6.7 Tool Baseline Checklist

Go To Top

Return to Prerequisites and Tooling

The handbook defines a static tool Baseline checklist that the team uses to record and verify the tools required for a particular phase, build, release candidate, test run, demonstration Baseline, or deployment Baseline. The checklist structure should remain stable across phases so that the team can compare tool Baselines without rewriting the handbook for every implementation cycle.

The checklist should identify each tool category that may affect Repository access, source generation, build execution, Node execution, validation, review, acceptance Evidence, or deployment. The checklist should treat each tool category as a required capability, not necessarily as a single product selection. This allows the team to support one approved tool, several approved tools, or several interoperable implementations for the same capability.

Each checklist item should capture:

1. Tool category.
2. Approved product, platform, or implementation.
3. Vendor, project, or source.
4. Required version or acceptable version range.
5. Product status.
6. Required, recommended, optional, experimental, not applicable, or to-be-decided classification.
7. Applicable role or environment.
8. Phase, build, test, demonstration, or deployment purpose.
9. Installation or setup reference.
10. Verification command or validation method.
11. Observed version, where the checklist instance records a completed verification.
12. Verification result in which the checklist instance records a completed verification.
13. Evidence reference, where the checklist instance links to supporting logs, command output, screenshots, reports, or review notes.
14. Owner or responsible role.
15. Related implementation decision record, where applicable.
16. Exceptions, constraints, or open risks.

The checklist should use a consistent version-range convention. Version expressions should support exact versions, bounded ranges, open-ended ranges, exclusions, and not-applicable values.

Representative version expressions include:

1. Exact version: `21.0.2`
2. Minimum version: `>= 21`
3. Compatible range: `>= 21, < 22`
4. Open-ended range: `>= 3.12`
5. Excluded version: `>= 3.12, != 3.12.1, < 3.13`
6. Not applicable: `N/A`
7. To be decided: `TBD`

The team should prefer bounded ranges when tool behaviour, generated output, runtime

compatibility, or acceptance Evidence may change across major versions. The team may use open-ended ranges when the tool has stable backward compatibility or when the phase-specific checklist identifies the risk.

The static checklist should cover at least the following tool categories:

1. Repository platform.
2. Version control.
3. Shell environment.
4. Editor or IDE.
5. Java environment.
6. Python environment.
7. DDS vendor and toolset.
8. IDL compiler or type generator.
9. Container runtime.
10. Build tool.
11. Package manager.
12. Test and validation tools.
13. Documentation generation tools.
14. Diagram or model export tools, where applicable.
15. Security, signing, credential, or secret-management tools, where applicable.
16. Operating-system assumptions.
17. CI/CD or automation runner environment, where applicable.

The checklist should also distinguish applicable roles and environments. A tool may apply to developers, testers, technical reviewers, CI automation, demonstration operators, release managers, documentation maintainers, or deployment environments.

Representative role and environment categories include:

1. Developer workstation.
2. Tester workstation.
3. Technical reviewer workstation.
4. Demonstration operator workstation.
5. CI runner.
6. Local container runtime.
7. Demonstration runtime environment.
8. Deployment runtime environment.
9. Documentation build environment.
10. Release or Evidence-collection environment.

Role classification prevents the team from requiring every participant to install every tool. A developer may need Git, Java, Python, DDS tooling, a container runtime, build tools, and validation tools. A tester may need Git, a container runtime, test-execution tools, and log-inspection tools. A technical reviewer may need Git, documentation tools, and enough runtime tooling to reproduce acceptance Evidence. A deployment environment may need only runtime Artifacts, container runtime support, Configuration, credentials, and monitoring tools.

The checklist structure should support later automation. A completed checklist instance should contain enough information to help the team create setup scripts, such as Homebrew commands for macOS, package manager commands for Linux, Windows installation notes, container build

prerequisites, CI runner images, or deployment environment preparation scripts. The checklist should not require those setup scripts to exist immediately; it should capture tool names, version ranges, installation references, and verification commands in a format that automation can later use.
