# Alaska Soil Data Bank (AKSDB) v2 Documentation

Nic Jelinski

2026-01-01

# Table of contents

# Home

> **Note**
>
> This book is the documentation hub for **AKSDB v2**: a reproducible, citable, and governance-aware system for managing Alaska soil datasets and publishing harmonized products.

## What is AKSDB v2?

AKSDB v2 is a **data + metadata + governance** system for assembling many heterogeneous soil datasets into a consistent, auditable set of **harmonized outputs**—while preserving provenance back to source datasets.

It is designed around four ideas:

- **Stable IDs** for long-lived references (machines + citations)
- **Human slugs** for readable URLs and docs navigation
- **Releases** to publish coherent snapshots of schema + vocab + mappings + products
- **Checksums/manifests** to guarantee integrity and reproducibility

If you remember one thing: *AKSDB is built so that you can always answer "what is this, where did it come from, and what changed?"*

## How to use this documentation

Use this book in one of three ways:

### 1) Read it end-to-end (recommended for contributors)

Start with **Concepts** → **Registry** → **Vocab/Glossary** → **Mapping** → **Products** → **Governance**.

### 2) Jump in based on your task

- **Adding a new source dataset?** Go to **Datasets** and **Mapping**
- **Publishing a harmonized output?** Go to **Products**, **Standards**, and **Releases**
- **Changing schema/vocab/rules?** Go to **Governance**
- **Confused by a term?** Go to **Glossary**
- **Need allowable values?** Go to **Vocab**

### 3) Use it as a reference

Think of **Registry / Vocab / Glossary** as the "API docs" for AKSDB.

## Quick links

- **Start here:** Getting Started
- **Mental model:** Concepts
- **Metadata spine:** Registry
- **Allowed values:** Vocab
- **Definitions:** Glossary
- **Source datasets:** Datasets
- **Harmonized outputs:** Products
- **Crosswalk rules:** Mapping
- **Standards alignment:** Standards
- **Change process:** Governance
- **Copy/paste patterns:** Examples

## The AKSDB object model (bird's-eye view)

AKSDB documentation is organized around a few "objects" you'll see everywhere:

| Object | Key | Where documented | Purpose |
|---|---|---|---|
| **Dataset** | `ds_iid` | Datasets | A source dataset (pre-harmonization) |
| **Product** | `product_slug` (+ canonical ID) | Products | A harmonized publishable output |
| **Entity** | canonical entity ID | Registry | A described thing (table/file/layer/artifact) |
| **Party** | canonical party ID | Registry | A person or organization |

| Object | Key | Where documented | Purpose |
|---|---|---|---|
| **Role** | controlled vocab | Vocab + Standards | Attribution and responsibility |
| **Term** | `term_id` | Glossary | Canonical definition used across docs |

## Public vs private (two-repo worldview)

AKSDB v2 usually operates as two surfaces:

- **Private workspace**: raw inputs, restricted datasets, internal notes, intermediate artifacts
- **Public repo**: harmonized products, documentation, open metadata, citations

This book aims to describe the system in a way that supports both—without leaking private data.

## Guarantees (what AKSDB tries to make true)

AKSDB v2 aims for:

- **Referential stability:** IDs don't change once published
- **Reproducibility:** releases can be rebuilt (same inputs + same rules)
- **Traceability:** products can be traced to datasets and mapping rules
- **Controlled language:** vocabularies and glossary prevent "semantic drift"
- **Governed change:** schema and rules evolve with explicit deprecation/versioning

## Where to go next

If you are new to the project:

1. Read Getting Started
2. Read Concepts (IDs → slugs → releases → checksums)
3. Skim Registry to understand the metadata spine
4. Use Examples to do real work quickly

## Citation and licensing

AKSDB v2 is intended to be **citable** at multiple levels:

- the overall project/release
- each harmonized product
- each source dataset (original citation preserved)

(See **Standards** for how we align with DataCite / EML, and how roles map to CRediT.)

## Project status

This documentation is evolving alongside: - the registry tables and controlled vocabularies, - mapping rules for key sources, - and the first "stable" harmonized product releases.

If something feels incomplete, check **Governance** for how to propose changes and add missing pages.

# Part I

# Concepts::concepts

# 1 Concepts

The AKSDB v2 mental model: IDs → slugs → releases → checksums

## 1.1 Why these concepts exist

AKSDB v2 is designed for: - reproducibility (same inputs → same outputs), - long-lived references (stable identifiers), - safe evolution (versioning + deprecation), - traceability (checksums + provenance).

Everything else in the docs assumes this mental model.

## 1.2 The core chain

1) IDs **IDs** are canonical, stable identifiers for machines and long-term references. They should be:

- unique,
- immutable once published,
- never "meaningful" in a way that invites reinterpretation.

2) Slugs **Slugs** are human-friendly names used for URLs and docs navigation. They can change (with redirects / alias rules), but must never break ID-based references.

3) Releases A **release** is a named snapshot of:

- schema,
- vocab,
- mappings,
- products (harmonized outputs), with explicit compatibility expectations.

4) Checksums **Checksums** anchor trust:

- detect unintended change,

- support manifests,
- enable provenance ("this file came from these inputs under this mapping version").

## 1.3 In this section

- **IDs & slugs**: what is stable, what can change, and how aliases work
- **Releases**: what constitutes a release, and how we version it
- **Checksums & provenance**: manifests, file integrity, and lineage
- **Validation checks**: what we assert about tables and outputs

## 1.4 Suggested next pages

- IDs & slugs
- Releases & versioning
- Checksums & provenance
- Validation checks

# Part II

# Registry::registry

# 2 Registry

Metadata registry tables: packages, entities, parties, roles

## 2.1 What the registry is

The registry is the **metadata spine** of AKSDB v2. It defines the "who/what/where/why" across datasets and products:

- **packages**: publishable units / bundles (datasets, products, releases)
- **entities**: things described by metadata (tables, files, layers, artifacts)
- **parties**: people and organizations
- **roles**: how parties relate to entities (author, curator, funder, etc.)

## 2.2 How to use this section

1. Read the table docs (structure + field meanings)
2. Use the data dictionary to enforce consistent meanings
3. Refer back here whenever you add a dataset/product page

## 2.3 Registry docs

- packages
- entities
- parties
- data dictionary

## 2.4 Design notes

- Registry rows should be referencable by canonical IDs
- Human-readable display names belong in slugs/labels, not in IDs
- Controlled vocabularies should be referenced, not duplicated

# 3 packages::Packages

Notes

**File:** `metadata/registry/packages.csv`
**Grain: 1 row = 1 released snapshot** (a frozen package)
**Purpose:** Dataset/release-level metadata: identity, provenance, coverage, rights, and integrity checks for a snapshot.
**Keys & relationships**

- **Primary key:** `release_id`
- **Stable concept key:** `dataset_id`
- **Immutable handle:** `ds_iid` (slug; used in scripting; must not change once assigned)
- **Joins**
  - `entities.release_id` → `packages.release_id`
  - `party_roles.release_id` → `packages.release_id`

**Required minimum (recommended)**

- `dataset_id`, `ds_iid`, `release_id`, `release_slug`
- `title`, `abstract`
- `ingest_date`
- `license` (or explicit `intellectual_rights`)
- `file_manifest_path`, `manifest_sha256`, `file_count`, `total_bytes`
- 1 creator + 1 contact via `party_roles`

The **packages** registry is the authoritative *release ledger* for AKSDB. Each row records a **frozen snapshot** of a dataset (a published package) with a stable identity, provenance pointers, basic descriptive metadata, coverage fields, rights/access information, and integrity checks (manifest + checksums). Downstream, this table drives dataset/release index pages, enables deterministic joins to `entities` and `party_roles`, and supports reproducible citation/export to metadata standards (e.g., DataCite/EML) by providing a single, versioned source of truth for each released package.

## 3.1 Columns

| Column label | Column name | Type | Format | Description |
|---|---|---|---|---|
| Dataset concept ID | dataset_id | string | UUIDv4 | Stable identifier for the dataset concept across time (all releases/snapshots share the same dataset_id). Enables stable joins and lineage even if names/slugs change (though `ds_iid` should be immutable too). For a given `ds_iid`, `dataset_id` must be consistent across all rows. *Example:* *`550e8400-e29b-41d4-a716-4466`* |
| Dataset handle (slug) | ds_iid | string | `^[a-z0-9]+(-[a-z0-9]+)*$` | Stable, human-readable dataset handle used in scripting and folder naming. |

- Must be globally unique across the registry; never reused.**Validation rules:** - Must match regex.
- Must be unique across all `packages` rows (recommended).
- Must not change for a given `dataset_id`.**Examples:** - Valid: `nrcs-nasis-pedons-ak`
- Invalid: `NRCS_NASIS_AK` — uppercase/underscore | | Release (snapshot) ID | release_id | string | UUID (recommend UUIDv7) | Unique identifier for a frozen snapshot of a dataset at a point in time. | **Required:** yes**Definition:** Unique identifier for a frozen snapshot of a dataset at a point in time.**Purpose:** The join key for all release-scoped metadata: entities/files, roles, QA/QC, derived products.**Validation rules:** - Must be unique within `packages`.

- Must be a valid UUID. | | Release (snapshot) slug | release_slug | string | parseable slug (project convention) | Human-readable identifier for the snapshot, typically derived from ds_iid + dates + internal counter. | **Required:** yes**Definition:** Human-readable identifier for the snapshot, typically derived from `ds_iid` + dates + internal counter.**Purpose:** Readable folder names and provenance in logs/artifacts.**Validation rules:** - Must start with `ds_iid` (recommended).
- Must be unique within the registry.**Examples:** - `nrcs-nasis-pedons-ak--ing20251231--r003` | | Title | title | string | | Human-readable dataset/release title appropriate for citation. | **Required:** yes**Definition:** Human-readable dataset/release title appropriate for citation. | | Abstract | abstract | string | | Short description of what the dataset/release contains and why it exists. | **Required:** yes**Definition:** Short description of what the dataset/release contains and why it exists. | | Purpose | purpose | string | | Optional statement of intended use / motivation (helpful for EML-style metadata). | **Required:** no**Definition:** Optional statement of intended use / motivation (helpful for EML-style metadata). | | Source organization | source_org | string | | Organization responsible for producing or stewarding the upstream dataset. | **Required:** recommended yes**Definition:** Organization responsible for producing or stewarding the upstream dataset. | | Source citation | source_citation | string | | Preferred citation text for the upstream dataset (include DOI if available). | **Required:** recommended yes**Definition:** Preferred citation text for the upstream dataset (include DOI if available). | | Landing page URL | landing_page_url | string | | Public landing page for the dataset or release (repository record, project page). | **Required:** no**Definition:** Public landing page for the dataset or release (repository record, project page). | | DOI | doi | string | | DOI for the dataset release/version, if minted. | **Required:** no**Definition:** DOI for the dataset release/version, if minted. | | EML package identifier | eml_packageId | string | | Repository-specific EML package ID if publishing to an EML-based repository (e.g., EDI/ADC). | **Required:** no**Definition:** Repository-specific EML package ID if publishing to an EML-based repository (e.g., EDI/ADC). | | | repo_scope | | | | | | | repo_identifier | | | | | Repository identifiers | repo_revision | string | | Optional split fields for package identification/versioning within a repository. | **Required:** no**Definition:** Optional split fields for package identification/versioning within a repository. | | | coverage_start | | | | | Coverage window | coverage_end | date or string | `YYYY-MM-DD` (preferred) or `YYYY`/`YYYY-MM` if partial | Temporal extent of the observations represented in this release (not ingest date). | **Required:** no**Definition:** Temporal extent of the *observations* represented in this release (not ingest date). | | Published/updated date (upstream) | published_date | date | | Provider-reported publication or last-updated date for the upstream dataset snapshot. | **Required:** no**Definition:** Provider-reported publication or last-updated date for the upstream dataset snapshot. | | Ingest date | ingest_date | date | | Date this snapshot was frozen/ingested into your harmonization pipeline. | **Required:** yes**Definition:** Date this snapshot was frozen/ingested into your harmonization pipeline. | | | geo_description | | | | | | west | | | | | | | east | | | | | | | south | | | | | | Spatial coverage (bbox + description) | north | string + float | | Overall spatial footprint/description for the

release; bbox coordinates in decimal degrees. | **Required:** no**Definition:** Overall spatial footprint/description for the release; bbox coordinates in decimal degrees. | | Spatial reference | spatial_reference | string | | EPSG code, WKT, or other SRS description if relevant. | **Required:** no**Definition:** EPSG code, WKT, or other SRS description if relevant. | | License | license | string | | License identifier (prefer SPDX where possible). | **Required:** recommended yes**Definition:** License identifier (prefer SPDX where possible). | | Intellectual rights / usage constraints | intellectual_rights | string | | Free-text rights statement when a standard license is not sufficient. | **Required:** no**Definition:** Free-text rights statement when a standard license is not sufficient. | | | methods_summary | | | | | Methods & processing | processing_summary | string | | High-level methods and any key processing steps performed for this release. | **Required:** no**Definition:** High-level methods and any key processing steps performed for this release. | | Upstream lineage | upstream_release_ids | string | `|`-separated list of `release_id` | Release IDs that this release derives from or depends on (lineage). | **Required:** no**Definition:** Release IDs that this release derives from or depends on (lineage). | | File checksum manifest path | file_manifest_path | string | | Relative path (within release folder) to the per-file checksum manifest (e.g., checksums.sha256). | **Required:** yes**Definition:** Relative path (within release folder) to the per-file checksum manifest (e.g., `checksums.sha256`). | | Dataset fingerprint (manifest hash) | manifest_sha256 | string | sha256 hex | SHA-256 of the checksum manifest file. | **Required:** yes**Definition:** SHA-256 of the checksum manifest file. Represents the integrity of the entire release contents. | | File count | file_count | int | | Number of files/entities included in this release. | **Required:** yes**Definition:** Number of files/entities included in this release. | | Total bytes | total_bytes | int | | Sum of size_bytes across entities for this release. | **Required:** yes**Definition:** Sum of `size_bytes` across entities for this release. | | Notes | notes | string | | Free-text notes, known limitations, quirks, etc. | **Required:** no**Definition:** Free-text notes, known limitations, quirks, etc. |

# Part III

# Vocabulary::vocab

# 4 Vocab

Controlled vocabularies: enums, roles, status fields

## 4.1 What belongs here

Anything that should be **standardized** and **validated** belongs in `vocab/`, including: - roles (aligned with CRediT where applicable), - status fields (draft/published/deprecated), - entity types, - license identifiers, - field-level enums.

## 4.2 How vocab is used

- Registry tables reference vocab values
- Validation checks assert only allowed values appear
- Releases snapshot vocab at a point in time

## 4.3 Start here

- Roles vocabulary (registry + standards)
- Status vocabulary (draft → published → deprecated)
- Shared enums used across tables

# 5 Controlled vocabularies standard

This project uses **controlled vocabularies (CVs)** to standardize categorical values across tables, pipelines, and exports. A controlled vocabulary is a curated list of allowed concepts with stable identifiers, labels, definitions, and governance metadata. Using CVs improves interoperability, reduces ambiguity, and enables consistent joins across harmonized datasets.

## 5.1 What a controlled vocabulary CSV represents

- **One CSV = one vocabulary** (one list of concepts for one topic such as methods, roles, units, properties).
- **One row = one concept**.
- The **machine value** used in data products is stored in `term_code`.
- The **stable project identifier** used for joins and governance is stored in `concept_iid`.

## 5.2 Identifiers and namespaces

- `concept_iid` is a stable identifier minted by this project (typically in the `aksdb:` namespace).
- Even if a term originates from an external standard (EML, ISO, etc.), the project still mints its own `concept_iid` so the record can be versioned, governed, and referenced consistently.
- External provenance and mappings are captured in `source`, `exact_match_ids`, and `close_match_ids`.

## 5.3 Standard column set

All controlled vocabulary CSVs in this project use the following columns:

- `concept_iid`
- `term_code`
- `pref_label`
- `alt_labels`

- definition
- status
- created
- modified
- scope_note
- related_ids
- exact_match_ids
- close_match_ids
- replaced_by
- source
- note

### 5.3.1 Governance fields

- status indicates lifecycle state. Allowed values:
  - accepted (current valid term)
  - proposed (candidate term under review)
  - deprecated (do not use for new data; may have a replacement)
  - draft (work-in-progress, not ready for use)
- created and modified track history and **must be ISO 8601**: YYYY-MM-DD.

### 5.3.2 Labels and definitions

- pref_label is the human-facing label used in UIs and reports.
- alt_labels captures synonyms, abbreviations, and common variants to improve search and mapping.
- definition is a short, stable description of the concept's meaning.
- scope_note clarifies boundaries, intended usage, and edge cases.

### 5.3.3 Relationships and mappings

- related_ids can point to other concept_iid values when a useful non-hierarchical relationship exists.
- exact_match_ids is for external identifiers that are equivalent in meaning (for example, an authoritative schema or standard identifier).
- close_match_ids is for external identifiers that are similar but not strictly equivalent (crosswalks).
- replaced_by supports deprecation by pointing to the preferred successor concept when status=deprecated.

## 5.4 Multi-value encoding convention

When a single cell contains multiple values, the project enforces this encoding:

- Enclose the set in curly braces
- Separate values with pipes

Examples:

- `alt_labels = {EC|electrical conductivity|conductivity}`
- `related_ids = {aksdb:foo/a|aksdb:foo/b}`

Empty cells represent "no value".

## 5.5 CSV quoting and commas

CSV uses commas to separate fields. If a cell contains a comma, a newline, or a double quote, the value must be quoted using double quotes. Where possible, keep definitions and notes free of commas to minimize quoting and reduce noisy diffs.

## 5.6 Regex and validation

Regular expressions (regex) are used in validation to ensure consistent value shapes. Common uses:

- Enforce identifier formats (for example, `concept_iid` prefix rules)
- Enforce machine code formats (`term_code`)
- Validate multi-value fields (either empty or `{...}` with `|` separators)
- Validate date formats (`created`, `modified`)

Recommended regex checks:

- Multi-value cell is either empty or wrapped in braces:
  - `^(\{[^{}]*\}|)$`
- Basic term code (letters only, supports CamelCase):
  - `^[A-Za-z][A-Za-z]*$`
- ISO date:
  - `^\d{4}-\d{2}-\d{2}$`

## 5.7 CSVW metadata sidecars

Each controlled vocabulary CSV should include a matching CSVW metadata file:

- `<vocab>.csv-metadata.json`

The CSVW file provides machine-readable descriptions of: - column datatypes - required fields - controlled enums (such as `status`) - separators for multi-value fields

This supports automated validation and consistent parsing across tools.

### 5.7.1 What CSVW is for

CSVW (CSV on the Web) is a W3C standard for describing CSV files with machine-readable metadata. In this project, the CSVW sidecar is used as the **authoritative schema** for each vocabulary so that scripts and pipelines do not have to guess how to interpret the CSV.

### 5.7.2 How to interpret the CSVW file

Key fields you will see:

- `url`: which CSV file this metadata describes.
- `dialect`: how the CSV is formatted (delimiter, quote character, encoding, and whether there is a header row).
- `tableSchema.primaryKey`: the column or columns that uniquely identify rows.
- `tableSchema.columns`: per-column rules:
  - `name`: the exact column header in the CSV.
  - `required`: whether empty values are allowed.
  - `null`: which values are treated as "missing" (this project uses `""`).
  - `datatype`: the expected datatype (for example `date`, `string`, `anyURI`) and optionally a `format` pattern (regex-like).
  - `separator`: how to split multi-valued cells (this project uses `|` in columns whose CSV values are encoded as `{a|b|c}`).

### 5.7.3 How to use CSVW in this project

Use the CSVW file in any workflow that reads vocabularies:

- **Validation (recommended in CI):**
  - check required fields are present
  - check `status` is one of the allowed enum values

  - check `created` and `modified` are valid ISO dates
  - check multi-value fields conform to `{...}` and split cleanly on `|`

- **Parsing (ETL and ingestion):**
  - use the `separator` metadata to parse multi-value fields into arrays/lists consistently
  - use `datatype` to parse and type-cast fields consistently (for example dates)

- **Documentation generation (optional):**
  - column names and constraints can be derived from CSVW to keep validation rules synchronized with the CSV

### 5.7.4 Canonical multi-value parsing rule

For any column that is both: - encoded as `{a|b|c}` in the CSV, and - has `"separator": "|"` in CSVW,

the canonical parse is:

1) if blank → empty list

2) else strip leading `{` and trailing `}`

3) split on `|`

4) trim whitespace on each value

## 5.8 Documentation format for each vocabulary

Each vocabulary must have a companion Markdown documentation page that includes:

1. A brief narrative introduction (rationale + what the vocabulary is for)
2. A "Controlled vocabulary file" section with:
   - controlled vocabulary name / table name (filename)
   - label (human-readable label)
3. A "Columns" section with a table:
   - Column label (human-readable)
   - Column name (exact CSV header)
   - Description (detailed)
4. A "References" section that points to the authority defining the terms and any mapping standards

# 6 eml_responsibility_roles

This controlled vocabulary enumerates the allowed **EML Party responsibility roles** (EML `RoleType`) used to describe how a person or organization is associated with a dataset or other resource (for example, author, originator, custodian/steward, distributor, and point of contact). Each row provides a stable AKSDB identifier (`concept_iid`) for joining and governance, and the exact EML machine value (`term_code`) that should be written into EML metadata.

The vocabulary also includes optional crosswalks to ISO 19115 `CI_RoleCode` values via `close_match_ids` when a reasonable "close match" exists.

## 6.1 Controlled vocabulary file

- **Controlled vocabulary name / table name:** `/metadata/vocab/eml_responsibility_roles.csv`
- **Label:** EML responsibility roles

## 6.2 Columns

| Column label | Column name | Description |
|---|---|---|
| Concept identifier | `concept_iid` | Stable, project-minted identifier for the concept. Used as the primary key for joins and internal governance (do not recycle). Pattern in this vocab: `aksdb:eml_responsibility_role/<RoleType` |
| EML role code | `term_code` | The exact machine-readable EML `RoleType` literal value to write into EML (case sensitive). Examples include `principalInvestigator` and `pointOfContact`. |

| Column label | Column name | Description |
|---|---|---|
| Preferred label | `pref_label` | Human-readable display label for the role. |
| Alternative labels | `alt_labels` | Optional synonyms and variants for search and display. Multi-values are encoded as `{value1\|value2\|...}`. |
| Definition | `definition` | Short definition of the role concept (project-readable). |
| Status | `status` | Governance status for the concept. Allowed values: `accepted`, `proposed`, `deprecated`, `draft`. |
| Created | `created` | Date the row/concept was introduced to the vocabulary in ISO format `YYYY-MM-DD`. |
| Modified | `modified` | Date the row/concept was last changed in ISO format `YYYY-MM-DD`. |
| Scope note | `scope_note` | Optional usage guidance and boundaries for applying the role. |
| Related identifiers | `related_ids` | Optional related concept identifiers (non-hierarchical). Multi-values are encoded as `{value1\|value2\|...}`. |
| Exact match identifiers | `exact_match_ids` | External identifier(s) that are considered exact matches. In this vocab this is the authoritative EML XSD URL used as the source definition for RoleType. |
| Close match identifiers | `close_match_ids` | External identifier(s) that are close (not exact) matches. Here this is used for ISO 19115 `CI_RoleCode` crosswalks when applicable. Multi-values are encoded as `{value1\|value2\|...}`. |

| Column label | Column name | Description |
|---|---|---|
| Replaced by | `replaced_by` | If `status=deprecated`, the `concept_iid` of the preferred replacement concept. |
| Source | `source` | Human-readable documentation URL for the authority defining the vocabulary values (here: EML Party/RoleType documentation). |
| Note | `note` | Optional editorial notes and provenance details. Multi-values are encoded as `{value1\|value2\|...}`. |

## 6.3 References

- EML Party / RoleType documentation: see `source` column in `eml_responsibility_roles_iso.csv`.
- EML 2.1.1 Party schema (XSD): see `exact_match_ids` column in `eml_responsibility_roles_iso.csv`.
- ISO 19115 `CI_RoleCode` codelist (used for close matches): see the ISO codelist URL referenced in the `note` column for each mapped term.

# 7 datacite_contributor_roles

This controlled vocabulary enumerates the allowed **DataCite Contributor** `contributorType` values (roles) that may be used to describe the contribution made by a person or organization in DataCite metadata.

The vocabulary also includes optional crosswalks to the **CRediT contributor roles taxonomy** via `close_match_ids` when a reasonable "close match" exists.

## 7.1 Controlled vocabulary file

- **Controlled vocabulary name / table name:** `/metadata/vocab/datacite_contributor_roles.csv`
- **Label:** DataCite contributor roles

## 7.2 Columns

| Column label | Column name | Description |
|---|---|---|
| Concept identifier | `concept_iid` | Stable, project-minted identifier for the concept. Recommended pattern for this vocab: `aksdb:datacite_contributor_role/<Contri` |
| Preferred label | `label` | Human-readable label for the role (typically the same as the DataCite controlled value, optionally spaced for readability). |
| Term code | `term_code` | The exact machine-readable DataCite `contributorType` literal value that should be written into DataCite metadata. |

| Column label | Column name | Description |
|---|---|---|
| Definition | `definition` | Short definition of the role (sourced from DataCite controlled list definitions where available). |
| Status | `status` | Governance status for the concept. Allowed values: `accepted`, `proposed`, `deprecated`, `draft`. |
| Created | `created` | Date the row/concept was introduced to the vocabulary in ISO format `YYYY-MM-DD`. |
| Modified | `modified` | Date the row/concept was last changed in ISO format `YYYY-MM-DD`. |
| Scope note | `scope_note` | Optional usage guidance and boundaries for applying the role. |
| Related identifiers | `related_ids` | Optional related concept identifiers (e.g., broader/narrower/hierarchical). Multi-values are encoded as `{value1\|value2\|...}`. |
| Exact match identifiers | `exact_match_ids` | External identifier(s) for the authoritative DataCite XSD enumeration and/or per-term documentation anchor used as the source definition for `contributorType`. Multi-values are encoded as `{value1\|value2\|...}`. |
| Close match identifiers | `close_match_ids` | External identifier(s) for close matches (e.g., CRediT role URLs) where applicable. Multi-values are encoded as `{value1\|value2\|...}`. |
| Replaced by | `replaced_by` | If `status=deprecated`, the `concept_iid` of the preferred replacement concept. |

| Column label | Column name | Description |
|---|---|---|
| Source | `source` | Human-readable documentation URL for the authoritative definition of the vocabulary values (here: DataCite `contributorType` documentation). |
| Note | `note` | Optional editorial notes and provenance details. Multi-values are encoded as `{value1\|value2\|...}`. |

## 7.3 References

- DataCite `contributorType` controlled list definitions: https://datacite-metadata-schema.readthedocs.io/en/latest/appendices/appendix-1/contributorType/
- DataCite `contributorType` schema enumeration (XSD include): https://schema.datacite.org/meta/kernel-4.6/include/datacite-contributorType-v4.xsd
- CRediT role descriptors (used for close matches): https://credit.niso.org/contributor-roles-defined/

# 8 credit_roles

This controlled vocabulary enumerates the allowed **CRediT (Contributor Role Taxonomy)** contributor roles.

The vocabulary uses a stable internal identifier (`concept_iid`) and a machine-friendly code (`term_code`) that can be used in structured metadata. The authoritative role descriptor pages are referenced via `exact_match_ids`.

## 8.1 Controlled vocabulary file

- **Controlled vocabulary name / table name:** `/metadata/vocab/credit_roles.csv`
- **Label:** CRediT roles

## 8.2 Columns

| Column label | Column name | Description |
|---|---|---|
| Concept identifier | `concept_iid` | Stable, project-minted identifier for the concept. Recommended pattern for this vocab: `aksdb:credit_role/<RoleCode>`. |
| Preferred label | `pref_label` | Human-readable label for the role (as shown in CRediT). |
| Term code | `term_code` | Machine-friendly role code (letters only) used for clean programmatic identifiers. |
| Alternate labels | `alt_labels` | Optional synonyms / label variants. Multi-values are encoded as `{value1\|value2\|...}`. |

| Column label | Column name | Description |
|---|---|---|
| Definition | `definition` | Short definition of the role (from the authoritative CRediT role descriptors). |
| Status | `status` | Governance status for the concept. Allowed values: `accepted`, `proposed`, `deprecated`, `draft`. |
| Created | `created` | Date the row/concept was introduced to the vocabulary in ISO format `YYYY-MM-DD`. |
| Modified | `modified` | Date the row/concept was last changed in ISO format `YYYY-MM-DD`. |
| Scope note | `scope_note` | Optional usage guidance and boundaries for applying the role. |
| Related identifiers | `related_ids` | Optional related concept identifiers (e.g., broader/narrower/hierarchical). Multi-values are encoded as `{value1\|value2\|...}`. |
| Exact match identifiers | `exact_match_ids` | External identifier for the authoritative role descriptor (CRediT per-role URL). |
| Close match identifiers | `close_match_ids` | Optional external identifiers for close matches (e.g., other role taxonomies). Multi-values are encoded as `{value1\|value2\|...}`. |
| Replaced by | `replaced_by` | If `status=deprecated`, the `concept_iid` of the preferred replacement concept. |
| Source | `source` | Human-readable documentation URL for the authority defining the vocabulary values (here: CRediT role descriptors). |

| | | |
|---|---|---|
| Note | `note` | Optional editorial notes and provenance details. Multi-values are encoded as `{value1\|value2\|...}`. |

## 8.3 References

- CRediT role descriptors (authoritative list + definitions): https://credit.niso.org/contributor-roles-defined/
- CRediT home (overview + role list): https://credit.niso.org/
- Guidance on using/implementing CRediT: https://credit.niso.org/implementing-credit/
- CRediT as an ANSI/NISO standard (background): https://casrai.org/credit/

# Part IV

# Glossary::glossary

# 9 Glossary

Canonical definitions of terms used across AKSDB

## 9.1 Purpose

The glossary prevents “same word, different meaning” drift.

A glossary entry should: - define the term precisely, - list synonyms/aliases, - specify scope (where it applies), - link to related registry fields and standards mappings.

## 9.2 In this section



## 9.3 Authoring rule of thumb

If a term appears in: - registry field descriptions, - mapping rules, - product documentation, and could be interpreted multiple ways—add a glossary entry.

# 10 Terms

#### 10.0.0.1 slug

DEFINITION:

COMMENT:

REFERENCE:

#### 10.0.0.2 UUIDv4

DEFINITION:

COMMENT:

REFERENCE:

# Part V

# Datasets::datasets

# 11 Datasets

Per-source dataset documentation (keyed by ds_iid)

## 11.1 What a dataset page is

A dataset page documents a **source dataset** before (or alongside) harmonization: - what it is and where it came from, - spatial/temporal coverage, - collection and processing notes, - access constraints (public vs private), - how it maps into AKSDB v2 entities/products.

Each dataset page is keyed by `ds_iid`.

## 11.2 What every dataset page should include

- Summary + citation
- Access + license
- Provenance (who, when, how acquired)
- Primary artifacts (tables/files/layers)
- Known issues and QC notes
- Mapping links (crosswalks used)

## 11.3 Dataset pages

- MTJ Permafrost
- KPU GAAR

# Part VI

# Products::products

# 12 Products

Harmonized outputs (keyed by product_slug)

## 12.1 What a product is

A **product** is a harmonized, documented output intended for reuse: - tables, - geospatial layers, - derived indicators, - releases of cleaned/harmonized datasets.

Products are keyed by `product_slug` (human-friendly), but should also carry canonical IDs and checksums.

## 12.2 What every product page should include

- What it contains (schema + files)
- Intended use + non-use cases
- Inputs (datasets + versions)
- Mapping rules used
- Validation checks passed
- Change notes across releases

## 12.3 Start here

Create product pages once: - mapping rules stabilize, - validation checks are defined, - and you are ready to publish/release.

# Part VII

# Mappings::mappings

# 13 Mappings

Crosswalks, transformation rules, and harmonization logic

## 13.1 Why mapping docs matter

Mappings are where “interpretation” happens: - column crosswalks, - unit conversions, - depth/horizon rules, - categorical harmonization, - join keys and entity resolution.

These docs make harmonized products defensible and reproducible.

## 13.2 What a mapping doc should include

- Inputs and outputs (tables/fields)
- Rule statements (human-readable)
- Edge cases and precedence rules
- Examples (before/after)
- Version notes (what changed and why)

## 13.3 Typical mapping topics

- Source → canonical field crosswalks
- Controlled vocab normalization
- ID assignment rules
- Checksums/manifests generated per run

# Part VIII

# Standards::standards

# 14 Standards

How AKSDB aligns with EML, DataCite, and CRediT

## 14.1 Purpose

Standards provide interoperability and citation-quality metadata.

This section documents: - which standards you reference, - what parts you implement, - how registry fields map to them, - where you intentionally diverge (and why).

## 14.2 Pages

- EML
- DataCite
- CRediT
- CF Conventions

## 14.3 Practical output

The goal is to support: - a strong `CITATION.cff`, - dataset/product citations, - machine-readable exports (where appropriate), - consistent attribution via roles.

# Part IX

# Governance::governance

# 15 Governance

How schema, docs, vocab, and mappings change over time

## 15.1 What governance covers

Governance defines: - how you propose changes, - how you version them, - how you deprecate safely, - what guarantees users can rely on.

This is what keeps AKSDB v2 coherent as it grows.

## 15.2 Pages

- Schema versioning
- Adding a field
- Deprecating a field
- Glossary rules
- Roles and attributes

## 15.3 Governance principle (default)

Prefer: - additive changes, - explicit deprecations, - compatibility notes per release, over silent breaking changes.

# Part X

# Examples::examples

# 16 Examples

Minimal worked patterns you can copy-paste

## 16.1 What belongs here

Examples are small, opinionated templates for: - adding a dataset page, - adding a product page, - writing a mapping doc, - defining a new controlled vocabulary, - running validation checks, - producing a release manifest (checksums + provenance).

## 16.2 Suggested starter examples

- “Add a dataset” template
- “Add a product” template
- “Mapping doc skeleton”
- “Registry row examples” (packages/entities/parties/roles)
- “Release checklist” (what must be true to publish)