}/{catalogId}	Long path IDs	Catalog array, not a paginated asset list
POST /assets/catalog/subCatalog	Required body Long catalogId; isManagement, array status	Child catalogs; states UNPUBLISHED, UNDER_REVIEW, PUBLISHED, OFFLINE
GET /assets/catalog/detail/{id}	Catalog Long id	Details wrapped in status, msg, data
POST /assets/assets/list	Body structureId, catalogId, pageNum, pageSize; optional assetsName, range	Assets; range LIST, BROWSE, MY
GET /assets/assets/detail/{catalogId}/{objectId}	Long catalog and underlying object IDs	assetId, ownership, steward, definition, standards, security

The list uses SearchDto; data is an open Map. Confirm actual fields, page origins, and publication filters. For a business portal, use published, currently visible objects. Management scope, client-supplied usernames, or fields such as noRight must not expand access; the server must enforce identity and scope.

DataAssetDetailDto includes assetId, assetCode, modelId, definition, department, dataManager, domains, securityLevel, and securityCatalog. Preserve structure/catalog/asset/underlying-object mappings. assetId does not replace the path's objectId.

Classification and discovery results

Preserve both occurrences of datasecurity below: one is the service prefix, the other the controller path.

Method and complete path	Inputs	Purpose
GET /datasecurity/datasecurity/catalog/search	Required query String keywords	Search security categories
GET /datasecurity/datasecurity/catalog/classify/{catalogId}	Long path ID	Category details; open response object
POST /datasecurity/datasecurity/check/search/publishassets	Body assetName, categoryId, modelId, schema, tagId, pageNo, pageSize	Published security assets; tagId identifies security level
POST /datasecurity/datasecurity/result/get	Body taskId, modelId, tableId, ruleId, pageNum, pageSize, etc.	Discovery results for review, not necessarily published decisions

Example published asset query:

json

```
{
  "pageNo": 1,
  "pageSize": 20,
  "modelId": 1001,
  "schema": "demo_sales"
}
```

Replace sample source and schema. Responses use status, msg, data; verify actual fields. Discovery suggestions are not reviewed classifications, and no result does not prove non-sensitive data.

Display security guidance in a portal

1. Retrieve authorized catalogs and assets and retain identifiers.
2. Read definitions, department, steward, and security details.
3. Read published security assets and category definitions when needed.
4. Clearly represent missing review, unavailable access, and empty results for owner follow-up.
5. Direct users through [requests and approvals](#) and approved delivery.

Classification results and [security gateway](#) runtime policies are separate. A sensitivity level does not prove masking on every JDBC, API, or direct database path.

The source reference checked OpenAPI definitions without changing catalogs, suggestions, or policies.
Confirm fromPage, open Maps, and pagination using deployment samples.

Lineage and graph APIs

Use `SQLINK_BASE_URL=https://<host>/sqlink` for lineage and `GRAPH_BASE_URL=https://<host>/graph` for cross-object graphs. Lineage explains origins, transformations, and dependencies. A graph can also contain standards or asset associations; not every edge represents computational lineage.

SQLink queries

Method and complete path	Inputs	Result
POST /sqlink/item/searchItems	Body itemName, folderId, String modelId, String typeId, currentPage, pageSize	content, totalItems
POST /sqlink/lineageGraph/query	Body String id, queryType, direction, depth, hasTemp	nodes, relationships
POST /sqlink/lineageGraph/getTransform	Query Boolean isGraph, default true; graph conditions body	startIds, endId, transforms, lineageId, etc.
POST /sqlink/lineageGraph/getImpact	Body id, direction, depth, resourceTypes	Impact objects with identity, type, namespace, direction

Map source system, source, schema, table/field, and SQLink node ID explicitly. Numeric metadata IDs do not substitute for node IDs.

Assess a field change before release

1. Locate the corresponding SQLink node and confirm recent harvesting and parsing.
2. Query downstream with FORWARD, explicit depth, and type constraints.
3. Read transformations and verify expressions and connecting nodes.
4. Generate an impact list for business owner review.
5. Reharvest, parse, and verify key paths after the change.

Example impact body with a placeholder ID:

json

```
{
  "id": "example-sqlink-node-id",
  "queryType": "COLUMN",
  "direction": "FORWARD",
  "depth": 3,
  "resourceTypes": ["TABLE", "COLUMN", "REPORT"],
  "hasTemp": false
}
```

Common direction values are FORWARD, BACKWARD, and ALL; types can include SCHEMA, TABLE, COLUMN, REPORT, REPORT_COLUMN, and FILE. Actual enums may be narrower. Confirm the deployed contract, especially differences from the [legacy SQLink reference](#).

Missing downstream results may reflect incomplete harvesting, parsing, depth, filters, or permissions rather than no impact.

Optional graph and unified asset access

Confirm these deployment-dependent capabilities. Path v1/v2 labels are API names, not product release claims. If unavailable, use the appropriate business service without assuming automatic fallback.

Method and complete path	Inputs	Result
POST /graph/graph/v2/nodes/search	keyword, typeId, page, size	content, total; confirm page origin
POST /graph/graph/v2/rerelations/query	typeId, String objectId, sourceSystem, layers, scope	nodes, relations, lastUpdateTime, truncated
POST /graph/asset-access/v1/discovery/search	query, typeIds, limit, cursor, scope	items, nextCursor, hasMore
POST /graph/asset-access/v1/context/resolve	Array assetKeys, profile	results; inspect each item

Pass the complete returned assetKey unchanged:

🔗json

```
{
  "assetKeys": [{
    "sourceSystem": "<sourceSystem from search result>",
    "typeId": 123,
    "objectId": "<objectId from search result>"
  }]
}
```

Replace all placeholders, including type ID. Valid profile, scope, and scope types are not fully enumerated; obtain confirmed values. `truncated=true` means an incomplete relationship set. Follow `nextCursor` rather than constructing tokens. Retain available `sourceVersion` and `sourceUpdatedAt` for freshness.

DIG boundaries

Reading graphs is separate from publishing a DIG space MCP. Use [context publishing](#) to expose a governed, accepted business scope. Direct graph queries do not grant space documents, approvals, or knowledge access. The source guide checked definitions without parsing, modifying graphs, rebuilding projections, or publishing; validate business output with known deployment samples.

Parser plugin call chain

New job formats use SQLink Parser SPI plugins rather than ordinary REST controllers. REST handles upload, parsing, and queries; the plugin produces `LineageContainer`, steps, and relationships.

Method and complete path	Purpose
POST /sqlink/lineage/getLineageParseEngines	Installed typeName, descriptions, plugin information
POST /sqlink/lineage/uploadLineageFile	Form lineageEngine, objectType, file, folderId, defaultSchema, parse
POST /sqlink/lineage/parseLineage	lineageIds, status, onlyFailed, incrementalParsing
POST /sqlink/task/getTaskParseDetail	Progress, errors, missing tables
POST /sqlink/lineage/getLineageTableByPage	Parsed tables and state
POST /sqlink/lineage/getLineage	File and parsing details
POST /sqlink/lineageGraph/getTransform	Transformation logic
POST /sqlink/lineageGraph/getImpact	Impact

Plugin	Manifest	Input
WhaleTunnel	type=WHALETUNNEL, loadClass=com.datablau.sqlink.plugin.seatunnel.WhaleTunnelLineageParser, build=1.0.0, .json	whaletunnel.job_config, exportData source/transform/sink; datasource_name, source_table_name, result_table_name
Apache SeaTunnel	type=SEATUNNEL, loadClass=com.datablau.sqlink.plugin.seatunnel.SeatunnelLineageParser, build=1.0.0, .json	seatunnel.job_config, plugin_name, input/output, table_path, SQL, CTEs, aliases, transformations

Depend on SPI and standard output objects rather than copying JAR implementation. Report unsupported tasks clearly instead of creating misleading partial lineage. Supply default schema explicitly or use the supported DEFAULT_SCHEMA.

Query a complete table path

POST /sqlink/llm/getLineageOfTables requires namespace, schema, and table:

📄 json

```
{  
  "namespace": "<catalog-or-source>",  
  "schema": "sales",  
  "table": "customer",  
  "direction": "ALL"  
}
```

Use supported FORWARD, BACKWARD, or ALL. Retain full paths, node types, and source systems. A table name alone is ambiguous across sources and schemas.

REST API integration scenarios

These patterns combine existing APIs; they are not built-in automated workflows. Confirm [authentication](#), deployment, scope, and owners first.

Enterprise portal and data dictionary

Help users understand an object's meaning, source, ownership, and standards.

1. Locate systems with `/base/modelCategory/getModelCategories`.
2. Find sources and objects with `/metadata/models/fromre/page` and `/metadata/entities/searchMetadata`.
3. Read `/metadata/entities/{objectId}/get` and `/columns`.
4. Add [asset details](#) while retaining catalog/object mappings.
5. Sample against DAM before scheduled synchronization.

Accept when names, schemas, descriptions, fields, and standards match and revoked permissions invalidate portal caches. Metadata visibility does not grant data downloads.

Standards for development and modeling

1. Read published standards by catalog and details by `domainId`.
2. Read associated reference values and revisions; code lists differ from value lists.
3. Display definition, type, length/precision, and value constraints.
4. Verify field associations and map standard > field > external model.
5. Compare modification time, revision, state, and effective scope.

Accept when definitions match effective revisions and changes have owners. Synchronization does not modify models or prove data compliance. See [standards APIs](#).

Quality dashboards and tickets

1. Map technical rules and tasks to business objects.
2. Read results and events for actual instances.
3. Retain `taskId`, `ruleId`, `jobResultId`, `batch`, and verification state.
4. Synchronize summaries, owners, and DAM links by default; retrieve detail only with authorization.
5. Verify actual remediation and verification after external ticket updates.

Accept stable deduplication keys, separate execution and data results, and protected business detail. See [quality APIs](#).

Pre-release lineage impact

1. Identify the DAM object and corresponding SQLink node.
2. Query bounded downstream lineage through `/sqlink/lineageGraph/query`.
3. Check expressions through `/sqlink/lineageGraph/getTransform`.
4. Review `/sqlink/lineageGraph/getImpact` with actual paths and owners.
5. Reharvest, parse, and regress downstream after release.

Accept verified identity, current parsing, explicit filters, and appropriate treatment of missing lineage. See [lineage APIs](#).

Asset security guidance

1. Browse published assets within the integration identity's scope.
2. Read stewardship, ownership, classification, and level.
3. Use published security assets and category details; distinguish suggestions from reviewed decisions.
4. Direct consumers to formal requests and approved delivery.

Accept authorized visibility and traceable security information. Classification alone does not prove gateway enforcement. See [asset APIs](#).

Trusted context for AI applications

1. If supported, discover assets using `/graph/asset-access/v1/discovery/search`.
2. Resolve complete returned `assetKey` values with `/graph/asset-access/v1/context/resolve` and inspect each result.
3. Supplement with graphs or SQLink, retaining freshness, scope, and truncation indicators.
4. For persistent governed context, follow DIG [end-to-end governance](#) to create, accept, and publish a space MCP.
5. Test same-name objects, missing evidence, unauthorized scope, and outdated files.

Accept answers traceable to authorized objects and valid evidence, with explicit uncertainty. REST calls and space MCP publication are separate implementations.

Release handoff

Area	Deliver
Contract	Methods, complete paths, parameter locations, versions, schemas
Identity	Authentication, scope, lifetime, rotation and revocation owners
Objects	Source systems and verified cross-service ID mappings
Synchronization	Full/incremental method, termination, state/revision handling, cache invalidation
Operations	Limits, backoff, timeout, instance tracking, alerts
Acceptance	Known and unauthorized samples, errors, UI/API comparisons
Upgrades	Contract differences, adaptations, regression results

Obtain actual rate limits, idempotency headers, page limits, and SLAs from deployment owners. Confirm workflow and contracts before extending writes, approvals, or schedules.