Portolan Specification

v0.2.0

2026-08-28

Contents

Portolan Specification — Core 6
- Introduction 6
- Core Structure 7
- Conformance and Versioning 7
- Recommended STAC Extensions 8
- Catalogs 8
- Collections 8
  - Single-File Collections 8
  - Raster Collections 8
- Nested Catalogs, Flat Collections 9
- Items 9
- Assets 9
- Links 11
  - Alternate-Language Trees 11
- Human-Readable Titles 12
- Bounding Boxes and Spatial Extent 12
- Temporal Metadata 12
- Data Storage 12
- Providers 13
- Source Provenance 13
- License 14
- AGENTS.md 14
- README.md 14
- Catalog Logo 14
- Metadata 15
- Visualization 15
  - Visualization Styles 15

Portolan Specification — Formats 16
- Vector 16
  - GeoParquet 16
  - PMTiles 17
  - Partitioned Collections 18
- Raster 18

Tabular (Non-Geospatial) 20
Point Cloud 21
Requirements 21
PORTO-CORE-001 21
PORTO-CORE-002 21
PORTO-CORE-003 21
PORTO-CORE-004 21
PORTO-CORE-005 21
PORTO-CORE-006 22
PORTO-CORE-007 22
PORTO-CORE-008 22
PORTO-CORE-009 22
PORTO-CORE-010 22
PORTO-CORE-011 22
PORTO-CORE-012 22
PORTO-CORE-013 23
PORTO-CORE-014 23
PORTO-CORE-015 23
PORTO-CORE-016 23
PORTO-CORE-017 23
PORTO-CORE-071 23
PORTO-CORE-072 23
PORTO-CORE-018 23
PORTO-CORE-078 24
PORTO-CORE-019 24
PORTO-CORE-020 24
PORTO-CORE-021 24
PORTO-CORE-022 24
PORTO-CORE-023 24
PORTO-CORE-024 24
PORTO-CORE-025 25
PORTO-CORE-026 25
PORTO-CORE-027 25
PORTO-CORE-028 25
PORTO-CORE-029 25
PORTO-CORE-030 25
PORTO-CORE-031 25
PORTO-CORE-032 26
PORTO-CORE-033 26
PORTO-CORE-081 26
PORTO-CORE-035 26

PORTO-CORE-036 26
PORTO-CORE-079 26
PORTO-CORE-080 26
PORTO-CORE-037 27
PORTO-CORE-038 27
PORTO-CORE-039 27
PORTO-CORE-040 27
PORTO-CORE-041 27
PORTO-CORE-042 27
PORTO-CORE-043 28
PORTO-CORE-044 28
PORTO-CORE-045 28
PORTO-CORE-073 28
PORTO-CORE-046 28
PORTO-CORE-047 28
PORTO-CORE-048 29
PORTO-CORE-049 29
PORTO-CORE-050 29
PORTO-CORE-051 29
PORTO-CORE-052 29
PORTO-CORE-053 29
PORTO-CORE-054 29
PORTO-CORE-055 29
PORTO-CORE-056 30
PORTO-CORE-057 30
PORTO-CORE-058 30
PORTO-CORE-059 30
PORTO-CORE-060 30
PORTO-CORE-061 30
PORTO-CORE-062 30
PORTO-CORE-063 31
PORTO-CORE-074 31
PORTO-CORE-075 31
PORTO-CORE-076 31
PORTO-CORE-077 31
PORTO-CORE-064 31
PORTO-CORE-065 31
PORTO-CORE-066 32
PORTO-CORE-067 32
PORTO-CORE-068 32
PORTO-CORE-069 32

PORTO-CORE-070 32
PORTO-FMT-001 32
PORTO-FMT-002 33
PORTO-FMT-003 33
PORTO-FMT-045 33
PORTO-FMT-004 33
PORTO-FMT-005 33
PORTO-FMT-006 33
PORTO-FMT-044 34
PORTO-FMT-007 34
PORTO-FMT-008 34
PORTO-FMT-009 34
PORTO-FMT-010 34
PORTO-FMT-011 34
PORTO-FMT-012 35
PORTO-FMT-013 35
PORTO-FMT-014 35
PORTO-FMT-015 35
PORTO-FMT-016 35
PORTO-FMT-017 35
PORTO-FMT-018 35
PORTO-FMT-019 36
PORTO-FMT-020 36
PORTO-FMT-021 36
PORTO-FMT-022 36
PORTO-FMT-023 36
PORTO-FMT-024 36
PORTO-FMT-025 36
PORTO-FMT-026 37
PORTO-FMT-027 37
PORTO-FMT-028 37
PORTO-FMT-029 37
PORTO-FMT-030 37
PORTO-FMT-031 37
PORTO-FMT-032 37
PORTO-FMT-033 38
PORTO-FMT-034 38
PORTO-FMT-035 38
PORTO-FMT-036 38
PORTO-FMT-037 38
PORTO-FMT-038 38

PORTO-FMT-039 38
PORTO-FMT-040 39
PORTO-FMT-041 39
PORTO-FMT-042 39
PORTO-FMT-043 39
PORTO-FMT-046 39
PORTO-FMT-047 39
PORTO-FMT-048 39
Best Practices 40
Contents 40
Conversion Defaults 40
Raster (COG) 40
Partitioning 41
Thumbnails 41
Best Practices — Documentation 41
Two files, two audiences 41
Evaluate the data, not just describe it 41
What belongs in AGENTS.md 42
What makes a good README.md 43
Keep the documentation honest 44
Best Practices — Git-Backed Catalogs 44
Why use git for catalog management? 44
Keep metadata in git, not data 44
Separate published metadata from repository files 45
Links and the publish step 45
Keep the tools and inputs 45
Generate large catalogs 46
Validate changes with continuous integration 46
Handle validator differences 46
Record accepted deviations 46
Link a catalog to its repository 47
Include contribution guidance 47
Catalogs worth studying 47
Status 48
Best Practices — The Grader 48
The ladder 48
How to read the tiers 49
1. Required files and links 49
2. Metadata integrity 49
3. Cloud-native assets 49
4. Agent documentation 49

5. Visualization 50
6. Performance and first render 50
7. Maintenance signals 50
The A+ judgment 50
The overall grade 51
Best Practices — Multilingual Catalogs 51
The Basic Layout 51
Language Fields 52
Links Between Trees 52
Links to Untranslated Content 53
What to Translate 53
Agent Guidance 53
One Source Language 54
A Glossary for Important Terms 54
Keep Translations Current 54
Check the Result 54
Best Practices — Catalog Philosophy 55
Design for the person who shows up 55
Scope by steward, not by theme 55
Shallow and uniform beats deep and clever 55
Names outlive everything 55
Maintenance is the product 55
Catalogs worth studying 56
This is a discussion 56
Best Practices — Styling 56

Portolan Specification — Core

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in BCP 14 [RFC 2119] [RFC 8174] when, and only when, they appear in all capitals, as shown here.

This document defines the requirements that apply to every Portolan catalog, regardless of data format. Format-specific requirements live in `formats.md`.

Introduction

The goal of Portolan is to make it easier and cheaper for data providers to publish their data, and to make that data more accessible to both humans and agents. If Portolan is successful then all the world's spatial data will be much easier to query and utilize in decisions that affect all of our lives.

The center of Portolan is the SpatioTemporal Asset Catalog specification - every Portolan implementation is a STAC catalog, and can be used with any STAC tooling. But where STAC can be used with any data format and can be implemented as an API or files on cloud storage, Portolan requires well-formed cloud-native formats, stored as files on browser-accessible online services. All Portolan implementaions are static catalogs - there is no 'Portolan API' like the STAC API, as

many of the key benefits of Portolan, like scalability and lower cost, are achieved by the full embrace of cloud-native geospatial.

Portolan further requires that all catalogs follow the best practices to guide AI agents to make use of them. Today that means adding an agents.md file to every catalog and collection and building up a set of 'skills' that guide agents to make use of the cloud-native formats. In the future this will likely evolve as the general best practices for agents continue to advance.

The specification aims to standardize the minimum requirements for a 'great' catalog, and to provide guidance to go beyond that. But really the hope is that everyone goes well beyond the minimum and tries to make each catalog better than before. The aspiration of the authors of the specification is to build a real community of collaborators who are all working together to build great data catalogs and share tools and best practices so all can benefit.

Core Structure

A Portolan catalog is a directory of STAC metadata and cloud-native geospatial data. It MUST be a valid STAC catalog following STAC 1.1.0, with a root `catalog.json` always required, even for a single collection.

A catalog is a tree of catalogs, collections, items, and assets: catalogs and collections are internal nodes, items are leaves, and assets (data files) MUST sit at the collection or item level. Catalogs organize only and MAY nest into sub-catalogs; collections MUST be one level deep, containing only items or assets, never nested collections. Every catalog and sub-catalog MUST contain a `catalog.json`, an `AGENTS.md`, and a `README.md`; every collection MUST contain a `collection.json`, an `AGENTS.md`, and a `README.md`.

A catalog has two entry points: `catalog.json` for machines (required at the root) and `README.md` for humans. It is fully navigable; every catalog, collection, and item links to the rest, so any object is reachable from any other (see Links).

```
project/
├── catalog.json
├── AGENTS.md
├── README.md
└── {collection_id}/
    ├── collection.json
    ├── AGENTS.md
    ├── README.md
    └── {item_id}/
        └── data.parquet
```

Conformance and Versioning

A Portolan object declares conformance through the Portolan STAC extension, whose versioned schema URI carries the specification version the object was authored against.

Every catalog and collection MUST declare the versioned Portolan schema URI (e.g. `https://schemas.portolan-sdi.org/portolan/v0.2.0/schema.json`) in its `stac_extensions` array. The schema URI is the single signal of specification version; no separate version property is defined. Declaration happens at the catalog and collection level only, since assets cannot carry `stac_extensions`, and items inherit conformance from their collection.

The specification version MUST NOT be conflated with the dataset version. Dataset versioning, where used, MUST use the STAC version extension.

All objects within a catalog SHOULD declare the same Portolan schema URI. A validator MUST flag any catalog or collection whose declared URI differs from the root catalog's; such a mismatch is a warning, not an error, and a mixed-version catalog remains valid.

Declaring the Portolan extension is a claim of conformance, not proof of it. An object conforms to this specification only by passing the Portolan validator. Validation runs in separable passes:

- **Structural validation** against the STAC 1.1.0 schemas.
- **Portolan metadata validation**, covering every requirement checkable from metadata alone.
- **Portolan data validation**, covering requirements that require reading asset bytes, such as GeoParquet spatial ordering, row-group statistics, and embedded COG statistics.

The metadata passes MUST be executable without reading asset data; the data pass MAY be run independently.

Portolan schemas are published at `schemas.portolan-sdi.org`. `versions.json` is a tooling artifact and is NOT REQUIRED in a catalog for v0.1.

Recommended STAC Extensions

Consistent with the reuse-first approach, Portolan datasets use established STAC extensions wherever they apply rather than re-encoding the same information in `portolan`. Which extensions are required versus recommended, the exact schema URIs and versions to pin, and their usage are defined normatively by the Portolan STAC Profile (see `stac/`), not restated here.

Catalogs

The canonical entrypoint is a STAC `catalog.json` in the catalog root, which MUST be valid. Catalogs MAY use nested sub-catalogs for organization but MUST NOT use nested collections.

Collections

Each collection lives in a subdirectory named with its collection ID. For a nested collection the ID is a POSIX path (e.g. `environment/air-quality`) and the directory sits under one or more intermediate catalog directories (see Nested Catalogs, Flat Collections). A collection directory MUST contain a `collection.json` and holds one subdirectory per item. Collection IDs SHOULD contain only lowercase letters, numbers, hyphens, and underscores, start with a letter, and be unique within the catalog.

Single-File Collections

When a collection holds a single data file (e.g. one GeoParquet file), that data MUST be represented as a collection-level asset — no item directory or item JSON is needed (see Vector). Alongside the `collection.json` and the `.parquet`, such a collection may optionally carry a `.pmtiles`, a `thumbnail.png`, and a `styles/` directory.

Raster Collections

A collection holding multiple raster scenes MUST model each scene as an item carrying its COG as an item-level asset; scene COGs MUST NOT be listed as collection-level assets. Per-scene items are what let each scene carry its own footprint and acquisition time, which a flat asset list cannot

express. A collection holding a single COG follows the single-file rule above: it MUST expose that COG as a collection-level asset with no item directory (see Raster).

Nested Catalogs, Flat Collections

Portolan organizes hierarchy with nested catalogs, not nested collections: intermediate levels are catalogs (`catalog.json`) and collections are always leaves. A collection MUST NOT contain a child collection. This keeps collections flat for STAC API compatibility while still allowing thematic organization above them. A nested collection's ID is its POSIX path from the catalog root (e.g. `environment/air-quality`); `portolan add` writes a `catalog.json` at each intermediate level and links parent to child down to the leaf `collection.json`.

In directory terms, the root holds the root `catalog.json`, a level above a collection holds an intermediate (thematic) `catalog.json`, a data directory holds the leaf `collection.json`, and a subdirectory within a collection may hold a `catalog.json` that organizes many items. Deep nesting is allowed, with every level above the leaf a catalog; a catalog may also appear below a collection to organize its items (for example, a raster collection grouping items by year). A catalog or collection with twenty or more children SHOULD organize them into subcatalogs, thematic or otherwise, so users can browse the data.

```
environment/
├── catalog.json              ← intermediate catalog
├── air-quality/
│   ├── collection.json       ← leaf collection (data)
│   └── pm25.parquet
└── water-quality/
    ├── collection.json       ← leaf collection (data)
    └── turbidity.parquet
```

A `collection.json` with a child collection subdirectory beneath it is invalid.

Items

Most collections represent their data as one or more STAC items, each with its own metadata and assets. Portolan adds nothing to the STAC 1.1 Item beyond core: an item MUST be a valid STAC item, carrying an `id`, a `geometry` and `bbox` (or null geometry for non-spatial data), a `datetime` or a `start_datetime`/`end_datetime` interval, its assets, and the structural links defined under Links. The `datetime` clause restates STAC 1.1 core validity and is enforced by structural validation; the Portolan-level guidance to carry an explicit `datetime` lives under Temporal Metadata as a SHOULD. Item-level detail that other extensions already cover — file, raster, vector — MUST use those extensions rather than being re-specified here.

Items are not always required. A single-file collection has none: its data is a collection-level asset (see Single-File Collections). A partitioned collection MAY represent each partition as an item, but this is not required — see Partitioned Collections for when it is worthwhile. An ordinary collection with multiple distinct data files represents each as an item.

Assets

Every asset MUST include an `href`, the URI to the data, with relative or absolute both allowed for now. Where an asset `href` is absolute, it MUST use `https` rather than `s3`, since browsers cannot fetch `s3` URLs directly. An asset SHOULD also provide `s3` or other cloud-native URLs through the alternate extension, so tools that prefer direct bucket access can use them. Portolan additionally

requires a `type`, meaning the media type, and at least one role on every asset. STAC leaves both optional, but a validator and a browser cannot do their jobs without them, so Portolan makes them mandatory. The `title` and `description` fields are optional but recommended.

The required media type for each core format is:

Format	Media type
GeoParquet and plain Parquet	`application/vnd.apache.parquet`
COG	`image/tiff; application=geotiff; profile=cloud-optimized`
PMTiles	`application/vnd.pmtiles`
COPC	`application/vnd.laszip+copc`
Thumbnail	`image/png`, `image/jpeg`, or `image/webp`

Roles describe what each asset is for. Portolan uses the standard STAC role names wherever they fit and requires at least one per asset: `data` for the primary GeoParquet, COG, or Parquet; `visual` for the PMTiles rendering derivative; `thumbnail` for the preview image; `metadata` for sidecar metadata, with `iso-19115` used specifically for an ISO 19115 file. Multiple roles on one asset are fine — use all that apply. The `style` role, used for MapLibre style files, is a Portolan-defined role rather than a standard STAC one (see Visualization Styles).

The `source` role identifies an upstream original from which a cloud-native asset in this catalog was derived. For example, a publisher that converts an upstream Shapefile or GeoPackage to GeoParquet may include the original file as a `source` asset. PORTO-FMT-002 says that a mirror SHOULD include the original when it is directly downloadable rather than available only through an API. This means linking to the upstream file; the publisher does not need to retain, rehost, or redistribute it. A `source` asset is exempt from the format requirements because it represents the upstream original and may use a non-cloud-native format. A catalog that provides metadata for data that is already available upstream in a cloud-native format does not need a `source` asset; its `data` asset points directly to the upstream data. `collection-mirror` identifies the STAC-GeoParquet copy of a collection's items, as specified under Item mirror in formats.md.

Portolan reuses existing extensions rather than restating their fields: `file` for `file:values`, `table` for schema and columns, `raster` and `vector` for band and layer detail, `license` for per-asset license, and `scientific` for citation or DOI.

Assets SHOULD carry `file:size` and `file:checksum` from the file extension. Both are cheap to compute for bytes the publisher uploads, and they let a client budget a download and verify what it received. Neither is required, because a catalog that describes data it does not host cannot always produce them, and a fabricated value is worse than an absent one.

A `file:checksum` MUST use multihash encoding, not a raw sha256 string. Any `file:size` and `file:checksum` an asset carries MUST match the bytes its `href` resolves to. A stale value is a conformance failure, so a publisher that cannot keep one current should omit it.

Primary-vs-alternate. A non-cloud-native representation MAY be included as an alternate asset as long as an equivalent cloud-native primary asset exists (e.g. a GeoJSON alongside the required GeoParquet). The cloud-native asset is the primary; the rest are alternates.

Links

Every catalog, collection, and item MUST include the structural links that make the tree navigable: a catalog or collection MUST include `root` and `parent` links (except the root catalog, which has no parent) and a `child` or `item` link for every object it contains; an item MUST include `root`, `parent`, and `collection` links. Every structural link MUST carry a `type` of `application/json` (or `application/geo+json` for links to items).

Structural links can be relative or absolute; Portolan takes no position. Relative links keep a catalog portable, so the same tree of files validates locally, in staging, and in production without a rewrite. Absolute links name their location outright, which some hosts and clients handle more reliably. The STAC best practices on the use of links walk through the choice, and Git-Backed Catalogs covers how it plays out for a catalog maintained in git.

A catalog served over the internet from a single fixed URL SHOULD carry an absolute `self` link on its root catalog. The link records the canonical location, so someone holding a downloaded copy can see where it came from, and a tool can resolve the catalog's relative references into URLs from the metadata alone. STAC requires that a `self` link be absolute.

Every link in a catalog MUST resolve. A validator MUST resolve each relative link against the catalog's own file tree, whether on a local filesystem or on object storage, and confirm the referenced file is present and is the correct object, rather than merely checking that the `href` is present or well-formed. A link that fails to resolve, or resolves to the wrong object, is a conformance failure.

A validator resolves an absolute structural link through the base that the root `self` link names. It strips that base from the `href`. It then resolves the remainder against the file tree, as it does for a relative href.

A catalog that carries absolute structural links without a root `self` link gives a validator no base. The validator reports nothing for those links, because it cannot place them. An `href` that does not start with the base gets the same treatment. A root catalog that links a child in a different bucket therefore gets no check on that link.

The requirement to resolve still holds. A publisher who wants the check keeps a root `self` link.

Provenance (`via`) links are covered under Source Provenance.

Alternate-Language Trees

A catalog MAY publish one STAC object tree for each language. The root catalog links to each translated root with an `alternate` link, as defined by the STAC Language extension.

A translated tree represents the same catalog in another language. It is not part of the containment tree. A catalog MUST NOT link to a translated root with `child` or `item`.

Each language tree has its own root catalog. That root has no `parent` link. Every `root` link in the tree points to that root catalog. Equivalent objects keep the same ID across language trees.

A validator MUST treat a translated root as the start of a separate language tree. It MUST NOT report missing containment links or duplicate IDs between language trees.

Publishers choose which languages and objects to translate. Multilingual Catalogs gives maintenance guidance.

Human-Readable Titles

STAC Browser and other clients render `child` and `item` link titles directly; without them a client must fetch every child just to display its name. Portolan therefore requires human-readable titles throughout: every `catalog.json` and `collection.json` MUST have a non-empty `title` and `description`; titles MUST be human-readable, so a raw slug (`snake_case`) or a technical namespace prefix (`ns:LayerName`) is not acceptable; and every `child` and `item` link MUST include a `title`. A validator checks readability heuristically; it MUST flag a title that fails the check, and because heuristics misfire, the finding is a warning, not an error.

Bounding Boxes and Spatial Extent

Bounding boxes carry the spatial footprint that drives extent unions and map-UI browsing, and garbage coordinates poison the catalog-level extent and break viewers. So every `bbox` — catalog extent, collection extent, and item — MUST:

- contain no `NaN` or infinite values (including 3D elevation coordinates);
- contain no sentinel "effectively infinite" values (e.g. `±1.79e308`);
- use only WGS84 coordinates in range (longitude −180 to 180, latitude -90 to 90) with south ≤ north; and
- in a 3D bbox, order the vertical axis with minimum elevation ≤ maximum elevation.

STAC requires `extent.spatial.bbox` for collections; for tabular (non-geospatial) collections it represents the area of interest the data pertains to, not a geometric footprint (see Tabular).

Temporal Metadata

Items SHOULD carry an explicit `datetime` (or a `start_datetime` / `end_datetime` interval) describing when the data applies.

Data Storage

Cloud-native formats depend on clients fetching only the bytes they need, so Portolan catalogs assume data is hosted in cloud object storage reachable over HTTP range requests (S3-compatible or otherwise).

The requirements in this section apply to servers hosting the catalog's own cloud-native assets: the copies of the data that the publisher has uploaded and controls. They do not apply to upstream sources hosted by third parties.

A server hosting the catalog's cloud-native assets MUST support range requests: honor the `Range` header, return `206 Partial Content`, and advertise `Accept-Ranges: bytes`; HEAD requests MUST return an accurate `Content-Length`. The server MUST support HTTP/1.1 or greater; HTTP/2 or /3 is RECOMMENDED. Endpoints that compress or transform responses in ways that break range semantics are not conformant.

To let browser clients read data directly, a server hosting the catalog's cloud-native assets MUST also enable CORS on all metadata and asset files: `Access-Control-Allow-Origin: *` (or an equivalent read-permitting policy), allowed methods including `GET` and `HEAD`, allowed request headers including `Range`, `If-Match`, `If-Modified-Since`, `If-None-Match`, and `If-Unmodified-Since` (via `Access-Control-Allow-Headers`), and exposed response headers including `Content-Type`, `Content-Length`, `Content-Range`, `Accept-Ranges`, and `ETag` (via `Access-Control-Expose-Headers`).

The conditional-request headers let a browser revalidate a cached range instead of refetching it, which keeps tile and range readers cheap on repeat views. Provider-specific headers such as `x-amz-*` and `x-goog-*` fall outside this requirement, since reading a public catalog takes no signed request. Portolan leaves cloud-specific configuration to separate guidance.

Catalogs also reference bytes that the publisher does not host, including upstream sources that a mirror complements and original files carried over from an upstream source. Because those servers are operated by third parties and are outside the publisher's control, the requirements above do not apply to them.

A validator MUST probe the servers hosting the catalog's own cloud-native assets and MUST NOT require upstream servers to satisfy these requirements. An upstream server that does not support range requests or provide the required CORS headers limits the capabilities available to clients using that upstream copy, but does not make the catalog non-conformant.

Providers

Every collection MUST identify the parties responsible for its data through the STAC `providers` field. The list MUST include at least one provider with the `producer` role, the organization that originally captured or created the data, and exactly one provider with the `host` role, listed as the last element. The host is the organization responsible for operating and maintaining this copy of the catalog and its data, not the underlying cloud vendor whose storage it happens to sit on; a catalog on S3 maintained by a city GIS office lists the office as host, not AWS. Providers with the `processor` and `licensor` roles SHOULD be included where they apply. A single organization MAY hold multiple roles; a self-published dataset will typically list one provider as both producer and host. Catalogs MAY also declare providers, but the collection-level declaration is authoritative.

The host provider MUST include contact information: a `url` pointing to a page where the maintainer can be reached, or an `email` field with a maintainer address. `email` is not a core STAC field but is a valid extension of the provider object, and the Portolan schema enforces that at least one of the two is present. This is the maintainer-contact requirement for every Portolan catalog; a catalog whose host provides neither is non-conformant. A collection MAY additionally use the contacts extension for richer contact information, but doing so does not replace the url-or-email requirement on the host provider.

Source Provenance

Often a Portolan catalog is a mirror that complements an official source with cloud-native formats rather than being the source itself, and users need to trace back to the original. Every Portolan catalog is therefore one of two kinds:

- **Official** — published by the originating authority for the data. The catalog is a canonical home for the data, not a copy of one.
- **Mirror** — a cloud-native copy that complements a source maintained elsewhere. The mirror exists to add open formats, better access, or discoverability; the authoritative copy lives at the source.

Which kind a catalog is, is derived from its providers, not declared through any dedicated property. A catalog is official when its producer and host are the same organization; it is a mirror when they differ. An organization hosting data it did not produce is a mirror under this definition even when it is the primary distributor; the `via` link then simply records where the data originated.

A mirror MUST include a `via` link (type `text/html`) pointing to the original source. When that source publishes its own STAC catalog, the mirror MUST also include a `canonical` link pointing to the source's STAC, so consumers and agents can follow the chain to the authoritative metadata, not just a landing page. Whether the source publishes a STAC catalog cannot be determined from the mirror's metadata alone, so a validator MAY surface a mirror without a `canonical` link as informational, never as a failure. The `via` and `canonical` links MAY both be present and point at different targets (a human page and a STAC root). An official catalog carries no `via` or `canonical` link to an upstream source, because it is the source.

A mirror MUST record when it was last synced from its source by setting the top-level `updated` field (RFC 3339) at each synced catalog and collection to the time of the sync. A sync is a metadata update, so this reuses the core STAC field with no Portolan-specific addition; a consumer judges the freshness of a copy by comparing `updated` against the source. Sync cadence and richer source-freshness metadata are deferred to a future version.

License

Every collection MUST declare a `license` in its `collection.json`. The value MUST be an SPDX license identifier (e.g. `CC-BY-4.0`, `Apache-2.0`), or the STAC value `other` when no SPDX identifier fits, in which case the collection MUST include a license link (`rel: license`) pointing to the license text. The deprecated STAC 1.1 value `proprietary` MUST NOT be used. A collection whose data is genuinely restricted still declares its license explicitly rather than omitting it.

AGENTS.md

Every catalog and collection MUST include an `AGENTS.md` in Markdown, referenced in the STAC `links` array (`rel: "agents"`, `type: text/markdown`). It is a link, not an asset — it describes the data, it is not the data. Beyond existing and being linked, its content is open-ended; the aim is to help agents use the data well, covering access patterns (base URLs, S3 paths, code examples), useful aggregations and queries, data-quality notes, related collections, and schema or coordinate-system conventions — anything non-obvious. See best-practices for guidance on what makes a good `AGENTS.md`.

README.md

Every catalog and collection MUST have a `README.md` in Markdown, referenced in the STAC `links` array (`rel: "describedby"`, `type: text/markdown`). Like `AGENTS.md` it is a link, not an asset — it describes the data, it is not the data. `describedby` is the IANA-registered relation for a resource carrying a description of the linked resource, and is already common in STAC. The `README.md` MUST contain at minimum a title, description, license, and data provenance.

Catalog Logo

A registry lists many catalogs side by side. A logo makes each one recognizable before the reader has read a word. The root catalog MAY publish a logo as a link with `rel: "icon"`.

STAC defines no `logo` relation. `icon` is registered with IANA, and stac-js and STAC Browser already read it. STAC uses `preview` for a preview of the data itself, so `preview` is not the relation for a logo. A collection publishes its data preview as a `thumbnail` asset, described under Visualization.

An `icon` link MUST declare a `type`, and that type MUST be one of `image/apng`, `image/avif`, `image/gif`, `image/jpeg`, `image/png`, `image/svg+xml`, or `image/webp`. A client drops an icon whose media type

it does not recognize, so an undeclared type renders nowhere. stac-js rejects `image/svg+xml`. STAC Browser therefore does not show an SVG logo.

An `icon` link SHOULD carry a `title`. A page that renders the logo needs the title as the image's accessible label. Only the publisher knows what the logo shows.

The `href` SHOULD be relative, which keeps the catalog portable for the reason Links gives. A catalog stores the image under `_assets/` beside the root `catalog.json`. An absolute `href` to an external host stays valid, and a publisher who serves one logo across several catalogs will use one.

Metadata

Collections SHOULD include machine-readable metadata (e.g. Apache Ossie) when they have many coded or categorical variables or complex classification schemes, and SHOULD include column descriptions, which will likely become a requirement as tooling matures.

Visualization

Every geospatial collection MUST provide a way to render its data with zero infrastructure, either by:

- **rendering from source** — the data asset is small and simple enough for clients to draw directly (e.g. a small GeoParquet, or a COG that is already display-ready, whether through an embedded color table or as a continuous raster colorized at draw time via the render extension and the required min/max statistics); or
- **a visualization derivative** — an additional cloud-native product optimized for rendering, published as a collection-level asset with `roles: ["visual"]`; PMTiles is the recommended vector format today.

A consumer decides by inspection: use a `visual` asset if present, otherwise render from source. Non-geospatial collections are exempt. Whether an asset is small and simple enough to render from source is not decidable from metadata, so a validator cannot enforce this requirement directly; it MAY advise, as informational, when a large vector collection lacks a `visual` derivative. Collections MUST include a thumbnail generated from default styling and MUST provide visualization styles as standalone STAC assets appropriate to the render path, except where the render path is self-rendering (a display-ready COG or a small GeoParquet drawn directly), which needs no separate style file.

Visualization Styles

When a collection provides a render path it MUST provide at least one style telling clients how to draw it, in a format appropriate to that render path. Style files are STAC assets: each style MUST be registered as a collection-level asset carrying the `style` role, alongside the data and thumbnail. A client or agent discovers a collection's styles by filtering assets on that role, so no separate manifest is needed and this specification defines none. Because STAC assets are an unordered JSON object, and because STAC states that asset keys carry no meaning a client is expected to understand, the default style is identified by a second role rather than by position or key: when a collection provides more than one style, exactly one style asset MUST carry both `style` and `default` in its `roles`. STAC encourages multiple roles per asset, so this needs no extension. Asset keys stay free-form — `style-<variant>` keys such as `style-labeled` are the convention — and every style carries a human-readable name in its `title`. Where a collection provides exactly one style, that

style is the default and needs no marker, though carrying the `default` role there too is encouraged so a client can find the default the same way everywhere.

The concrete style format is defined per data format in `formats.md`: for vector (PMTiles) it is a MapLibre GL style file, while raster styling is still under discussion (see `specs/incubating/raster-styling.md`).

Portolan Specification — Formats

This document defines the format-specific requirements. The format-agnostic requirements are in `core.md`.

STAC allows assets in any format, but Portolan requires that every collection and item is available in a cloud-optimized format, and adds requirements on specific ones: in general GeoParquet and PMTiles for vector, COG for raster, COPC for point clouds, and Parquet for non-spatial data. A mirror SHOULD also include the original data as an asset when it is directly downloadable (not just an API). Catalogs SHOULD reference alternate formats of data and metadata — for example an existing ISO 19115 file referenced as an asset with the `metadata` role — though new catalogs need not pre-produce them, since tooling should make translation easy.

The requirements in this document apply to the catalog's own cloud-native assets: copies that the publisher derives and publishes. They do not apply to `source` assets, which represent the upstream originals from which those copies were derived and may use a non-cloud-native format.

A validator MUST apply the format requirements in this document to the catalog's own cloud-native assets and MUST NOT fail an asset for its format because it carries the `source` role. The `source` role identifies provenance, not hosting. The Data Storage requirements in core.md apply according to who hosts the bytes, as specified there, regardless of the roles an asset carries.

Vector

Vector has no single complete cloud-native format yet, so Portolan pairs two strong ones — GeoParquet and PMTiles — in a single STAC entity; the community is upgrading GeoParquet toward a more complete answer while watching formats like Iceberg, and this pairing will likely relax once GeoParquet gains overviews and more browsers render it directly.

GeoParquet

Data MUST be provided in GeoParquet 1.1 or 2.0, following the Best Practices for Distributing GeoParquet so it can be queried without a server. Files SHOULD be compressed to stay small, with `zstd` RECOMMENDED.

Rows MUST be spatially ordered so nearby features are nearby in the file. This rule applies to every file, no matter how many row groups it has. A validator checks it by looking at the rows themselves. It splits them into the groups a conforming writer would have produced, then checks that each of those groups covers a small part of the file.

A file with five or more row groups gets a second check, using the row groups it actually has. A reader can run this one from the file footer alone, without reading any data. The file passes if either of these is true:

- **low overlap** — fewer than 30% of consecutive row-group pairs have bounding boxes that overlap on their interiors; or

- **high locality** — row-group bounding boxes cover, on average, less than about 30% of the file's total extent.

Boxes that small let a reader skip about half the row groups when querying a window covering 10% of the extent. That is the benefit the 30% figure is meant to deliver, not a separate test to run.

The 30% figure comes from measuring Hilbert-sorted data, which is how producers usually sort. With five row groups, Hilbert-sorted boxes cover about 27% of the extent, and that number drops as row groups are added. Five row groups divide the extent five ways, so each box covers about a fifth of it before any overlap is counted. A stricter limit would fail well-sorted files for having few row groups.

Neither of these two checks applies to a file with fewer than five row groups. Both measure a percentage across the row groups, and with only a few groups the percentage cannot land on a useful value. Three row groups can only produce an overlap of 0%, 50%, or 100%. Two or three boxes cannot average less than 30% of the extent, however well the rows are sorted. A validator MUST NOT fail a file for missing a threshold that its row-group count puts out of reach. Row ordering is a separate rule and is not waived here, so a file with fewer than five row groups is still checked on its rows.

Files MUST provide per-row-group spatial statistics so readers can skip row groups from metadata alone — either:

- a GeoParquet 1.1 `bbox` covering column with Parquet min/max statistics on its leaf fields (a `bbox` column without statistics does not qualify); or
- for GeoParquet 2.x / Parquet `GEOMETRY`, native `GeospatialStatistics` per geometry column chunk.

For 2.x, a covering column remains RECOMMENDED even where native statistics exist, since it adds page-level min/max stats that enable finer-grained pruning.

Row groups MUST hold no more than 150,000 rows.

A vector collection SHOULD document its columns with the STAC table extension, carrying `table:columns` with names, types, and descriptions on the collection itself. An item that carries a GeoParquet data asset, as when partition files are modeled as items, SHOULD carry the same field in its `properties`. Those are the two places the extension scopes the field to, and the collection is the only one available to a partitioned collection, whose data sits behind `partition:glob` rather than in an asset. A collection whose data assets describe differing schemas MAY declare `table:columns` per asset instead.

Without it the attribute names and types live only in the Parquet footer, so a client has to read the file to learn what the collection holds. Names and types can be generated from the Parquet schema. The descriptions are written once and belong in both the metadata and the README schema table.

PMTiles

A PMTiles file SHOULD be provided as the visualization derivative, web-map-optimized and range-request friendly. It MUST be registered through a collection-level `rel: "pmtiles"` link per the web-map-links extension (v1.3.0): type `application/vnd.pmtiles`, a `pmtiles:layers` array of default-visible layers, with the extension's v1.3.0 schema declared in `stac_extensions`. The `pmtiles:layers` array MUST be non-empty; an empty list leaves a client with nothing to display. Because PMTiles

exists for visualization and partial reads rather than download, it is expressed as a link by default; when a provider intends the PMTiles file as a genuine distribution format of the data, it MAY additionally be registered as a collection-level asset, and the link and the asset then coexist.

When PMTiles are provided, the collection MUST include at least one visualization style as a standalone STAC asset with the `["style"]` role, registered per Visualization Styles. For PMTiles the style is a MapLibre GL style file (MapLibre GL style spec v8) in a `styles/` subdirectory, with media type `application/vnd.mapbox.style+json`, a complete, self-contained JSON loadable directly by MapLibre GL JS. By convention such a file sets `version` 8, a human-readable `name`, `sources.data.url` as the relative path from `styles/` to the PMTiles file (typically `../filename.pmtiles`), and `layers[].source` to `"data"`.

Partitioned Collections

Large files MAY be partitioned. Partitioning MUST be described per the partition extension (v1.0.0), with its schema declared in `stac_extensions` and its required fields carried — `partition:scheme`, `partition:keys`, and `partition:glob`. Field definitions live in the extension and are not restated here. Portolan adds the requirements the extension does not cover:

- The scheme's path structure MUST reflect spatial extent so readers can prune files without reading metadata.
- `partition:glob` is the normative bulk-access path; the collection description SHOULD also mention the glob for human readers, but validators read only the field. The https-only rule for absolute asset hrefs does not extend to the glob: globs are consumed by partition-aware readers rather than browsers, and bucket-native schemes (`s3://`, `gs://`) MAY be used where those readers need them (glob expansion requires listing, which plain https does not provide).
- Every partition file MUST share a single Parquet schema — the same columns, names, and types — so the glob can be queried as one table. This is validated by tooling reading file footers, not by JSON schema.

Partition files MAY be represented as items when partitions are user-meaningful units (countries, regions); for opaque schemes (hilbert, s2, h3) or hundreds of partitions, items SHOULD NOT be created — the glob pattern is the access path.

As a rough guide, consider partitioning files over ~2 GB, targeting 200 MB–1 GB per file — fewer, larger files outperform many small ones for DuckDB. This is not prescriptive; tune to your data and access patterns.

Raster

Raster data MUST be provided as Cloud Optimized GeoTIFF (COG) for efficient range-request access without full download. A COG here means a valid COG per the OGC Cloud Optimized GeoTIFF standard (OGC 21-026): an internally tiled GeoTIFF carrying georeferencing keys, with a header ordered so a reader can find the data it needs in an early range request. This is the baseline that `rio cogeo validate` and rasterio treat as a COG. (Formats such as GeoZarr are candidates for future support once default tooling can render and consume them.) How a raster collection is structured — one item per scene, with a single-COG collection handled as a single-file collection — is defined under Raster Collections.

Optimized GeoTIFF conformance. Beyond that baseline, a COG MUST conform to OGC 21-026's Optimized GeoTIFF requirements class (`/req/optimized_geotiff`), which adds three requirements:

- Small tiles (`/req/optimized_geotiff/small-sizes`): square internal tiles, sized no larger than a common screen viewport. 512×512 is the usual choice.
- Reduced-resolution subfiles number (`/req/optimized_geotiff/number`): internal overviews, each reducing resolution by a factor between 2 and 10, extending until the coarsest level spans one tile across or down.
- GeoTIFF keys (`/req/optimized_geotiff/geotiff`): georeferencing on the full-resolution IFD.

The overview requirement is the one that bites in practice. A raster larger than a single internal tile needs internal overviews so a reader can display it zoomed out without fetching full-resolution pixels; one that already fits within a tile is exempt, since it is its own overview. Base COG validators, including `rio cogeo validate`, treat missing overviews as a warning rather than a failure. Portolan raises this to a requirement.

Raster statistics. COGs MUST carry pixel statistics for rendering. Every band MUST carry an embedded minimum, maximum, mean, and standard deviation so a renderer can scale any data type without reading pixels. Statistics MUST be embedded in the file — an external `.aux.xml` (PAM) sidecar does not satisfy this — and MUST be written at creation time (e.g. `gdal_translate -of COG -stats`), residing in the file's leading header block so they arrive in a reader's first range request.

The required statistics are:

Statistic	Requirement
Minimum, maximum, mean, standard deviation	**MUST**
Valid percent	SHOULD (MUST when the band has a nodata value)
Approximate flag (`STATISTICS_APPROXIMATE = YES`)	MUST when estimated

The exact on-disk encoding (the TIFF `GDAL_METADATA` tag and its XML layout, and `GDAL_NODATA`) is specified in `specs/incubating/geotiff-stats-headers.md`. Compliance is defined by the tag contents, not by use of GDAL.

Item mirror. A raster collection that models scenes as items SHOULD also publish a stac-geoparquet item mirror at `items.parquet` in the collection root. One range request then returns the whole collection's item metadata, in place of one HTTP fetch per scene. Clients can search a large scene collection, or assemble it into a data cube, without a STAC API server.

An **item mirror** is a Parquet copy of a collection's items. This is distinct from Portolan's provenance meaning of *mirror*, where a mirror is a catalog republishing data it did not produce. See Source Provenance.

Always refer to this construct as an **item mirror** or **STAC-GeoParquet mirror**, never simply **mirror**.

No item-count threshold applies. Tooling generates the item mirror directly from the item JSON, and avoiding repeated JSON fetches provides a benefit even for small collections.

A published item mirror MUST be registered as a collection-level asset carrying media type `application/vnd.apache.parquet` and the role `collection-mirror`, per Referencing STAC Geopar-

quet Collections in STAC Collection JSON. That single registration is the whole requirement; no `rel: "items"` link is needed.

The item JSON remains the normative representation. An item mirror is a derived Parquet copy and MUST exactly reproduce the collection's items at publication time: one row per item, with every row containing that item's fields.

An item mirror that falls out of sync with its source items produces incorrect query results, and clients have no reliable way to detect the mismatch.

The GeoParquet requirements above apply to item mirrors exactly as they apply to vector datasets. Rows MUST be spatially ordered, the file MUST include per-row-group spatial statistics, and row groups MUST contain no more than 150,000 rows. Clients query an item mirror spatially just as they query any other GeoParquet dataset.

A collection containing only a single COG has no items and therefore publishes no item mirror. Item mirrors for other collection types, and STAC-GeoParquet mirrors covering an entire catalog's collections, remain incubating. See `specs/incubating/stac-geoparquet.md`.

Raster styling (colormaps, legends, continuous vs. categorical vs. multiband) is still under discussion — see `specs/incubating/raster-styling.md`.

Tabular (Non-Geospatial)

Portolan supports non-geospatial tabular data as companion data to a catalog's spatial layers — tables keyed by time, administrative code, or category rather than by location (e.g. census demographics by tract ID, permit records by parcel number, budget allocations by administrative unit, or time-series such as sensor readings). Tabular support is scoped to data that relates to the same geographic area as the catalog's spatial layers; Portolan is not a general-purpose data catalog.

A tabular collection MUST be distinguishable from a geospatial one so that validators and federation agents relax spatial requirements and route queries correctly. No explicit marker property is defined; a tabular collection is identified by its data, a Parquet asset with no geometry column. This MAY be revisited in a future version if implicit detection proves insufficient.

Data MUST be provided as a Parquet file (`application/vnd.apache.parquet`) exposed as a collection-level asset with role `["data"]`, following the single-file collection pattern — no item directory or item JSON. Where a source file is converted (e.g. a CSV), the original MAY be retained as an alternate asset under the primary-vs-alternate rule, with the Parquet as the primary.

Spatial requirements are relaxed. STAC requires `extent.spatial.bbox` on every collection, so a tabular collection MUST still carry one, but it represents the area of interest the data pertains to, not a geometry footprint, and validators MUST treat it as informational. GeoParquet spatial metadata, PMTiles and other visualization derivatives, and geometry validation do not apply, as there is no geometry.

Tabular collections SHOULD populate `extent.temporal` when the data has a time dimension, and SHOULD document their columns with the STAC table extension (`table:columns` with names, types, and descriptions). Because tabular data has no geometry to signal its meaning, the column schema is the primary semantic handle for consumers and agents.

When geometry and attributes live in separate files (e.g. census geometries joined to a demographics table), the metadata MUST document the join columns explicitly in the README and MUST include a working code example showing how to join them. Richer multi-file relationship modeling (via the table extension or Apache Ossie) is still being worked out.

All other Portolan core requirements apply unchanged: `collection.json`, `README.md`, `AGENTS.md`, providers, and `via` provenance links.

```
eurostat-electricity-prices/
├── collection.json
├── AGENTS.md
├── README.md
├── electricity-prices.parquet
└── electricity-prices.csv   (source, if converted)
```

Point Cloud

Point cloud support is not yet implemented; when complete, data MUST be provided as Cloud-Optimized Point Cloud (COPC). Full requirements will follow a reference implementation — see `specs/incubating/point-cloud.md`.

Requirements

PORTO-CORE-001

Severity: **MUST** · Enforcement: **validator** · Section: Core Structure

> It MUST be a valid STAC catalog following STAC 1.1.0, with a root `catalog.json` always required, even for a single collection.

PORTO-CORE-002

Severity: **MUST** · Enforcement: **validator** · Section: Core Structure

> catalogs and collections are internal nodes, items are leaves, and assets (data files) MUST sit at the collection or item level.

PORTO-CORE-003

Severity: **MAY** · Enforcement: **validator** · Section: Core Structure

> Catalogs organize only and MAY nest into sub-catalogs;

PORTO-CORE-004

Severity: **MUST** · Enforcement: **validator** · Section: Core Structure

> collections MUST be one level deep, containing only items or assets, never nested collections.

PORTO-CORE-005

Severity: **MUST** · Enforcement: **validator** · Section: Core Structure

Every catalog and sub-catalog MUST contain a `catalog.json`, an `AGENTS.md`, and a `README.md`; every collection MUST contain a `collection.json`, an `AGENTS.md`, and a `README.md`.

PORTO-CORE-006

Severity: **MUST** · Enforcement: **validator** · Section: Conformance and Versioning

Every catalog and collection MUST declare the versioned Portolan schema URI (e.g. `https://schemas.portolan-sdi.org/portolan/v0.2.0/schema.json`) in its `stac_extensions` array.

PORTO-CORE-007

Severity: **MUST** · Enforcement: **validator** · Section: Conformance and Versioning

The specification version MUST NOT be conflated with the dataset version.

PORTO-CORE-008

Severity: **MUST** · Enforcement: **validator** · Section: Conformance and Versioning

Dataset versioning, where used, MUST use the STAC version extension.

PORTO-CORE-009

Severity: **SHOULD** · Enforcement: **validator** · Section: Conformance and Versioning

All objects within a catalog SHOULD declare the same Portolan schema URI.

PORTO-CORE-010

Severity: **MUST** · Enforcement: **validator** · Section: Conformance and Versioning

A validator MUST flag any catalog or collection whose declared URI differs from the root catalog's; such a mismatch is a warning, not an error, and a mixed-version catalog remains valid.

PORTO-CORE-011

Severity: **MUST** · Enforcement: **process** · Section: Conformance and Versioning

The metadata passes MUST be executable without reading asset data; the data pass MAY be run independently.

PORTO-CORE-012

Severity: **MAY** · Enforcement: **process** · Section: Conformance and Versioning

`versions.json` is a tooling artifact and is NOT REQUIRED in a catalog for v0.1.

PORTO-CORE-013

Severity: **MUST** · Enforcement: **validator** · Section: Catalogs

The canonical entrypoint is a STAC `catalog.json` in the catalog root, which MUST be valid.

PORTO-CORE-014

Severity: **MUST** · Enforcement: **validator** · Section: Catalogs

Catalogs MAY use nested sub-catalogs for organization but MUST NOT use nested collections.

PORTO-CORE-015

Severity: **MUST** · Enforcement: **validator** · Section: Collections

A collection directory MUST contain a `collection.json` and holds one subdirectory per item.

PORTO-CORE-016

Severity: **SHOULD** · Enforcement: **validator** · Section: Collections

Collection IDs SHOULD contain only lowercase letters, numbers, hyphens, and underscores, start with a letter, and be unique within the catalog.

PORTO-CORE-017

Severity: **MUST** · Enforcement: **validator** · Section: Single-File Collections

When a collection holds a single data file (e.g. one GeoParquet file), that data MUST be represented as a collection-level asset — no item directory or item JSON is needed (see Vector).

PORTO-CORE-071

Severity: **MUST** · Enforcement: **validator** · Section: Raster Collections

A collection holding multiple raster scenes MUST model each scene as an item carrying its COG as an item-level asset; scene COGs MUST NOT be listed as collection-level assets.

PORTO-CORE-072

Severity: **MUST** · Enforcement: **validator** · Section: Raster Collections

A collection holding a single COG follows the single-file rule above: it MUST expose that COG as a collection-level asset with no item directory (see Raster).

PORTO-CORE-018

Severity: **MUST** · Enforcement: **validator** · Section: Nested Catalogs, Flat Collections

A collection MUST NOT contain a child collection.

PORTO-CORE-078

Severity: **SHOULD** · Enforcement: **validator** · Section: Nested Catalogs, Flat Collections

A catalog or collection with twenty or more children SHOULD organize them into subcatalogs, thematic or otherwise, so users can browse the data.

PORTO-CORE-019

Severity: **MUST** · Enforcement: **validator** · Section: Items

an item MUST be a valid STAC item, carrying an `id`, a `geometry` and `bbox` (or null geometry for non-spatial data), a `datetime` or a `start_datetime`/`end_datetime` interval, its assets, and the structural links defined under Links.

PORTO-CORE-020

Severity: **MUST** · Enforcement: **process** · Section: Items

Item-level detail that other extensions already cover — file, raster, vector — MUST use those extensions rather than being re-specified here.

PORTO-CORE-021

Severity: **MAY** · Enforcement: **validator** · Section: Items

A partitioned collection MAY represent each partition as an item, but this is not required — see Partitioned Collections for when it is worthwhile.

PORTO-CORE-022

Severity: **MUST** · Enforcement: **validator** · Section: Assets

Every asset MUST include an `href`, the URI to the data, with relative or absolute both allowed for now.

PORTO-CORE-023

Severity: **MUST** · Enforcement: **validator** · Section: Assets

Where an asset `href` is absolute, it MUST use `https` rather than `s3`, since browsers cannot fetch `s3` URLs directly.

PORTO-CORE-024

Severity: **SHOULD** · Enforcement: **process** · Section: Assets

An asset SHOULD also provide `s3` or other cloud-native URLs through the alternate extension, so tools that prefer direct bucket access can use them.

PORTO-CORE-025

Severity: **MUST** · Enforcement: **validator** · Section: Assets

Portolan additionally requires a `type`, meaning the media type, and at least one role on every asset.

PORTO-CORE-026

Severity: **MUST** · Enforcement: **validator** · Section: Assets

The required media type for each core format is: | Format | Media type | |———-|———————| | GeoParquet and plain Parquet | `application/vnd.apache.parquet` | | COG | `image/tiff; application=geotiff; profile=cloud-optimized` | | PMTiles | `application/vnd.pmtiles` | | COPC | `application/vnd.laszip+copc` | | Thumbnail | `image/png`, `image/jpeg`, or `image/webp` |

PORTO-CORE-027

Severity: **MUST** · Enforcement: **validator** · Section: Assets

Portolan uses the standard STAC role names wherever they fit and requires at least one per asset: `data` for the primary GeoParquet, COG, or Parquet; `visual` for the PMTiles rendering derivative; `thumbnail` for the preview image; `metadata` for sidecar metadata, with `iso-19115` used specifically for an ISO 19115 file.

PORTO-CORE-028

Severity: **SHOULD** · Enforcement: **validator** · Section: Assets

Assets SHOULD carry `file:size` and `file:checksum` from the file extension.

PORTO-CORE-029

Severity: **MUST** · Enforcement: **validator** · Section: Assets

A `file:checksum` MUST use multihash encoding, not a raw sha256 string.

PORTO-CORE-030

Severity: **MUST** · Enforcement: **validator** · Section: Assets

Any `file:size` and `file:checksum` an asset carries MUST match the bytes its `href` resolves to.

PORTO-CORE-031

Severity: **MAY** · Enforcement: **validator** · Section: Assets

A non-cloud-native representation MAY be included as an alternate asset as long as an equivalent cloud-native primary asset exists (e.g. a GeoJSON alongside the required GeoParquet).

PORTO-CORE-032

Severity: **MUST** · Enforcement: **validator** · Section: Links

Every catalog, collection, and item MUST include the structural links that make the tree navigable: a catalog or collection MUST include `root` and `parent` links (except the root catalog, which has no parent) and a `child` or `item` link for every object it contains; an item MUST include `root`, `parent`, and `collection` links.

PORTO-CORE-033

Severity: **MUST** · Enforcement: **validator** · Section: Links

Every structural link MUST carry a `type` of `application/json` (or `application/geo+json` for links to items).

PORTO-CORE-081

Severity: **SHOULD** · Enforcement: **process** · Section: Links

A catalog served over the internet from a single fixed URL SHOULD carry an absolute `self` link on its root catalog.

PORTO-CORE-035

Severity: **MUST** · Enforcement: **validator** · Section: Links

Every link in a catalog MUST resolve.

PORTO-CORE-036

Severity: **MUST** · Enforcement: **validator** · Section: Links

A validator MUST resolve each relative link against the catalog's own file tree, whether on a local filesystem or on object storage, and confirm the referenced file is present and is the correct object, rather than merely checking that the `href` is present or well-formed. A link that fails to resolve, or resolves to the wrong object, is a conformance failure.

PORTO-CORE-079

Severity: **MAY** · Enforcement: **validator** · Section: Alternate-Language Trees

A catalog MAY publish one STAC object tree for each language. The root catalog links to each translated root with an `alternate` link, as defined by the STAC Language extension.

PORTO-CORE-080

Severity: **MUST** · Enforcement: **validator** · Section: Alternate-Language Trees

A translated tree represents the same catalog in another language. It is not part of the containment tree. A catalog MUST NOT link to a translated root with `child` or `item`. Each language tree has its own root catalog. That root has no `parent` link. Every `root` link in the tree points to that root catalog. Equivalent objects keep the same ID across language trees. A validator MUST treat a translated root as the start of a separate language tree. It MUST NOT report missing containment links or duplicate IDs between language trees.

PORTO-CORE-037

Severity: **MUST** · Enforcement: **validator** · Section: Human-Readable Titles

every `catalog.json` and `collection.json` MUST have a non-empty `title` and `description`;

PORTO-CORE-038

Severity: **MUST** · Enforcement: **validator** · Section: Human-Readable Titles

titles MUST be human-readable, so a raw slug (`snake_case`) or a technical namespace prefix (`ns:LayerName`) is not acceptable;

PORTO-CORE-039

Severity: **MUST** · Enforcement: **validator** · Section: Human-Readable Titles

and every `child` and `item` link MUST include a `title`.

PORTO-CORE-040

Severity: **MUST** · Enforcement: **validator** · Section: Human-Readable Titles

A validator checks readability heuristically; it MUST flag a title that fails the check, and because heuristics misfire, the finding is a warning, not an error.

PORTO-CORE-041

Severity: **MUST** · Enforcement: **validator** · Section: Bounding Boxes and Spatial Extent

So every `bbox` — catalog extent, collection extent, and item — MUST: - contain no `NaN` or infinite values (including 3D elevation coordinates); - contain no sentinel “effectively infinite” values (e.g. `±1.79e308`); - use only WGS84 coordinates in range (longitude -180 to 180, latitude –90 to 90) with south ≤ north; and - in a 3D bbox, order the vertical axis with minimum elevation ≤ maximum elevation.

PORTO-CORE-042

Severity: **SHOULD** · Enforcement: **validator** · Section: Temporal Metadata

Items SHOULD carry an explicit `datetime` (or a `start_datetime` / `end_datetime` interval) describing when the data applies.

PORTO-CORE-043

Severity: **MUST** · Enforcement: **validator** · Section: Data Storage

A server hosting the catalog's cloud-native assets MUST support range requests: honor the `Range` header, return `206 Partial Content`, and advertise `Accept-Ranges: bytes`; HEAD requests MUST return an accurate `Content-Length`.

PORTO-CORE-044

Severity: **MUST** · Enforcement: **process** · Section: Data Storage

The server MUST support HTTP/1.1 or greater; HTTP/2 or /3 is RECOMMENDED.

PORTO-CORE-045

Severity: **MUST** · Enforcement: **validator** · Section: Data Storage

To let browser clients read data directly, a server hosting the catalog's cloud-native assets MUST also enable CORS on all metadata and asset files: `Access-Control-Allow-Origin: *` (or an equivalent read-permitting policy), allowed methods including `GET` and `HEAD`, allowed request headers including `Range`, `If-Match`, `If-Modified-Since`, `If-None-Match`, and `If-Unmodified-Since` (via `Access-Control-Allow-Headers`), and exposed response headers including `Content-Type`, `Content-Length`, `Content-Range`, `Accept-Ranges`, and `ETag` (via `Access-Control-Expose-Headers`).

PORTO-CORE-073

Severity: **MUST** · Enforcement: **validator** · Section: Data Storage

A validator MUST probe the servers hosting the catalog's own cloud-native assets and MUST NOT require upstream servers to satisfy these requirements.

PORTO-CORE-046

Severity: **MUST** · Enforcement: **validator** · Section: Providers

Every collection MUST identify the parties responsible for its data through the STAC `providers` field.

PORTO-CORE-047

Severity: **MUST** · Enforcement: **validator** · Section: Providers

The list MUST include at least one provider with the `producer` role, the organization that originally captured or created the data, and exactly one provider with the `host` role, listed as the last element.

PORTO-CORE-048
Severity: **SHOULD** · Enforcement: **process** · Section: Providers

Providers with the `processor` and `licensor` roles SHOULD be included where they apply.

PORTO-CORE-049
Severity: **MAY** · Enforcement: **validator** · Section: Providers

A single organization MAY hold multiple roles; a self-published dataset will typically list one provider as both producer and host.

PORTO-CORE-050
Severity: **MAY** · Enforcement: **validator** · Section: Providers

Catalogs MAY also declare providers, but the collection-level declaration is authoritative.

PORTO-CORE-051
Severity: **MUST** · Enforcement: **validator** · Section: Providers

The host provider MUST include contact information: a `url` pointing to a page where the maintainer can be reached, or an `email` field with a maintainer address.

PORTO-CORE-052
Severity: **MAY** · Enforcement: **validator** · Section: Providers

A collection MAY additionally use the contacts extension for richer contact information, but doing so does not replace the url-or-email requirement on the host provider.

PORTO-CORE-053
Severity: **MUST** · Enforcement: **validator** · Section: Source Provenance

A mirror MUST include a `via` link (type `text/html`) pointing to the original source.

PORTO-CORE-054
Severity: **MUST** · Enforcement: **process** · Section: Source Provenance

When that source publishes its own STAC catalog, the mirror MUST also include a `canonical` link pointing to the source's STAC, so consumers and agents can follow the chain to the authoritative metadata, not just a landing page.

PORTO-CORE-055
Severity: **MAY** · Enforcement: **validator** · Section: Source Provenance

Whether the source publishes a STAC catalog cannot be determined from the mirror's metadata alone, so a validator MAY surface a mirror without a `canonical` link as informational, never as a failure.

PORTO-CORE-056

Severity: **MAY** · Enforcement: **validator** · Section: Source Provenance

The `via` and `canonical` links MAY both be present and point at different targets (a human page and a STAC root).

PORTO-CORE-057

Severity: **MUST** · Enforcement: **validator** · Section: Source Provenance

A mirror MUST record when it was last synced from its source by setting the top-level `updated` field (RFC 3339) at each synced catalog and collection to the time of the sync.

PORTO-CORE-058

Severity: **MUST** · Enforcement: **validator** · Section: License

Every collection MUST declare a `license` in its `collection.json`.

PORTO-CORE-059

Severity: **MUST** · Enforcement: **validator** · Section: License

The value MUST be an SPDX license identifier (e.g. `CC-BY-4.0`, `Apache-2.0`), or the STAC value `other` when no SPDX identifier fits, in which case the collection MUST include a license link (`rel: license`) pointing to the license text.

PORTO-CORE-060

Severity: **MUST** · Enforcement: **validator** · Section: License

The deprecated STAC 1.1 value `proprietary` MUST NOT be used.

PORTO-CORE-061

Severity: **MUST** · Enforcement: **validator** · Section: AGENTS.md

Every catalog and collection MUST include an `AGENTS.md` in Markdown, referenced in the STAC `links` array (`rel: "agents"`, `type: text/markdown`).

PORTO-CORE-062

Severity: **MUST** · Enforcement: **validator** · Section: README.md

Every catalog and collection MUST have a `README.md` in Markdown, referenced in the STAC `links` array (`rel: "describedby"`, `type: text/markdown`).

PORTO-CORE-063

Severity: **MUST** · Enforcement: **validator** · Section: README.md

The `README.md` MUST contain at minimum a title, description, license, and data provenance.

PORTO-CORE-074

Severity: **MAY** · Enforcement: **process** · Section: Catalog Logo

The root catalog MAY publish a logo as a link with `rel: "icon"`.

PORTO-CORE-075

Severity: **MUST** · Enforcement: **validator** · Section: Catalog Logo

An `icon` link MUST declare a `type`, and that type MUST be one of `image/apng`, `image/avif`, `image/gif`, `image/jpeg`, `image/png`, `image/svg+xml`, or `image/webp`.

PORTO-CORE-076

Severity: **SHOULD** · Enforcement: **validator** · Section: Catalog Logo

An `icon` link SHOULD carry a `title`.

PORTO-CORE-077

Severity: **SHOULD** · Enforcement: **validator** · Section: Catalog Logo

The `href` SHOULD be relative, which keeps the catalog portable for the reason Links gives.

PORTO-CORE-064

Severity: **SHOULD** · Enforcement: **process** · Section: Metadata

Collections SHOULD include machine-readable metadata (e.g. Apache Ossie) when they have many coded or categorical variables or complex classification schemes, and SHOULD include column descriptions, which will likely become a requirement as tooling matures.

PORTO-CORE-065

Severity: **MUST** · Enforcement: **process** · Section: Visualization

Every geospatial collection MUST provide a way to render its data with zero infrastructure, either by: - **rendering from source** — the data asset is small and simple enough for clients to draw directly (e.g. a small GeoParquet, or a COG that is already display-ready, whether through an embedded color table or as a continuous raster colorized at draw time via the

render extension and the required min/max statistics); or - **a visualization derivative** — an additional cloud-native product optimized for rendering, published as a collection-level asset with `roles: ["visual"]`; PMTiles is the recommended vector format today.

PORTO-CORE-066

Severity: **MAY** · Enforcement: **validator** · Section: Visualization

Whether an asset is small and simple enough to render from source is not decidable from metadata, so a validator cannot enforce this requirement directly; it MAY advise, as informational, when a large vector collection lacks a `visual` derivative.

PORTO-CORE-067

Severity: **MUST** · Enforcement: **validator** · Section: Visualization

Collections MUST include a thumbnail generated from default styling and MUST provide visualization styles as standalone STAC assets appropriate to the render path, except where the render path is self-rendering (a display-ready COG or a small GeoParquet drawn directly), which needs no separate style file.

PORTO-CORE-068

Severity: **MUST** · Enforcement: **validator** · Section: Visualization Styles

When a collection provides a render path it MUST provide at least one style telling clients how to draw it, in a format appropriate to that render path.

PORTO-CORE-069

Severity: **MUST** · Enforcement: **validator** · Section: Visualization Styles

Style files are STAC assets: each style MUST be registered as a collection-level asset carrying the `style` role, alongside the data and thumbnail.

PORTO-CORE-070

Severity: **MUST** · Enforcement: **validator** · Section: Visualization Styles

when a collection provides more than one style, exactly one style asset MUST carry both `style` and `default` in its `roles`.

PORTO-FMT-001

Severity: **MUST** · Enforcement: **validator** · Section: Portolan Specification — Formats

STAC allows assets in any format, but Portolan requires that every collection and item is available in a cloud-optimized format, and adds requirements on specific ones: in general

GeoParquet and PMTiles for vector, COG for raster, COPC for point clouds, and Parquet for non-spatial data.

PORTO-FMT-002

Severity: **SHOULD** · Enforcement: **process** · Section: Portolan Specification — Formats

A mirror SHOULD also include the original data as an asset when it is directly downloadable (not just an API).

PORTO-FMT-003

Severity: **SHOULD** · Enforcement: **process** · Section: Portolan Specification — Formats

Catalogs SHOULD reference alternate formats of data and metadata — for example an existing ISO 19115 file referenced as an asset with the `metadata` role — though new catalogs need not pre-produce them, since tooling should make translation easy.

PORTO-FMT-045

Severity: **MUST** · Enforcement: **validator** · Section: Portolan Specification — Formats

A validator MUST apply the format requirements in this document to the catalog's own cloud-native assets and MUST NOT fail an asset for its format because it carries the `source` role.

PORTO-FMT-004

Severity: **MUST** · Enforcement: **validator** · Section: GeoParquet

Data MUST be provided in GeoParquet 1.1 or 2.0, following the Best Practices for Distributing GeoParquet so it can be queried without a server.

PORTO-FMT-005

Severity: **SHOULD** · Enforcement: **process** · Section: GeoParquet

Files SHOULD be compressed to stay small, with `zstd` RECOMMENDED.

PORTO-FMT-006

Severity: **MUST** · Enforcement: **validator** · Section: GeoParquet

Rows MUST be spatially ordered so nearby features are nearby in the file. This rule applies to every file, no matter how many row groups it has. A validator checks it by looking at the rows themselves. It splits them into the groups a conforming writer would have produced, then checks that each of those groups covers a small part of the file. A file with five or more row groups gets a second check, using the row groups it actually has. A reader can run this one from the file footer alone, without reading any data. The file passes if either of these is true: - **low overlap** — fewer than 30% of consecutive row-group pairs have bounding boxes that

overlap on their interiors; or - **high locality** — row-group bounding boxes cover, on average, less than about 30% of the file's total extent.

PORTO-FMT-044

Severity: **MUST** · Enforcement: **validator** · Section: GeoParquet

A validator MUST NOT fail a file for missing a threshold that its row-group count puts out of reach. Row ordering is a separate rule and is not waived here, so a file with fewer than five row groups is still checked on its rows.

PORTO-FMT-007

Severity: **MUST** · Enforcement: **validator** · Section: GeoParquet

Files MUST provide per-row-group spatial statistics so readers can skip row groups from metadata alone — either: - a GeoParquet 1.1 `bbox` covering column with Parquet min/max statistics on its leaf fields (a `bbox` column without statistics does not qualify); or - for GeoParquet 2.x / Parquet `GEOMETRY`, native `GeospatialStatistics` per geometry column chunk.

PORTO-FMT-008

Severity: **SHOULD** · Enforcement: **validator** · Section: GeoParquet

For 2.x, a covering column remains RECOMMENDED even where native statistics exist, since it adds page-level min/max stats that enable finer-grained pruning.

PORTO-FMT-009

Severity: **MUST** · Enforcement: **validator** · Section: GeoParquet

Row groups MUST hold no more than 150,000 rows.

PORTO-FMT-010

Severity: **SHOULD** · Enforcement: **process** · Section: PMTiles

A PMTiles file SHOULD be provided as the visualization derivative, web-map-optimized and range-request friendly.

PORTO-FMT-011

Severity: **MUST** · Enforcement: **validator** · Section: PMTiles

It MUST be registered through a collection-level `rel: "pmtiles"` link per the web-map-links extension (v1.3.0): type `application/vnd.pmtiles`, a `pmtiles:layers` array of default-visible layers, with the extension's v1.3.0 schema declared in `stac_extensions`.

PORTO-FMT-012
Severity: **MUST** · Enforcement: **validator** · Section: PMTiles

The `pmtiles:layers` array MUST be non-empty; an empty list leaves a client with nothing to display.

PORTO-FMT-013
Severity: **MAY** · Enforcement: **validator** · Section: PMTiles

when a provider intends the PMTiles file as a genuine distribution format of the data, it MAY additionally be registered as a collection-level asset, and the link and the asset then coexist.

PORTO-FMT-014
Severity: **MUST** · Enforcement: **validator** · Section: PMTiles

When PMTiles are provided, the collection MUST include at least one visualization style as a standalone STAC asset with the `["style"]` role, registered per Visualization Styles.

PORTO-FMT-015
Severity: **MUST** · Enforcement: **validator** · Section: PMTiles

For PMTiles the style is a MapLibre GL style file (MapLibre GL style spec v8) in a `styles/` subdirectory, with media type `application/vnd.mapbox.style+json`, a complete, self-contained JSON loadable directly by MapLibre GL JS.

PORTO-FMT-016
Severity: **MAY** · Enforcement: **validator** · Section: Partitioned Collections

Large files MAY be partitioned.

PORTO-FMT-017
Severity: **MUST** · Enforcement: **validator** · Section: Partitioned Collections

Partitioning MUST be described per the partition extension (v1.0.0), with its schema declared in `stac_extensions` and its required fields carried — `partition:scheme`, `partition:keys`, and `partition:glob`.

PORTO-FMT-018
Severity: **MUST** · Enforcement: **process** · Section: Partitioned Collections

The scheme's path structure MUST reflect spatial extent so readers can prune files without reading metadata.

PORTO-FMT-019

Severity: **SHOULD** · Enforcement: **process** · Section: Partitioned Collections

`partition:glob` is the normative bulk-access path; the collection description SHOULD also mention the glob for human readers, but validators read only the field.

PORTO-FMT-020

Severity: **MAY** · Enforcement: **validator** · Section: Partitioned Collections

The https-only rule for absolute asset hrefs does not extend to the glob: globs are consumed by partition-aware readers rather than browsers, and bucket-native schemes (`s3://`, `gs://`) MAY be used where those readers need them (glob expansion requires listing, which plain https does not provide).

PORTO-FMT-021

Severity: **MUST** · Enforcement: **validator** · Section: Partitioned Collections

Every partition file MUST share a single Parquet schema — the same columns, names, and types — so the glob can be queried as one table. This is validated by tooling reading file footers, not by JSON schema.

PORTO-FMT-022

Severity: **SHOULD** · Enforcement: **process** · Section: Partitioned Collections

Partition files MAY be represented as items when partitions are user-meaningful units (countries, regions); for opaque schemes (hilbert, s2, h3) or hundreds of partitions, items SHOULD NOT be created — the glob pattern is the access path.

PORTO-FMT-023

Severity: **MUST** · Enforcement: **validator** · Section: Raster

Raster data MUST be provided as Cloud Optimized GeoTIFF (COG) for efficient range-request access without full download.

PORTO-FMT-024

Severity: **MUST** · Enforcement: **validator** · Section: Raster

Beyond that baseline, a COG MUST conform to OGC 21-026's Optimized GeoTIFF requirements class (`/req/optimized_geotiff`), which adds three requirements:

PORTO-FMT-025

Severity: **MUST** · Enforcement: **validator** · Section: Raster

A raster larger than a single internal tile needs internal overviews so a reader can display it zoomed out without fetching full-resolution pixels; one that already fits within a tile is exempt, since it is its own overview. Base COG validators, including `rio cogeo validate`, treat missing overviews as a warning rather than a failure. Portolan raises this to a requirement.

PORTO-FMT-026

Severity: **MUST** · Enforcement: **validator** · Section: Raster

COGs MUST carry pixel statistics for rendering.

PORTO-FMT-027

Severity: **MUST** · Enforcement: **validator** · Section: Raster

Every band MUST carry an embedded minimum, maximum, mean, and standard deviation so a renderer can scale any data type without reading pixels.

PORTO-FMT-028

Severity: **MUST** · Enforcement: **validator** · Section: Raster

Statistics MUST be embedded in the file — an external `.aux.xml` (PAM) sidecar does not satisfy this —

PORTO-FMT-029

Severity: **MUST** · Enforcement: **process** · Section: Raster

and MUST be written at creation time (e.g. `gdal_translate -of COG -stats`), residing in the file's leading header block so they arrive in a reader's first range request.

PORTO-FMT-030

Severity: **MUST** · Enforcement: **validator** · Section: Raster

Minimum, maximum, mean, standard deviation | **MUST** |

PORTO-FMT-031

Severity: **SHOULD** · Enforcement: **validator** · Section: Raster

Valid percent | SHOULD (MUST when the band has a nodata value) |

PORTO-FMT-032

Severity: **MUST** · Enforcement: **process** · Section: Raster

Approximate flag (`STATISTICS_APPROXIMATE = YES`) | MUST when estimated |

PORTO-FMT-033

Severity: **MUST** · Enforcement: **process** · Section: Tabular (Non-Geospatial)

A tabular collection MUST be distinguishable from a geospatial one so that validators and federation agents relax spatial requirements and route queries correctly.

PORTO-FMT-034

Severity: **MUST** · Enforcement: **validator** · Section: Tabular (Non-Geospatial)

Data MUST be provided as a Parquet file (`application/vnd.apache.parquet`) exposed as a collection-level asset with role `["data"]`, following the single-file collection pattern — no item directory or item JSON.

PORTO-FMT-035

Severity: **MAY** · Enforcement: **validator** · Section: Tabular (Non-Geospatial)

Where a source file is converted (e.g. a CSV), the original MAY be retained as an alternate asset under the primary-vs-alternate rule, with the Parquet as the primary.

PORTO-FMT-036

Severity: **MUST** · Enforcement: **validator** · Section: Tabular (Non-Geospatial)

STAC requires `extent.spatial.bbox` on every collection, so a tabular collection MUST still carry one, but it represents the area of interest the data pertains to, not a geometry footprint, and validators MUST treat it as informational.

PORTO-FMT-037

Severity: **SHOULD** · Enforcement: **validator** · Section: Tabular (Non-Geospatial)

Tabular collections SHOULD populate `extent.temporal` when the data has a time dimension, and SHOULD document their columns with the STAC table extension (`table:columns` with names, types, and descriptions).

PORTO-FMT-038

Severity: **MUST** · Enforcement: **process** · Section: Tabular (Non-Geospatial)

When geometry and attributes live in separate files (e.g. census geometries joined to a demographics table), the metadata MUST document the join columns explicitly in the README and MUST include a working code example showing how to join them.

PORTO-FMT-039

Severity: **MUST** · Enforcement: **process** · Section: Point Cloud

Point cloud support is not yet implemented; when complete, data MUST be provided as Cloud-Optimized Point Cloud (COPC).

PORTO-FMT-040

Severity: **SHOULD** · Enforcement: **validator** · Section: Raster

A raster collection that models scenes as items SHOULD also publish a stac-geoparquet item mirror at `items.parquet` in the collection root.

PORTO-FMT-041

Severity: **MUST** · Enforcement: **validator** · Section: Raster

A published item mirror MUST be registered as a collection-level asset carrying media type `application/vnd.apache.parquet` and the role `collection-mirror`

PORTO-FMT-042

Severity: **MUST** · Enforcement: **validator** · Section: Raster

An item mirror is a derived Parquet copy and MUST exactly reproduce the collection's items at publication time: one row per item, with every row containing that item's fields.

PORTO-FMT-043

Severity: **MUST** · Enforcement: **validator** · Section: Raster

The GeoParquet requirements above apply to item mirrors exactly as they apply to vector datasets. Rows MUST be spatially ordered, the file MUST include per-row-group spatial statistics, and row groups MUST contain no more than 150,000 rows.

PORTO-FMT-046

Severity: **SHOULD** · Enforcement: **validator** · Section: GeoParquet

A vector collection SHOULD document its columns with the STAC table extension, carrying `table:columns` with names, types, and descriptions on the collection itself.

PORTO-FMT-047

Severity: **SHOULD** · Enforcement: **validator** · Section: GeoParquet

An item that carries a GeoParquet data asset, as when partition files are modeled as items, SHOULD carry the same field in its `properties`.

PORTO-FMT-048

Severity: **MAY** · Enforcement: **validator** · Section: GeoParquet

A collection whose data assets describe differing schemas MAY declare `table:columns` per asset instead.

Best Practices

This folder is **guidance, not conformance**. Nothing here is a MUST/SHOULD/MAY requirement — those live in `specs/portolan/`. These are discussion pieces and recommendations on how to build good Portolan catalogs, aimed at both the people who publish them and the agents that consume them.

Contents

- `documentation.md` — what belongs in your `AGENTS.md` and what makes a good `README.md`.
- `philosophy.md` — how to think about scoping, organizing, and maintaining a Portolan catalog; pointers to exemplary catalogs.
- `git-backed-catalogs.md` — keeping catalog metadata in a repository, with the data outside it, and validating every change in CI.
- `multilingual-catalogs.md` — how to publish and maintain catalog metadata in several languages.
- `grader.md` — the rubric for rating catalog quality, A+ through F.
- `styling.md` — making visualization styles that are clear and distinctive across a catalog.
- `conversion-defaults.md` — the COG, partition, and thumbnail settings the Portolan CLI uses when converting source data.

Conversion Defaults

This is guidance, not conformance. What a Portolan catalog must contain lives in `specs/portolan/`; this page records the settings the Portolan CLI reaches for when it converts source data into cloud-native form. Most are borrowed from rio-cogeo and GDAL. A file produced with different settings still conforms as long as it meets the requirements in the spec, so treat these as sensible starting points rather than rules.

Raster (COG)

The Raster section requires a valid COG with internal overviews and embedded statistics. Within that envelope, the CLI picks these defaults.

- **Compression: DEFLATE.** Lossless and readable everywhere, a safe default across data types. `zstd` and `LZW` are fine alternatives where the reader supports them. Reach for lossy `JPEG` or `WEBP` only for visual imagery where some pixel loss is acceptable, never for measured or categorical data.
- **Predictor: horizontal for integer; test floating-point data.** Predictor `2` often shrinks integer rasters. Predictor `3` can help dense, continuous floating-point rasters, but it can increase the size of sparse float rasters. Test representative files before you set a predictor default. Skip the predictor for RGB byte imagery and already-compressed inputs, where it adds nothing.
- **Internal tiles: 512×512.** OGC 21-026 asks for square tiles no larger than a screen viewport and recommends a power of two: 256, 512, or 1024. 512 sits in the middle and is the tile size the overview requirement keys off, so a raster wider or taller than 512 pixels needs overviews.
- **Overview resampling: nearest for categorical, averaging for continuous.** `nearest` preserves exact class values and is the CLI default. For continuous imagery, `average` or `bilinear` produce smoother zoomed-out views. Choose by what the pixels mean.

Partitioning

When a file is large enough to partition, follow the sizing guidance already in the Partitioned Collections section: consider partitioning past roughly 2 GB, target 200 MB to 1 GB per file, and keep row groups under 150,000 rows. Fewer, larger files read faster in DuckDB than many small ones.

Thumbnails

Core requires a `thumbnail`-role asset on every Collection. The CLI renders it as a small PNG that reads as a real map: projected to Web Mercator at the data's true aspect ratio over a light basemap, rather than stretched into a square. A few hundred pixels on the long edge is plenty. The thumbnail is a preview to help someone recognize the data at a glance, not a substitute for opening it.

Best Practices — Documentation

Every catalog and collection carries two documentation files: an `AGENTS.md` for agents and a `README.md` for humans. Core requires both files and sets a content minimum for the README. This page covers what to put in them, drawn from the catalogs whose documentation agents use most successfully today.

The goal of all of it, borrowed from the Candid Core Framework (Trochim & Roy, 2025, doi:10.5281/zenodo.15227664), is to help users "swiftly, accurately, and confidently assess a dataset's practical value". Discovery is the solved half of the problem: a well-formed STAC catalog can be found and crawled. What documentation adds is evaluability. In the framework's words, effective metadata must transparently communicate "articulated data quality, explicitly acknowledged biases, candid limitations, recommended use cases informed by real-world insights, and collective community knowledge shared transparently."

Two files, two audiences

Write the README for a person deciding whether to trust and use the data. Write the AGENTS.md for an agent that has already committed to using it and now needs to get queries right on the first try. The files share a subject but not a job, so resist copying one into the other. A README schema table orients a reader; an AGENTS.md that names the join key saves an agent a failed query.

The STAC `description` field is a third surface, and it overlaps the README by design. They are two interfaces to the same facts: the description serves browsers, search results, and item lists; the README serves a person who has already clicked through. Duplication between them is fine. Drift is not, so generate both from one source where you can, and when you hand-edit one, update the other.

Evaluate the data, not just describe it

A schema table and an extent tell a user what the data is. The documentation that earns trust also tells them what it is *for*, and what it is not for. Future versions of this guidance may make these required sections; today they separate complete documentation from excellent documentation.

- **Suggested uses.** Name what the data has actually supported: the analysis in the source paper, the operational program it feeds, the question a known user answered with it. Attributed, real uses beat abstract claims of suitability. Keep them informative rather than restrictive; a list of proven uses invites reuse, a fence around "intended use" forecloses it.

- **Application limitations and inappropriate uses.** Say when *not* to use the data, in plain terms. The FTW global catalog does this well: a field polygon "is a *remote-sensing field unit*, **not** a cadastral/legal parcel. This is not a land-tenure product." One sentence like that prevents a whole category of misuse that no accuracy figure would catch.
- **Accuracy and uncertainty, quantified.** Report the validation numbers, and where confidence varies, ship it as data: a per-record confidence column or an uncertainty layer, with the derivation and a recommended threshold documented. An honest 85% with a map of where it is weak serves users better than an unqualified 95%.
- **Known biases.** State where the data is weaker, by region, season, or class, and why. FTW again: the confidence layer "is conservative outside the training distribution (e.g. smallholder systems)", with the suggested mitigation alongside. Naming a bias is what lets a user work around it.
- **Definitions.** When the data asserts a category — forest, urban, field, building — state the threshold and cite the definition used. Published forest datasets disagree by millions of square kilometres on definitional grounds alone. The category boundary is metadata, not a footnote.

Put these caveats where they will be seen. A warning buried in prose is easily missed once the data itself looks clean and continuous, so state limitations in the README *and* carry what you can in structured metadata: confidence as a column, quality masks as assets, definitions in column descriptions.

What belongs in AGENTS.md

The strongest agent guides published so far, such as the per-collection guides in Portolan NL, share a recognizable shape. In rough priority order:

- **What the data is, and how it connects.** Two or three sentences of context, then the identifiers that matter: which column is the stable key, and which codes join to other datasets. Portolan NL's administrative-areas guide points out that its CBS municipality codes are the join key used across all Dutch government data. That one sentence unlocks every cross-dataset query.
- **Runnable access patterns.** Show the exact DuckDB or GeoPandas invocation against the published URL, not a placeholder. For partitioned collections, include the glob pattern that reads every partition in one query. Note when remote reads stream over HTTP range requests, so agents query in place instead of downloading.
- **Quirks and caveats.** The highest-value section, because it holds what no schema can express. Real examples from Portolan NL: parcel ids are re-issued each year, so cross-year identity needs a spatial join; the RGB and infrared bands were flown in different seasons; black tile edges are unmasked nodata; one category exists in the data only from 2023 onward. Each sentence of this kind prevents a silently wrong answer.
- **The coordinate system, with consequences.** Name the CRS and say what follows from it. "EPSG:28992 is in metres, so `ST_Area` returns square metres. Divide by 10,000 for hectares" teaches more than the code alone.
- **Schema semantics.** Types and names live in `table:columns`; meaning lives here. Decode cryptic column names and code systems, state units, and when a collection ships several data files, document the schema of each file separately.

- **Query recipes worth stealing.** A handful of queries an analyst would run: an aggregation, a point-in-polygon lookup, a join into a related collection. Write recipes by running them, so they double as tests of the documentation.
- **Related collections and provenance.** Point at sibling collections that join well. Record how the data was produced, ideally as the exact command. Portolan NL's crop-parcels guide includes the full `ogr2ogr` invocation, flags and all, plus the geometry fix it needed.

Leave out what the machine metadata already says. Restating extents, licenses, and row counts from `collection.json` pads the file without helping. Agents read the JSON too, and duplicated facts drift.

What makes a good README.md

Core requires a title, description, license, and provenance. Treat that as a floor. The README is the landing page: for most users, human or agent, it is the first and often only thing read before deciding to use the data or move on. Write it as a narrative that carries a reader from "what is this" to "how do I get it", not as a checklist of fields.

- **Numbers build trust.** State feature counts, coverage, and the vintage of the data in the opening paragraph. "All 63,073 nationally listed monuments" tells a reader more than "monument locations".
- **Links carry the context.** Link the paper that describes the methods, the agency page the data comes from, the program that funded collection, and the datasets it derives from, in flowing sentences where each source is introduced, not as a bare reference list at the bottom. The Planet Venezuela earthquake catalog quotes its source program's own description and links every location name to the scenes it describes; a reader leaves its README knowing the event, the sensors, and the program, without having opened another tab first.
- **A schema table with described columns.** A column table with a description per field is the single most useful section for a human evaluating fit. Those descriptions belong in `table:columns` as well. Write them once in a manifest and generate both.
- **One runnable Quick Start.** The exact invocation that gets the data, near the top, against the published URL. One tested snippet that answers "this is how you get it" beats four aspirational ones.
- **Escalate to the interesting access pattern.** Past the Quick Start, skip the third and fourth basic query; show the pattern a reader would not have guessed. For a raster collection, that is treating the whole collection as one dataset: the Venezuela catalog turns its scene collection into a single xarray cube with `odc.stac.load`, and into one GDAL VRT from its STAC-GeoParquet index. For vectors, it is the glob query across every partition. One advanced, runnable pattern teaches what the catalog is capable of.
- **Catalog-level READMEs are tables of contents.** At the root and at each sub-catalog, a collections table (title, feature count, geometry, one-line description) plus a short section on where the data comes from gives a reader the whole shape of the catalog in one screen.
- **Cross-reference the agent guide.** A one-line callout near the top, pointing agents at the AGENTS.md, routes each audience to its file.

Keep the documentation honest

Drift between prose and metadata is the most common defect in otherwise strong catalogs: a README that lists a collection the catalog no longer contains, feature counts that disagree between pages, a Quick Start URL that returns 404 after a layout change, a guide claiming public domain while the metadata says otherwise. An agent guide is especially prone to surviving a refactor intact, still describing a structure the catalog no longer has, complete with example queries that filter on values that can no longer match. Three habits prevent it:

- Generate documentation from the STAC metadata on every publish, so tables and counts cannot diverge from the catalog they describe.
- Run every example before publishing. A broken Quick Start costs more trust than no Quick Start.
- When a description exists only in prose, propagate it into the metadata. Hand-written schema tables that never reach `table:columns` help humans and strand agents.

Best Practices — Git-Backed Catalogs

Why use git for catalog management?

Git gives publishers a controlled way to manage catalog metadata before and after publication. It provides change history, review, validation, and rollback without making Git part of the published catalog itself.

A catalog may already be public when someone discovers an error. With a git-backed workflow, the publisher can correct the metadata in a pull request, validate the change, and publish the new version. If a published change causes a problem, Git also preserves the previous version and the changes that led to it.

This workflow is useful when several people or agents maintain a catalog, when contributors need to propose corrections, or when the catalog contains generated metadata. It makes catalog maintenance more like software maintenance: changes are reviewed, checked, and recorded.

Nothing on this page is a requirement. Core defines what a catalog contains and how it declares conformance, but it does not define how publishers manage or publish catalogs. A catalog maintained another way can conform just as well.

Keep metadata in git, not data

The repository should contain the metadata needed to build and publish the catalog. Large data files should remain outside Git and be referenced by URL, because Git keeps old versions of committed files in its history.

Fields of the World separates repository files into three groups:

1. **Tracked and published:** STAC JSON, `README.md`, `AGENTS.md`, `llms.txt`, thumbnails, and logos.
2. **Tracked but not published:** build scripts, tests, publish configuration, and the repository's own README.
3. **Not tracked:** data files, credentials, and caches.

Committing large data files makes the repository expensive to clone and maintain. Git retains old binary versions even after files are deleted. The St. Louis catalog therefore ignores `catalog/**/*.parquet` and `catalog/**/*.pmtiles`.

A fresh clone may not contain the data needed by some checks. St. Louis uses `CI_LIGHT=1` in continuous integration to skip file-backed checks while keeping them available locally.

Separate published metadata from repository files

The repository should have a clear directory containing the files that it publishes. The three catalogs studied here use a directory such as `catalog/`, while build scripts, tests, and source material remain elsewhere in the repository.

This makes the publication boundary easy to inspect. It also lets tools such as STAC Browser open `catalog/catalog.json` directly from the repository to preview and inspect the catalog.

The publication directory does not need to contain the data assets referenced by the catalog. For example, a generated STAC-GeoParquet item index can live directly in object storage and be referenced by a collection.

A typical repository can use `catalog/` for published metadata, `staging/` or `sources/` for source material, `tools/` or `scripts/` for build code, and `tests/` for tests. A root `catalog.publish.yaml` can define the object-storage target and public base URL, keeping publication settings in one place.

Links and the publish step

Core takes no position on relative versus absolute links, and it recommends an absolute `self` link on the root of a catalog served from a single fixed URL. The STAC best practices on the use of links cover the general trade-offs. A git-backed workflow gets the best of both by keeping the tracked tree portable and letting the publish step add what only it knows.

Keep structural links relative in the repository. A git-backed catalog is authored in one place and served from another, and the same JSON has to be valid in the repository, in whatever preview a pull request builds, and in production. Relative links give all three for free: one tree of bytes, checked by CI on a laptop or a runner, published unchanged.

Leave the `self` link out of the tracked tree for the same reason. A tracked file whose correct content depends on where it is deployed produces diff noise, and it conflicts on merge or rebase for reasons unrelated to the change under review. Instead, have the publish step write the absolute `self` link onto the root catalog as it uploads. The tool doing the upload is the only one that knows the destination, and the publish configuration already holds the URL: a `catalog.publish.yaml` with a `public_base` key, as described above, has everything the step needs. The repository stays portable, and the published catalog records its canonical location.

Asset hrefs deserve their own decision. Absolute asset hrefs work better with some clients, because a client that reads one object in isolation has no base to resolve a relative href against. The Source Cooperative file listing is a concrete case: it renders asset hrefs as download links, and only absolute ones resolve. The publish step can rewrite asset hrefs to absolute from the same `public_base` it uses for the `self` link, or the tracked tree can carry absolute asset hrefs from the start when the data already has a stable home.

Keep the tools and inputs

The repository should contain the scripts, tools, and source inputs used to produce the catalog. This can be custom code or existing tools such as GDAL/OGR, `gpio`, or `portolan-cli`.

Sharing this pipeline makes the catalog easier to reproduce and maintain. It also gives other publishers concrete examples of different ways to build catalogs and gives tool maintainers real workflows to improve.

TriMet commits its `tools/` pipeline and source listings while ignoring source Shapefiles that can be fetched again.

Generated output should be deterministic. TriMet's `tests/test_regen.py` rebuilds the catalog and compares it with the committed version, which lets the repository enforce the rule that contributors edit the generator rather than generated output.

Generate large catalogs

Small catalogs can be edited by hand. Large item collections should be generated instead of maintained as thousands of individual files, because generated files make reviews, diffs, and clones unnecessarily expensive.

Fields of the World shows this distinction within one catalog. It commits its prediction collections but not its Sentinel-2 feature items, which contain about 22,700 tiles per year across two years, or about 45,000 item files. Its `scripts/features/README.md` explains why committing them is impractical, and `.gitignore` enforces the policy with `catalog/features/*/items/`.

The generated items are replaced by a STAC-GeoParquet item index in object storage. The committed `collection.json` points to `items.parquet`, allowing clients to read one index instead of thousands of individual item files.

Validate changes with continuous integration

Use rashid for Portolan conformance and stac-check for STAC validity and best practices. Both install from PyPI and run offline.

A minimal workflow is:

```
- run: python -m pip install stac-check rashid
- run: rashid check catalog/
```

Run these checks on pull requests and before publication. This catches errors before a change reaches the published catalog and gives agents a clear feedback loop when they edit metadata.

A root catalog with no collections can pass validation, which allows a repository to remain valid from its first commit:

```
$ rashid check catalog/ --schema
OK: 1 files checked, no findings.
```

Handle validator differences

`stac-check` recommends a `self` link, which the tracked tree omits on purpose because the publish step adds it. Treat `stac-check` best-practice recommendations as advisory and use rashid as the Portolan conformance gate, as Fields of the World does in `tests/test_stac_valid.py`.

Record accepted deviations

A catalog may have a reason to accept a validator finding temporarily, for example when it targets a spec change that has not shipped. TriMet records accepted deviations in `docs/conformance.md`, and its test fails when a finding is not listed there.

Link a catalog to its repository

If a catalog is maintained in Git, publish two links on the root catalog:

- `vcs` points to the repository.
- `issues` points to its issue tracker.

Use absolute URLs because the repository is outside the published catalog.

```
{
  "rel": "vcs",
  "href": "https://github.com/example/catalog",
  "title": "Source repository"
},
{
  "rel": "issues",
  "href": "https://github.com/example/catalog/issues",
  "title": "Issue tracker"
}
```

Fields of the World already publishes both links.

The STAC VCS Extension is also being developed to describe version-control information such as the VCS type, branch, commit, and tag. It is currently a proposal and is not part of Portolan. The extension can complement the repository link: `vcs` identifies the repository, while the extension can identify the particular version of the catalog that came from it.

The `issues` link serves a different purpose. It gives people and tools a direct place to report problems or propose corrections. STAC tooling could use this link to provide issue-reporting features directly from a catalog browser.

These links are a best practice, not a Core requirement. Not every catalog is maintained in Git, and a published catalog does not say whether a repository exists. A validator therefore cannot require them.

Include contribution guidance

The repository should explain how contributors can propose changes and report problems. A root `README.md` can point people to the relevant tools, documentation, issues, and pull requests.

The machine-readable `vcs` and `issues` links make the repository and issue tracker discoverable to software. The README and catalog description can provide additional context for human contributors.

Catalogs worth studying

- **ftw-data-catalog** — Metadata for a global machine-learning dataset with hundreds of gigabytes of data on Source Cooperative. Its `CLAUDE.md` documents the publication model and explains why a full recursive bucket listing was too slow.
- **portolan-catalog-stlouis** — Twenty collections mirrored from a city open-data portal, with the fetch, conversion, assembly, and validation workflow in the repository.
- **portolan-catalog-trimet** — Eight transit datasets, with deterministic regeneration tests and a documented conformance-waiver process.

Status

This page documents early practices in August 2026. These practices may change as more git-backed catalogs are published.

The `vcs` and `issues` links are a recommended convention, not a Portolan requirement. A future spec change could add a `SHOULD` for catalogs maintained in Git; that can be evaluated separately as the STAC VCS Extension and related link relations mature.

If you maintain a git-backed catalog and use a different approach, open an issue or pull request on the Portolan spec repository.

Best Practices — The Grader

Conformance is pass/fail: a catalog passes the validator or it does not. Above that floor, quality is a scale, and this rubric grades it from A+ to F. Compliance alone does not make data good — "you can have perfectly FAIR, utterly useless data" (Sundwall, 2025) — so the rubric measures two different things. Grades up through A rest on criteria checkable by inspection: a file resolves or 404s, a field is present or absent, a count matches or disagrees, so two reviewers reach the same grade. The A+ is a judgment call about whether the catalog is genuinely excellent to use, and it is assessed qualitatively.

One principle drives the weighting: **grade the machine contract as hard as the visible surface.** The two decouple in practice. A catalog can ship resolving thumbnails, thematic styles, and excellent prose while carrying empty provider lists and no agent links. Humans notice the first set. Agents and validators depend on the second. A rubric that rewards only what a browser shows will certify catalogs that fail every machine that reads them.

The ladder

The grades anchor to the spec, so a publisher always knows what a given level of effort earns:

- **F** — Not a validating STAC catalog. Malformed JSON, a missing or unreachable root, or structural links broken at scale. There is nothing here for a STAC client to consume.
- **D** — A real attempt, decently broken. The catalog validates as STAC but misses Portolan requirements in ways that block a class of consumers: a required file class absent throughout, no renderable assets, documentation that contradicts the metadata.
- **C** — Portolan-compliant, and nothing more. Every MUST in core is met: the required files exist and link, the schema URI is declared, primaries are cloud-native. Conformance is the entry ticket, not the destination.
- **B** — Compliant plus the SHOULDs, executed with minor defects. Thumbnails and styles resolve, agent guides exist at every level, providers and licenses are in order.
- **A** — Everything checkable, done well. The catalog sits at the top tier of every dimension below: dataset-specific agent guides, multiple purposeful visualizations, a default view that renders, prose that agrees with metadata everywhere.
- **A+** — Beyond anything the spec names. The judgment-call grade, described in its own section: candid evaluation of limitations and biases, a narrative that gives real context, an experience that renders instantly and reads beautifully, and contribution back to the shared network. When a user shows up at this catalog, it is an incredible experience.

How to read the tiers

Each dimension below anchors three tiers. **A** is the checkable ceiling. **B** is solid: complete with minor defects. **D** is broken in a way that blocks a class of consumers. Grades between the anchors interpolate: a C is a mix of B and D findings. A catalog's checkable grade is capped by these dimensions; the A+ sits above them all.

1. Required files and links

- **A** — `catalog.json`, `AGENTS.md`, and `README.md` exist and resolve at the root, every sub-catalog, and every collection. Each `AGENTS.md` is linked with `rel: "agents"` and each `README.md` with `rel: "describedby"`, both typed `text/markdown` and pointing at the raw file. Every catalog and collection declares the Portolan schema URI.
- **B** — All files exist and resolve, with scattered defects: a wrong media type on a `describedby` link, or a missing schema URI on some objects.
- **D** — A file class is absent across the catalog (every `AGENTS.md` 404s, or the links to them are missing), or the schema URI appears nowhere.

2. Metadata integrity

- **A** — Every asset carries the registered media type for its format and `file:size`. Every collection lists a producer and exactly one host provider, the host reachable by URL or email. Licenses are valid SPDX identifiers, and mirrors carry `updated`. Checksums (`file:checksum`) are credited when present, and they are cheap to generate, but their absence does not cost a grade.
- **B** — Providers, licenses, and media types are in order, but `file:size` is missing on some assets or `updated` has gone stale.
- **D** — Provider lists are empty (so the catalog has no owner and no contact), a forbidden license value like `proprietary` appears, or most data assets carry an unregistered media type.

3. Cloud-native assets

- **A** — The primary data asset of every collection is cloud-native (GeoParquet, COG, PMTiles, COPC) with the correct media type, a visual derivative exists wherever the source is not self-rendering, legacy formats appear only as clearly-roled alternates, and each file is registered exactly once.
- **B** — Cloud-native primaries and visuals throughout, with a mislabeled media type or a legacy file registered as a plain extra asset.
- **D** — Primaries typed `application/octet-stream` or in server-dependent formats, duplicate asset keys pointing at one file with conflicting roles, or a geospatial collection with no renderable asset at all.

4. Agent documentation

- **A** — Every `AGENTS.md` is dataset-specific: it names join keys, shows runnable remote queries against published URLs, states the CRS with its practical consequences, documents quirks the schema cannot express, offers non-trivial query recipes, and points at related collections.
- **B** — Present and linked at every level, but thin: access snippets and a schema restatement without quirks, recipes, or cross-references.
- **D** — Boilerplate repeated across collections, or content that contradicts the machine metadata it describes.

5. Visualization

- **A** — Every geospatial collection has a thumbnail that resolves and was rendered from its default style, plus at least one style asset per render path with the default marked by a `default` role. The catalog as a whole offers three or more visualizations that express genuinely different aspects of the data — a categorical theme, a confidence surface, a density view — not three renderings of the same picture. The only exception is data that is truly just a geometry with an attribute or two that do not visualize; per the styling guidance.
- **B** — A resolving thumbnail and a single default style on every collection, nothing more.
- **D** — Geospatial collections missing a thumbnail or a style entirely, or thumbnail links that 404.

6. Performance and first render

A catalog is judged at its front door: paste the root URL into a STAC browser, open a collection, and see whether data appears before the user gives up.

- **A** — The default view of every collection renders visible data immediately at its natural full extent. Tiled visual assets carry content from zoom 0 (or the collection's full-extent zoom): sub-pixel features get generalized low-zoom overviews — dissolved coverage, density fills, or representative points — rather than relying on the tiler to drop what does not fit. Visual assets live in the same repository as the catalog that references them.
- **B** — Everything renders, but slowly or partially: heavyweight thumbnails, low-zoom tiles that are technically present but visually near-empty, a first paint that takes long enough to doubt.
- **D** — The default view is blank. A tileset whose minimum zoom starts levels below the full extent, with no fallback layer or extent hint, shows a user nothing until they already know where to look — the worst possible first impression, and indistinguishable from a broken catalog.

7. Maintenance signals

- **A** — `updated` is set and recent on every synced object, mirrors carry `via` and (where upstream publishes STAC) `canonical` links to specific upstream endpoints, the host is contactable, and prose agrees with metadata everywhere: counts, licenses, and links say the same thing in the README, the AGENTS.md, and the JSON.
- **B** — Provenance links are specific and honest, but `updated` lags or appears only at some levels.
- **D** — `updated` is null across a mirror, no contact route exists, or the documentation contradicts the metadata: a README table listing collections that 404, counts that disagree between levels, a guide claiming one license while the JSON declares another.

The A+ judgment

Everything above can be verified by inspection, and an honest A is a catalog anyone can rely on. The A+ asks a different kind of question, drawn from the Candid Core Framework: does this catalog candidly outline known limitations, potential biases, and inappropriate use cases to promote responsible data application? Not "is the metadata complete", but is the dataset aware of its own blind spots — what does it preserve, what does it erase, and does it say so? The grader assesses this qualitatively, reading the catalog the way a prospective user would.

Marks of an A+ catalog:

- **Candid evaluation.** Suggested uses grounded in real applications; explicit inappropriate uses ("this is not a land-tenure product"); quantified accuracy with uncertainty shipped as data, not just a headline number; named biases with mitigations; definitions stated and cited.

- **Narrative and context.** The README reads as a story with the links woven in — the source paper, the producing agency, the program behind the data — so a user leaves understanding not just the schema but the world the data describes.
- **An experience, not just an artifact.** Renders instantly, styled with care, escalating access examples that are runnable and genuinely interesting, documentation scoped correctly at every level of the tree.
- **Network participation.** The catalog is registered on the Portolan registry. A team can use excellent data internally and earn an A; the A+ is reserved for catalogs contributing to the shared network.
- **Feedback loops.** The code that built the catalog is published, and a user who finds a mistake can file an issue or open a pull request against it. This practice is young and not yet expected, which is exactly why it distinguishes.

The overall grade

A catalog's checkable grade is its weakest dimension, raised by at most one step when most other dimensions sit at A. The floor rule is the point: excellence in visible dimensions cannot buy back a broken machine contract, because the consumers who depend on that contract never see the polish. The A+ is then awarded, or not, on top of a clean A.

Best Practices — Multilingual Catalogs

A multilingual catalog publishes a separate metadata tree for each language. The catalog uses one tree as its source and derives the other trees from it. `alternate` links connect equivalent catalogs and collections across the trees.

Use the STAC Language extension on each translated catalog and collection. It identifies the document's language and gives clients the information they need for a language selector.

The examples below use Romanian as the source language, with Russian and English translations. Core defines the required structure in Alternate-Language Trees.

The Basic Layout

Keep the source-language catalog at the root. Put each translation in a directory named with its RFC 5646 language tag:

```
national-geodata/
├── catalog.json              # Romanian source
├── README.md
├── AGENTS.md
├── cadastru/
│   ├── collection.json
│   ├── README.md
│   └── AGENTS.md
├── en/
│   ├── catalog.json
│   ├── README.md
│   ├── AGENTS.md             # may stay in the source language
│   └── cadastru/
│       ├── collection.json
│       ├── README.md
│       └── AGENTS.md
```

```
└── ru/
    └── ...
```

Use the same IDs and directory names in every tree. The file `en/cadastru/collection.json` is the English version of `cadastru/collection.json`, not a new collection with a new ID.

Only the metadata is duplicated. Data assets remain in the source tree, and translated metadata refers to them with relative hrefs.

Language Fields

Every translated catalog and collection declares the Language extension and its own language:

```
{
  "stac_extensions": [
    "https://schemas.portolan-sdi.org/portolan/v0.2.0/schema.json",
    "https://stac-extensions.github.io/language/v1.0.0/schema.json"
  ],
  "language": { "code": "ro", "name": "Română", "alternate": "Romanian" },
  "languages": [
    { "code": "ru", "name": "Русский", "alternate": "Russian" },
    { "code": "en", "name": "English" }
  ]
}
```

`language` describes the current document. `languages` lists the other available languages and must not repeat the current one.

Use an RFC 5646 tag for `code`. Write `name` in the language it names, such as `Română` or `Русский`. The optional `alternate` field gives a familiar name in another language. For a right-to-left language, add `"dir": "rtl"`.

Links Between Trees

Each root catalog links to every other root catalog:

```
{
  "rel": "alternate",
  "href": "./en/catalog.json",
  "type": "application/json",
  "title": "Engleză",
  "hreflang": "en"
}
```

The `type` matters because an `alternate` link can also point to HTML or another media type. The `hreflang` identifies the target language. The `title` names that language for the current reader, so a Romanian document uses `Engleză` and an English document uses `Romanian`.

Keep these hrefs relative. From the English root, the Romanian root is `../catalog.json` and the Russian root is `../ru/catalog.json`.

Add the same links to equivalent collections. This lets a reader switch languages without losing their place in the catalog.

Do not use `child` to link a translated root. The translation represents the same catalog; it is not a subcatalog.

Links to Untranslated Content

A language tree does not need to reproduce every item. A translated collection can link to an item in the source tree and identify its language with `hreflang`:

```
{
  "rel": "item",
  "href": "../../cadastru/parcele/parcele.json",
  "type": "application/geo+json",
  "title": "Cadastral parcels, 2024",
  "hreflang": "ro"
}
```

Translate the link title even when the target item remains in the source language. Clients can show the title without fetching the item.

Use `hreflang` on the `describedby` and `agents` links too. Both point at files in the same directory. The value names the language of the file it points at.

What to Translate

Translate the root catalog, the collections, and their `README.md` files. These carry the descriptions that people use to understand and find the data. The `AGENTS.md` files may stay in the source language, see Agent Guidance.

Translate the column descriptions under `table:columns` as well. A client shows them beside the column names, so they read as prose to the user. Keep the column names themselves unchanged.

Translate items only when their prose warrants it. Items often contain identifiers, geometries, times, and asset hrefs, so duplicating thousands of them may add little value.

Keep these values identical in every language:

- Collection IDs, item IDs, and asset keys
- Data column names
- Style layer IDs
- CRS identifiers
- License identifiers

They are identifiers or properties of shared data. Stable values also let clients match equivalent objects across trees.

Agent Guidance

Every catalog and collection MUST carry an `AGENTS.md`, and the `agents` link MUST point at the sibling file. Core states this under AGENTS.md. A translated node is a catalog or a collection, so it carries the file too. `rashid check` reports `PTL-FIL-001` and `PTL-FIL-002` when it does not.

The language of that file is a separate choice. Technical guidance describes the data rather than the prose, so it may stay in the source language. Mark the link with the language of the file, not the language of the tree:

```
{
  "rel": "agents",
  "href": "./AGENTS.md",
  "type": "text/markdown",
  "title": "Guidance for AI agents",
```

```
  "hreflang": "ro"
}
```

Translate `AGENTS.md` when the guidance itself is about the language. The term glossary below is one example, because a translating agent reads it.

One Source Language

Choose one source language and treat every other tree as derived. Use a language that the publisher can check against the upstream material. In this example, the publisher chooses Romanian because the upstream material is in Romanian.

Apply corrections to the source tree first, then regenerate the affected translations. A direct edit to a derived tree will disappear during the next regeneration.

Record the source language and the regeneration command in the root `AGENTS.md`.

A Glossary for Important Terms

General translation tools often choose reasonable but inconsistent words. A short glossary prevents that drift and protects distinctions that matter in the data. Keep it in the root `AGENTS.md`:

```
## Translation

Source language: Romanian (`ro`). Edit `catalog.json`, `collection.json`,
`README.md`, and `AGENTS.md` in the root tree. Regenerate `en/` and `ru/`.

Use these terms. Do not substitute a synonym.

| Concept          | ro                   | ru                   | en                |
| ---------------- | -------------------- | -------------------- | ----------------- |
| cadastral parcel | parcelă cadastrală   | кадастровый участок | cadastral parcel |
| land cover       | acoperirea terenului | земельный покров    | land cover       |
| geodetic network | rețea geodezică      | геодезическая сеть  | geodetic network |

Keep the agency names and abbreviations AGCC and INGEOCAD unchanged.
```

Include terms that affect search or meaning. For example, Spanish uses `riesgo` for risk and `peligro` for hazard. Those terms are related, but they are not interchangeable in flood metadata.

Keep Translations Current

Use the source-tree diff to decide what to regenerate. If one collection description changes, update that collection in each translated tree. There is no need to translate the whole catalog again.

CI can catch omissions. Fail the check when a source metadata file changes without a corresponding change in every translated tree.

Check the Result

Run `rashid check` on the catalog. It validates each language tree and the links between them.

Some warnings use English-language heuristics. `PTL-FIL-005` looks for license and provenance text in a collection `README.md`. `PTL-TTL-002` checks whether a title looks like prose. Review these warnings instead of assuming they apply equally to every language.

Finally, open the catalog in STAC Browser. Check that every language appears in the selector and opens the equivalent catalog or collection.

Best Practices — Catalog Philosophy

Core defines the shape of a catalog: the required structure and the nesting rules. This page covers the judgment calls the spec leaves open: what one catalog should contain, how to organize it, and what maintaining it actually involves.

Design for the person who shows up

A catalog is a data product, and the measure of a data product is what a user can do with it (Sundwall, 2025). Every choice below — scope, depth, naming, maintenance — serves one scenario: someone arrives at your catalog knowing nothing, and within minutes they understand what the data is, whether it fits their problem, and how to get it. The README is that first landing point; the structure is what makes the second minute as smooth as the first. Optimize for the visitor's experience, not the publisher's convenience, and most of the judgment calls resolve themselves.

Scope by steward, not by theme

The most durable catalog boundaries follow data stewardship. Portolan NL, a cloud-native mirror of Dutch national geodata, organizes 21 collections as one sub-catalog per producing institution: Kadaster, Rijkswaterstaat, CBS, and so on. Each sub-catalog inherits one upstream, one license posture, and one contact point from its institution, so provenance stays coherent without per-collection bookkeeping. Themes shift with fashion. Stewardship rarely moves.

Within a steward's catalog, give each published dataset one collection. When related datasets accumulate — boundaries at several administrative levels, a product family with shared provenance — group their collections under a sub-catalog rather than packing multiple datasets into one collection as sibling assets. A collection whose assets are really three datasets makes every consumer disambiguate them; a sub-catalog makes the grouping navigable.

Shallow and uniform beats deep and clever

Portolan NL keeps every path the same length: root, institution, collection. No collection sits deeper than another, so traversal is predictable and an agent that has walked one branch has learned the whole catalog. Thematic groupings ("water infrastructure", "boundaries") appear as tables in the README, not as directory levels. Prose reorganizes for free. Directories are forever. Add an intermediate catalog when a steward publishes enough collections that a flat list stops being readable, not because a taxonomy feels tidy.

Names outlive everything

Collection ids become URLs, join targets, and lines in other people's scripts, so choose them as contracts. Keep ids short, lowercase, and snake_case, and let the title do the describing: an id like `administrative_areas` under the title "Administrative Areas of the Netherlands (municipal, provincial, national)" gives scripts a stable handle while the title serves discovery and translation. Keep tooling artifacts out of ids. An export prefix that leaks from an ETL pipeline into a collection id is a temporary detail made permanent, and so is a pipeline stage name like `alpha/` baked into a published path.

Maintenance is the product

Publishing a catalog is the cheap part. What separates a catalog people build on from one they route around is everything that happens after:

- **Regenerate documentation from metadata on every publish.** Hand-edited pages drift: collection tables that link to renamed collections, feature counts that disagree between levels, examples that 404 after a layout change. Generated pages cannot.
- **Tend the invisible contract.** Providers, media types, the schema URI, and `updated` timestamps on mirrors are invisible in a browser and load-bearing for agents and validators. A catalog can look polished, with resolving thumbnails and rich prose, while failing every machine that reads it. Budget maintenance for both surfaces.
- **Record syncs.** A mirror that never updates `updated` gives consumers no way to judge freshness, which reads as abandonment even when the data is current.

Catalogs worth studying

Different data, different organizations, different concerns — browse them to see the patterns before reading about them:

- Portolan NL — an institution-scoped national mirror with exceptional per-collection agent guides and thematic style variants.
- Planet — Venezuela earthquake — an event-response imagery catalog: narrative README with the event story up front, escalating access recipes from a browser link to a whole-collection data cube, and pre-merged convenience mosaics so users need not juggle forty tiles.
- Fields of the World — Global — a global ML-derived dataset: candid caveats ("not a land-tenure product"), per-polygon confidence with the derivation documented, quality layers shipped as sibling data, and the build code linked from the catalog itself.
- The reference catalog in this repo — small, fully conformant, and regenerated end-to-end from manifests.

This list should grow toward a handful of exemplars across data types and publisher types. If you have built a catalog that teaches a pattern these do not, propose it.

This is a discussion

Nothing on this page is definitive. It is a working community's current best understanding of how to make great data products, extracted from a small number of real catalogs, and it will change as more people publish and report back. If your experience contradicts a recommendation here, or your catalog solved a problem this page does not cover, open an issue or a pull request on the spec repository — the guidance improves the same way the catalogs do.

Best Practices — Styling

Guidance for the visualization styles that core requires. These are recommendations, not requirements.

- **Create multiple data-driven styles for rich collections.** If a collection has interesting categorical or numeric attributes, offer a style for each, e.g. buildings by age, by use, or by height. Mark the default one by giving its asset a `default` role alongside `style`.
- **Vary default colors across a catalog** so it is not monotone. Use the subject matter as a guide: water in blues, vegetation in greens, the built environment in warm tones. Each collection should have a visually distinct default.
- **Use MapLibre GL expressions to reveal patterns.** Leverage `interpolate`, `match`, `case`, and `step` to encode data into color, and include a `description` explaining what the colors represent.

- **Consider labels.** For collections with named features (roads, monuments, admin areas), include a label layer or a dedicated “with labels” style variant.