Artifacts Folder Index · PYN / SID / CID Views

1. Purpose

1.1 The `Artifacts/` tree is a generated index of code artifacts. It does not create new identities. It projects what the registry already knows about PYN (entry-point identity), SID (section identity), CID (capability-bearing units), capability labels, environments, counts, and ordered sequences.
1.2 The registry is the single source of truth. The `Artifacts/` tree is a read-only filesystem view derived from registry data.
1.3 The index is exposed through three parallel and deterministic views: `PY/` (PYN-centric), `SID/` (SID-centric), and `CID/` (CID-centric).
1.4 Each view has a fixed depth and fixed decision points, which makes the structure suitable for automated generation and validation.

2. Top-Level Layout

2.1 The root layout of the Artifacts index is:

```
Artifacts/
  PY/
  SID/
  CID/
```

2.2 Each branch is an index into the same underlying artifacts, using a different primary key.
2.3 The `PY/` branch groups by PYN (entry-point identity).
2.4 The `SID/` branch groups by SID (section identity).
2.5 The `CID/` branch groups by CID (capability-bearing code unit).
2.6 This document defines the folder structure and naming conventions only. The exact files, symlinks, or manifests stored inside leaf folders are defined by the generator and may evolve, as long as the folder structure remains compliant with this specification.

3. PYN View (`Artifacts/PY`)

3.1 Structure

3.1.1 The PYN view uses exactly two decision levels beneath `PY/`.
3.1.2 The structure is:

```
Artifacts/
  PY/
    <env>/
      SID-count_<nnn>/
```

3.1.3 Level 1 is `PY`.

3.1.4 Level 2 is `<env>`, the execution environment.
3.1.5 Level 3 is `SID-count_<nnn>`, a bucket keyed by the number of SIDs associated with a PYN in that environment.
3.1.6 No additional folder nesting is defined by this specification under `SID-count_<nnn>/` for the PYN view.

### 3.2 Environment (`<env>`)

3.2.1 `<env>` identifies the execution environment associated with the PYN.
3.2.2 Valid environment names are defined by the registry and must match registry values exactly.
3.2.3 Example environment labels include `macos`, `ubuntu-22`, `ios`, `prod`, and `dev`. These are illustrative and not exhaustive.
3.2.4 Example environment folders:

```
Artifacts/PY/macos/
Artifacts/PY/ubuntu-22/
Artifacts/PY/prod/
```

### 3.3 SID Count (`SID-count_<nnn>`)

3.3.1 Within each environment, PYN entries are grouped by how many SIDs they have in that environment.
3.3.2 The folder name has the following pattern:

3.3.2.1 Prefix: `SID-count_`.
3.3.2.2 Suffix: a zero-padded integer representing the count of SIDs.
3.3.2.3 Examples: `SID-count_001`, `SID-count_010`, `SID-count_123`.

3.3.3 Example layout:

```
Artifacts/
  PY/
    macos/
      SID-count_001/
      SID-count_002/
      SID-count_010/

    ubuntu-22/
      SID-count_003/
```

3.3.4 This view supports queries such as “which PYNs in this environment have only one section” or “which PYNs have a large number of sections” based on the SID-count buckets.

## 4. SID View (`Artifacts/SID`)

4.1 Structure

4.1.1 The SID view uses three decision levels beneath `SID/`.
4.1.2 The structure is:

```
Artifacts/
  SID/
    <env>/
      CID-count_<nnn>/
        <cid-sequence-pattern>/
```

4.1.3 Level 1 is `SID`.
4.1.4 Level 2 is `<env>`, the execution environment.
4.1.5 Level 3 is `CID-count_<nnn>`, a bucket keyed by the number of CIDs associated with a SID in that environment.
4.1.6 Level 4 is `<cid-sequence-pattern>`, a bucket keyed by the exact ordered sequence of CIDs associated with that SID in that environment.
4.1.7 No additional folder nesting is defined by this specification beneath `<cid-sequence-pattern>/` in the SID view.

4.2 Environment (`<env>`)

4.2.1 The `<env>` label follows the same rules as in the PYN view.
4.2.2 Allowed environment values are defined in the registry and must match exactly.
4.2.3 Example layout:

```
Artifacts/SID/macos/
Artifacts/SID/ubuntu-22/
Artifacts/SID/prod/
```

4.3 CID Count (`CID-count_<nnn>`)

4.3.1 Within each environment, SIDs are first grouped by how many CIDs they contain in that environment.
4.3.2 The folder name has the following pattern:

4.3.2.1 Prefix: `CID-count_`.
4.3.2.2 Suffix: a zero-padded integer representing the count of CIDs.
4.3.2.3 Examples: `CID-count_001`, `CID-count_004`, `CID-count_120`.

4.3.3 Example layout:

```
Artifacts/
  SID/
    macos/
      CID-count_001/
      CID-count_003/
```

```
   CID-count_010/

  ubuntu-22/
   CID-count_002/
```

4.3.4 This level groups SIDs that share the same CID count but not necessarily the same CID identities or ordering. The next level refines this grouping.

4.4 CID Sequence Pattern (`<cid-sequence-pattern>`)

4.4.1 Inside each `CID-count_<nnn>/` folder, SIDs are further grouped by the exact ordered CID sequence.
4.4.2 SIDs placed in the same `<cid-sequence-pattern>/` folder must share all of the following properties:

4.4.2.1 The same number of CIDs (already enforced by `CID-count_<nnn>/`).
4.4.2.2 The same set of CID keys.
4.4.2.3 The same sequence of CIDs in the same order.

4.4.3 The name `<cid-sequence-pattern>` must be deterministically derived from the ordered list of CID keys.
4.4.4 A recommended encoding pattern is:

4.4.4.1 Prefix: `CID-seq_`.
4.4.4.2 Suffix: an underscore-separated list of CID keys in order.
4.4.4.3 Example: `CID-seq_cid_abcd1234_cid_bbbb2222_cid_cccc3333`.

4.4.5 Example layout:

```
Artifacts/
 SID/
  macos/
   CID-count_003/
    CID-seq_cid_abcd1234_cid_bbbb2222_cid_cccc3333/
    CID-seq_cid_x1111111_cid_y2222222_cid_z3333333/
```

4.4.6 All SIDs under `CID-seq_cid_abcd1234_cid_bbbb2222_cid_cccc3333/` share the same ordered CID list:

```
[cid_abcd1234, cid_bbbb2222, cid_cccc3333]
```

4.4.7 This view is intended to identify structurally identical sections across different files or repositories, where structure is defined by the ordered sequence of CIDs in a given environment.

5. CID View (`Artifacts/CID`)

### 5.1 Structure

5.1.1 The CID view focuses on CIDs and their capability roles.
5.1.2 The structure is:

```
    Artifacts/
      CID/
        <cid-key>/
          <cid-key>__cap_<capability>/
```

5.1.3 Level 1 is `CID`.
5.1.4 Level 2 is `<cid-key>`, the canonical CID identifier.
5.1.5 Level 3 is `<cid-key>__cap_<capability>`, the CID combined with a capability label.
5.1.6 No additional folder nesting is defined by this specification beneath `<cid-key>__cap_<capability>/`.

### 5.2 CID Key (`<cid-key>`)

5.2.1 The `<cid-key>` is the canonical CID string as stored in the registry.
5.2.2 A recommended pattern is `cid_<hash>`, where `<hash>` is a deterministic hash of the code section.
5.2.3 Examples of CID keys:

```
    cid_abcd1234
    cid_9876fedc
```

5.2.4 Example layout:

```
    Artifacts/
      CID/
        cid_abcd1234/
        cid_9876fedc/
```

### 5.3 CID plus Capability (`<cid-key>__cap_<capability>`)

5.3.1 Within each `<cid-key>/` folder, artifacts are grouped by capability label.
5.3.2 The folder name has the following pattern:

5.3.2.1 Prefix: `<cid-key>__cap_`.
5.3.2.2 Suffix: `<capability>`, a stable capability label.
5.3.2.3 The `<cid-key>` in the folder name must match the parent folder name exactly.

5.3.3 Examples:

```
    Artifacts/
      CID/
        cid_abcd1234/
          cid_abcd1234__cap_http-client/
          cid_abcd1234__cap_file-writer/

        cid_9876fedc/
          cid_9876fedc__cap_summarizer/
          cid_9876fedc__cap_log-enricher/
```

5.3.4 Capability labels describe what the code section is capable of doing, not necessarily what it did in a single execution.
5.3.5 Each `cid_*__cap_*` folder groups artifacts associated with a specific CID in a specific capability role.

6. Naming Conventions (Summary)

6.1 Environment Folders (`<env>`)

6.1.1 Environment labels are defined in the registry.
6.1.2 Folder names must match the registry's environment entries exactly.
6.1.3 Examples include `macos`, `ubuntu-22`, `prod`, and `dev`.

6.2 Count Folders

6.2.1 `SID-count_<nnn>` is used under `Artifacts/PY/<env>/` to group PYNs by SID count.
6.2.2 `CID-count_<nnn>` is used under `Artifacts/SID/<env>/` to group SIDs by CID count.
6.2.3 In both cases, `<nnn>` is a zero-padded integer.
6.2.4 Examples of valid names include `SID-count_001`, `SID-count_010`, `CID-count_001`, and `CID-count_120`.

6.3 CID Keys

6.3.1 CID keys should follow a consistent pattern such as `cid_<hash>`.
6.3.2 CID keys are used as folder names at `Artifacts/CID/<cid-key>/`.
6.3.3 CID keys must match registry entries exactly.

6.4 CID Sequence Patterns

6.4.1 CID sequence patterns are deterministic encodings of ordered CID lists.
6.4.2 A recommended encoding is `CID-seq_<cid1>_<cid2>_..._<cidN>`.
6.4.3 Each `<cid*>` entry in the name must be a valid CID key that matches the registry.
6.4.4 The sequence in the name must match the actual ordered sequence of CIDs under the SID in that environment.

6.5 CID plus Capability Names

6.5.1 CID plus capability folder names follow the pattern `<cid-key>__cap_<capability>`.
6.5.2 `<cid-key>` must match the parent folder name exactly.
6.5.3 `<capability>` must be a stable capability label defined in or synchronized with the registry.
6.5.4 Examples include `cid_abcd1234__cap_http-client`, `cid_abcd1234__cap_file-writer`, and `cid_9876fedc__cap_summarizer`.

7. Generator and Registry Relationship

7.1 Registry Responsibilities

7.1.1 The registry stores PYN, SID, and CID identities.
7.1.2 The registry stores environment labels and environment associations for PYNs and SIDs.
7.1.3 The registry stores capability labels and their associations with CIDs.
7.1.4 The registry stores relationship data, including:

7.1.4.1 The number of SIDs per PYN per environment.
7.1.4.2 The number of CIDs per SID per environment.
7.1.4.3 The ordered CID sequences per SID per environment.

7.2 Generator Responsibilities

7.2.1 The Artifacts generator reads identity and relationship data from the registry.
7.2.2 The generator creates and maintains the following projections:

7.2.2.1 `Artifacts/PY/<env>/SID-count_<nnn>/`.
7.2.2.2 `Artifacts/SID/<env>/CID-count_<nnn>/<cid-sequence-pattern>/`.
7.2.2.3 `Artifacts/CID/<cid-key>/<cid-key>__cap_<capability>/`.

7.2.3 The generator ensures that all folder names comply with the naming conventions defined in this document.
7.2.4 The generator may remove or rebuild `Artifacts/` content when regenerating the index to maintain consistency with the registry.

7.3 Usage and Constraints

7.3.1 Tools and humans may use the `Artifacts/` tree for navigation and inspection by PYN, SID, or CID.
7.3.2 The registry remains the only authoritative source of identity and relationships.
7.3.3 The `Artifacts/` tree must be treated as a projection.

7.3.4 Manual edits inside `Artifacts/` are unsupported and may be overwritten by the generator.
7.3.5 Any detected inconsistency between the `Artifacts/` tree and the registry should be resolved by correcting registry data and regenerating the index, not by manual changes to `Artifacts/`.