7.9 Container Naming Container names should identify the project, phase, node or service, and where useful, the runtime role. The naming convention should help developers inspect running containers, logs, networks, and images without having to guess which process belongs to which node. A representative container image naming pattern is: fsa-phase0-<node-or-service> Representative examples include: fsa-phase0-fx-validation-node fsa-phase0-control-plane-observer fsa-phase0-dds-tools Runtime container names may follow the same pattern with an environment or instance suffix if needed: fsa-phase0-fx-validation-node-1 fsa-phase0-control-plane-observer-1 Container names should not replace node names. The node name identifies the architectural runtime participant. The container name identifies the packaging and execution unit. In simple Phase 0 cases, one container may run one node, but the team should still preserve the distinction between logical node identity and container runtime identity. The team should keep container naming consistent across container definitions, build scripts, run scripts, logs, documentation, and acceptance evidence. A reviewer should be able to trace a container image or running container back to the node or support service it packages. 7.10 File Headers and Inline Documentation The team should consistently document handwritten artefacts. File headers and inline documentation help developers and reviewers understand why an artefact exists, how it participates in Phase 0, and whether it is handwritten, generated, or configuration-driven. Every handwritten source file and script should include a short header that identifies the artefact’s purpose, its role in the Phase 0 baseline, and any important usage constraints. Generated files should include a generated-file warning where the generator or file format permits it. Language-specific documentation conventions should follow the normal practice of the language or file type: Java source should use Javadoc for public classes and methods. Python source should use module, class, and function docstrings. JavaScript source should use JSDoc for public interfaces. TypeScript source should use TSDoc or JSDoc for public interfaces. Bash scripts should use a script header block and short comments before non-trivial functions. IDL files should use comment blocks to explain structures, fields, and topic-facing types. YAML, XML, and other configuration files should use header comments where the format permits comments. Markdown files should begin with a clear title and a short purpose statement. Documentation should explain purpose, assumptions, inputs, outputs, lifecycle effects, exceptions, exit codes, generated status, and configuration dependencies where those details matter. Documentation should not repeat obvious code. The goal is to make the implementation reviewable and maintainable, not to create noise. 7.11 Spelling, Grammar, and Style The team should write repository documentation, comments, file headers, script help text, configuration descriptions, generated documentation templates, catalogue entries, and acceptance evidence in clear, consistent English. Project documentation should use UK spelling unless a tool, programming language, external standard, product name, API, command, package, or quoted source requires another spelling. Examples include “artefact”, “behaviour”, “organisation”, “normalise”, “initialise”, “modelling”, and “authorisation”. The team should use active voice where practical. Documentation should identify the actor responsible for an action, such as “the script validates prerequisites” or “the node publishes status”, rather than relying on passive constructions such as “prerequisites are validated” or “status is published”. The team should use consistent terminology from the architecture documents and the handbook. Terms such as Node, Topic, Data Structure Definition, Data Structure Instance, Control Plane, Data Plane, generated artefact, baseline, and acceptance evidence should keep their defined meanings. Documentation should use complete sentences, correct grammar, and consistent punctuation. Headings should use consistent capitalisation. Lists should use parallel structure where practical. Short, direct sentences are preferable to long sentences with multiple embedded qualifications. The team should avoid the following in committed artefacts: Informal wording that weakens technical clarity. Unexplained abbreviations or local shorthand. Personal pronouns, such as “I”, “we”, “you”, “our”, or “your”, unless they appear in quoted material or an explicitly approved communication artefact. Passive voice where active voice would make the responsible actor clearer. Personal notes that refer to an individual developer rather than the project baseline. Temporary comments that do not link to an open issue, risk, or decision. Editing-history labels such as “new”, “old”, “final”, “final-final”, or “backup”.

Unresolved comments such as “fix later”, “temporary hack”, “not sure why this works”, or “come back to this”. Machine-specific or developer-specific references, such as local paths, usernames, workstation names, or “Nick’s version”. Such comments or labels should enter the controlled baseline only when they appear in an explicitly marked open issue, risk, decision record, or troubleshooting note. Spelling and grammar checks should form part of the review. Reviewers should correct minor issues when practical and request a revision when unclear writing affects traceability, reproducibility, reviewability, or evidence of acceptance.

From:
https://wiki.didosolutions.com/ - **Dido Solutions, Inc Wiki**

Permanent link:
https://wiki.didosolutions.com/fxdemo/05-part/07-naming-and-file-conventions/07-9-container-naming/start?rev=1786321570

Last update: **2026/08/09 17:26**