STM32Cube Agentic AI installer Documentation

Release 1.0.0

Nov 03, 2026

Table Of Contents

1 STM32Cube for Agentic AI Installer 6
- 1.1 Scope . . . 6
- 1.2 Quick orientation . . . 7
- 1.3 Routing behavior . . . 7

2 Concepts 8
- 2.1 Cube bundle model . . . 8
- 2.2 Context bundle vs tool bundle . . . 8
- 2.3 Context bundle commands . . . 8
- 2.4 Why a context may install a tool . . . 9
- 2.5 Version management and SemVer . . . 9
- 2.6 Machine install vs project deployment . . . 10
- 2.7 Requirements of a context . . . 10
  - 2.7.1 Python of the eSW context . . . 11
- 2.8 Lock files and reproducibility . . . 12
- 2.9 Orchestration catalog . . . 12
  - 2.9.1 CMake build guidance without the Build Agentic AI context . . . 13
  - 2.9.2 STM32Cube user-journey skill . . . 13
- 2.10 Additive deployment behavior . . . 13

3 Prerequisites 14
- 3.1 cube bundle manager . . . 14
- 3.2 Installer bundle . . . 14
- 3.3 Operating system . . . 14
- 3.4 Disk space . . . 14
- 3.5 Network access . . . 14
- 3.6 PATH . . . 15

4 Commands Reference 16
- 4.1 Global options . . . 16
- 4.2 Output envelope . . . 16
  - 4.2.1 Success . . . 16
  - 4.2.2 Failure . . . 16
  - 4.2.3 Parsing guidance . . . 17
- 4.3 Key concepts . . . 17
- 4.4 Commands . . . 17
  - 4.4.1 setup . . . 18
  - 4.4.2 doctor . . . 20
  - 4.4.3 machine-bootstrap . . . 22
  - 4.4.4 machine-install . . . 24
  - 4.4.5 machine-discovery . . . 24
  - 4.4.6 show-catalog . . . 25
  - 4.4.7 list-supported-targets . . . 26
  - 4.4.8 project-add . . . 27
  - 4.4.9 project-deploy . . . 29
  - 4.4.10 project-deploy-installed . . . 33
  - 4.4.11 project-lock-show . . . 34
  - 4.4.12 project-remove . . . 35
  - 4.4.13 get-user-manual . . . 37

5 Target selection 38
- 5.1 Supported targets . . . 38
- 5.2 File placement by dimension . . . 38

5.3 Rule of thumb . . . 38

6 Lock Files . . . 40
6.1 Overview . . . 40
6.2 Cube project lock . . . 40
6.2.1 Bundles excluded from the Cube project lock . . . 40
6.2.2 Projects with a legacy Cube lock . . . 41
6.3 Services lock (`services-lock.json`) . . . 41
6.3.1 Managed artifact replay and conflicts . . . 41
6.3.2 Tainted services-lock concept . . . 42
6.3.3 Inspecting the services lock . . . 42
6.4 Reproducing a project environment . . . 43
6.4.1 Committing lock files to source control (recommended) . . . 43
6.4.2 When to use `project-lock-show` exported preset instead . . . 43
6.5 Key rules . . . 44

7 Project Housekeeping . . . 45
7.1 Why there is no `project-update` command . . . 45
7.2 The housekeeping levers . . . 45
7.3 Practical scenarios . . . 46
7.3.1 1) Catalog is behind, but a newer bundle is installed intentionally . . . 46
7.3.2 2) Intentionally mix services from different bundle versions . . . 47
7.3.3 3) Reconcile a preset to a single effective bundle version per context . . . 47
7.3.4 4) Keep preset diversity exactly as recorded . . . 48
7.3.5 5) Keep unpinned updates under lock control . . . 48
7.4 Connector upgrades and tool dependencies . . . 49
7.5 Recommended operating policy . . . 49
7.6 Related pages . . . 50

8 Usage . . . 51
8.1 0. Quick start: set up the toolkit in one command . . . 51
8.2 1. Recommended: full onboarding on a fresh machine . . . 52
8.3 2. One-shot non-interactive onboarding (CI or scripted) . . . 52
8.4 3. Reproducible deployment from a preset file . . . 52
8.5 4. Reproduce an existing project environment . . . 53
8.6 5. Add a single context to a project . . . 53
8.7 6. Add a context with explicit service selection . . . 54
8.8 7. Update a deployed context to a newer version (upgrade) . . . 54
8.9 8. Roll back to an older context version (downgrade) . . . 54
8.10 9. Two-step workflow: install first, deploy later . . . 55
8.11 10. Deploy to a specific IDE and agentic system . . . 55
8.12 11. Inspect what is installed and deployed . . . 55
8.13 12. Dry run before committing . . . 56
8.14 13. Remove a context from a project . . . 56
8.15 14. Access the user manual . . . 57

9 Troubleshooting . . . 58
9.1 Diagnostics and tips . . . 58
9.2 Error codes . . . 58
9.2.1 `ADVISORY_ONLY` . . . 59
9.2.2 `ALL_BUNDLES_FAILED` . . . 59
9.2.3 `ARTIFACT_DUPLICATE_DESTINATION` . . . 59
9.2.4 `ARTIFACT_USER_OWNED_CONFLICT` . . . 59
9.2.5 `ARTIFACT_CONNECTOR_SERVICE_CONFLICT` . . . 59
9.2.6 `ARTIFACT_CROSS_CONNECTOR_CONFLICT` . . . 60
9.2.7 `ARTIFACT_CONTENT_HASH_MISMATCH` . . . 60
9.2.8 `SERVICES_LOCK_SCHEMA_UNSUPPORTED` . . . 60
9.2.9 `PROJECT_REMOVE_REQUIRED_SERVICE_CONFIRMATION_REQUIRED` . . . 60
9.2.10 `ARTIFACT_ENTRYPOINT_MISSING` . . . 60

9.2.11 ARTIFACT_METADATA_INVALID . . . 60
9.2.12 ARTIFACT_KIND_UNDEFINED . . . 60
9.2.13 ARTIFACT_NAME_INVALID . . . 61
9.2.14 ARTIFACT_REFERENCE_KIND_MISMATCH . . . 61
9.2.15 ARTIFACT_REFERENCE_MISSING . . . 61
9.2.16 ARTIFACT_REFERENCE_UNKNOWN . . . 61
9.2.17 ARTIFACT_RESOURCE_OWNS_RESOURCE . . . 61
9.2.18 BUNDLE_INSTALL_FAILED . . . 61
9.2.19 BUNDLE_LOCKED_VERSION_NOT_INSTALLED . . . 62
9.2.20 BUNDLE_NOT_DEPLOYED . . . 62
9.2.21 BUNDLE_NOT_FOUND . . . 62
9.2.22 BUNDLE_NOT_IN_AI_CATALOG . . . 62
9.2.23 CATALOG_INVALID . . . 62
9.2.24 CATALOG_NOT_FOUND . . . 63
9.2.25 CATALOG_PARSE_ERROR . . . 63
9.2.26 CATALOG_VALIDATION_FAILED . . . 63
9.2.27 CUBE_BUNDLE_INFO_ERROR . . . 63
9.2.28 CUBE_LIST_ERROR . . . 63
9.2.29 CUBE_NOT_FOUND . . . 63
9.2.30 INVALID_OPTION_COMBINATION . . . 63
9.2.31 INVALID_USAGE . . . 64
9.2.32 JSON_FILE_WRITE_FAILED . . . 64
9.2.33 LEGACY_CUBE_PROJECT_LOCK_UNSUPPORTED . . . 64
9.2.34 LIST_SUPPORTED_TARGETS_FAILED . . . 64
9.2.35 LOCK_READ_ERROR . . . 64
9.2.36 MACHINE_DISCOVERY_FAILED . . . 64
9.2.37 MANUAL_NOT_FOUND . . . 65
9.2.38 MCP_CONFIG_PARSE_FAILED . . . 65
9.2.39 MCP_FORMAT_INCOMPATIBLE . . . 65
9.2.40 MCP_MERGE_CONFLICT . . . 65
9.2.41 MCP_SERIALIZATION_FAILED . . . 66
9.2.42 NO_SUPPORTED_TARGET . . . 66
9.2.43 PATH_TRAVERSAL_DETECTED . . . 66
9.2.44 PRESET_INVALID . . . 66
9.2.45 PRESET_NOT_FOUND . . . 66
9.2.46 PROJECT_DEPLOY_INVALID_OPTIONS . . . 66
9.2.47 PROJECT_NOT_FOUND . . . 67
9.2.48 PROJECT_REMOVE_FAILED . . . 67
9.2.49 PROJECT_REMOVE_IO_FAILED . . . 67
9.2.50 PROJECT_REMOVE_MANAGED_PATHS_MISSING . . . 67
9.2.51 PROJECT_REMOVE_PERMISSION_PRECHECK_FAILED . . . 67
9.2.52 ROUTING_SECOND_CHOICE_POLICY_INVALID . . . 68
9.2.53 ROUTING_SECOND_CHOICE_SCHEMA_UNSUPPORTED . . . 68
9.2.54 SERVICES_LOCK_UNRELIABLE_WRITE_FAILED . . . 68
9.2.55 SERVICES_SCHEMA_UNSUPPORTED . . . 68
9.2.56 SERVICES_VALIDATION_FAILED . . . 68
9.2.57 SHOW_CATALOG_FAILED . . . 68
9.2.58 TARGET_AMBIGUOUS . . . 69
9.2.59 TARGET_CHANGED_IN_DEPLOYED_PROJECT . . . 69
9.2.60 TARGET_COMBINATION_NOT_SUPPORTED . . . 69
9.2.61 TARGET_NOT_DETECTED . . . 69
9.2.62 TARGET_PROFILE_UNSUPPORTED . . . 69
9.2.63 USER_JOURNEY_POLICY_INVALID . . . 70
9.2.64 USER_JOURNEY_SCHEMA_UNSUPPORTED . . . 70
9.3 Codes of `setup` and `doctor` . . . 70
9.3.1 VS Code extension repair failures . . . 70
9.3.2 CONTEXTS_EMPTY . . . 70
9.3.3 CONTEXT_REQUIRES_VSCODE . . . 70

9.3.4 BUNDLE_IDE_NOT_SUPPORTED . . . 71
9.3.5 INVALID_PYTHON_CHOICE . . . 71
9.3.6 INVALID_VSCODE_PROFILE_NAME . . . 71
9.3.7 PROJECT_FOLDER_UNSAFE . . . 71
9.3.8 PROJECT_FOLDER_CREATE_FAILED . . . 71
9.3.9 SETUP_PROJECT_ALREADY_DEPLOYED . . . 71
9.3.10 SETUP_FAILED and DOCTOR_FAILED . . . 71
9.3.11 Warnings of setup and doctor . . . 72

1 STM32Cube for Agentic AI Installer

The **STM32Cube for Agentic AI installer** is the command-line entry point for setting up and maintaining the STM32Cube Agentic Context toolkit on your machine and in your projects. Use `cube agentic-ai` for installer commands. Context bundle names follow the `cube-agentic-ai-<connector>` pattern.

This reference explains command syntax, defaults, deployment state, and project maintenance. For a first installation, use the installation guide in SEE ALSO. For context capabilities and typical development requests, use the STM32Cube for Agentic AI documentation in SEE ALSO.

The reference is published online and is also included in the installer bundle. Run `cube agentic-ai get-user-manual --format html` to open the bundled HTML manual, or use `--format tldr` for a concise command summary. The bundled manual describes the installed installer version. The online manual can describe a newer release.

It gives you a stable command set to:

- **Set up** the toolkit in one command (`setup`): install on the machine, resolve what the contexts need, and deploy into a project.
- **Diagnose** what the installed contexts need to work, and fix it (`doctor`).
- **Install** STM32Cube Agentic Context bundles on your machine.
- **Deploy** them into a project folder, wiring skills, agents, instructions, prompts, and MCP servers to your agentic AI system.
- **Lock** the exact versions used in a project so the setup is reproducible.
- **Inspect** what is installed and what is deployed at any time.

Command results use machine-readable JSON, so both human operators and agentic AI systems can drive the same workflow. Interactive commands also print prompts. Help and version requests print text.

1.1 Scope

The installer covers:

Area	What the installer does
Machine setup	Install context bundles via the `cube` bundle manager
Requirements	Check what contexts need (`doctor`), resolve dependencies, and report prerequisites and VS Code extension commands
Tool dependencies	When a context requires a Cube tool, the `cube` bundle manager installs it automatically — no separate action is required
Project deployment	Deploy STM32Cube Agentic Context files into your project folder
Target wiring	Place files in the right location for your IDE and agentic AI system, or use the portable `.cube-ai/` layout
Version locking	Record exact context and tool dependency versions so projects are reproducible
Routing catalog	Generate a Markdown catalog from services declared by contexts deployed to the project

Contexts use consistent project-file ownership, so the installer can track the files it deploys for later updates and removal. When a context requires a Cube tool, the `cube` bundle manager installs that tool automatically.

The installer does **not**:

- Install VS Code extensions without authorization: `setup --install-vscode-extensions` installs declared extensions, and `doctor --fix --vscode-profile "<name>"` installs missing declared extension dependencies only in that explicitly named, existing profile. Other extension checks report commands to run. The commands `setup` and `doctor --fix` also run a bundle's declared Python-package installation. Neither installs Python itself.
- Maintain a separate catalog of tools — tool lifecycle is the responsibility of the `cube` bundle manager via context dependencies.
- Modify IDE settings beyond the documented project wiring and explicitly authorized extension installation.

1.2 Quick orientation

- *Concepts* — concise conceptual model: context bundle vs tool bundle, automatic tool dependencies, versioning, scope boundaries, lock files, and routing catalog.
- *Prerequisites* — what must be installed on your machine before running any command.
- *Commands reference* — full syntax, options, and defaults for all commands.
- *Target selection* — how `--ide` and `--agentic-system` interact, with the common deployment patterns.
- *Lock files* — how the installer records deployed versions and service selections, and how to reproduce a project environment.
- *Project housekeeping* — strategy guide for evolving deployments with add/remove/deploy flows.
- *Usage* — annotated command sequences for the most common workflows.
- *Troubleshooting* — error codes, their meaning, and the next action to take.

1.3 Routing behavior

Project commands use the `usage-router` mode by default. This mode generates the routing catalog from services declared by the contexts deployed to the project. It does not arbitrate overlapping context responsibilities. To opt in to overlap arbitration, pass `--routing-mode usage-router-with-arbitration` to a project deployment command.

2 Concepts

This page groups the core concepts used by the STM32Cube for Agentic AI installer.

2.1 Cube bundle model

The installer uses `cube` to install and manage STM32Cube Agentic Context bundles. Some contexts require Cube tool bundles, which `cube` installs automatically.

Installer commands install contexts on the machine and deploy selected context files into projects.

2.2 Context bundle vs tool bundle

The installer distinguishes two bundle roles:

- Context bundle:
  - Carries agent-facing artifacts (skills, agents, instructions, prompts, MCP wiring, routing inputs).
  - Contains the context capabilities selected for project deployment.
- Tool bundle:
  - Carries an executable or tool runtime used by one or more contexts.
  - Is usually consumed as a dependency rather than directly deployed by installer workflows.

Context bundle names follow the `cube-agentic-ai-<connector>` pattern. For example, the STM32Cube Build Context uses `cube-agentic-ai-build`, and the STM32CubeProgrammer Context uses `cube-agentic-ai-programmer`. The installer tracks context-owned project files for updates and removal.

In machine discovery output:

- Context bundles managed by installer workflows are classified as `IN_SCOPE`.
- Tool bundles required directly by installed contexts are classified as `DIRECT_AI_DEPENDENCY`.
- Unrelated bundles remain `OUT_OF_SCOPE`.

2.3 Context bundle commands

Each context bundle provides commands to show its purpose, version, and available services.

Recommended commands:

- `--help`: short purpose and usage summary.
- `--version`: installed context bundle version.
- `--list-ai-services`: machine-readable service catalog in JSON.

The `--list-ai-services` output is the most useful command for deployment planning because it declares, per service:

- `identifier`: stable service identifier used by service-selection workflows (for example project add service selection).
- `name`: service name used in routing catalog generation.

- `description`: user-facing description of the service intent.

Service identifiers such as `S1` belong to one context bundle. The same identifier can mean something different in another context. Inspect the installed version's output before making an interactive service selection:

```
cube cube-agentic-ai-esw --list-ai-services
```

The embedded-software context guide in the toolkit user documentation maps its service IDs to user-facing capabilities. A project's service lock records the selections already deployed there.

Example:

```
{
  "services": [
    {
      "identifier": "S1",
      "name": "st-docs.source-discovery",
      "description": "Find the right official ST document and return source-grounded identifiers␣
↪and locations."
    }
  ]
}
```

2.4 Why a context may install a tool

Some contexts use STM32Cube tools packaged as separate bundles. In that case, the context bundle declares the tool dependency.

When you install the context, the `cube` bundle manager resolves and installs that dependency automatically. This is expected behavior, not a side path.

Consequences:

- You do not need a separate installer command for that tool bundle.
- Project lock state records both context and resolved tool bundle versions.
- Reproducibility includes the context and its required tool bundles.

Scope limitation:

- Automatic installation covers only dependencies declared by the context bundle. Tools required by its skills but not declared as bundle dependencies must be installed separately. Consult that context's prerequisites.
- This applies only to tools delivered as Cube bundles.
- Tools not packaged as Cube bundles (for example IDE distributions or required editor extensions) are outside installer scope and must be installed independently.

2.5 Version management and SemVer

Bundle versions follow semantic versioning (SemVer).

You can:

- Pin an exact version, for example `cube-agentic-ai-esw@<version>`.
- Select versions through ranges when supported by the bundle manager and context dependencies.
- Use presets or lock state to replay an exact environment.

In practice:

- `machine-install` accepts a bundle reference with an optional version.
- `project-add` accepts a bundle reference with an optional version.
- `project-deploy --preset` can replay exact versions exported from an existing project.

The current installer catalog uses minimum-version ranges (`>=`) for its nine connector bundles, with no upper bound. Later major versions can satisfy these ranges. The required installer bundle, which carries the housekeeping and doctor skills, is pinned to the exact installer version: the installer never updates itself. Run `cube agentic-ai show-catalog --short false` to inspect the catalog shipped with your installer. `machine-bootstrap` enforces these catalog requirements, but does not proactively upgrade an already compliant version unless `--upgrade-installed-in-range true` is requested. Deployment with an unpinned bundle name instead reuses the highest installed semantic version, unless `--bundle-version-policy lock-driven` selects the project-locked version.

2.6 Machine install vs project deployment

The installer separates machine scope and project scope.

Machine scope:

- Installs context bundles once on the machine.
- Reuses installed bundles across multiple projects.

Project scope:

- Deploys selected context files into one project top folder.
- Wires files for the selected IDE and agentic AI system (or canonical layout).
- Maintains project lock state and merged MCP/routing artifacts for that project.

One machine install can serve many projects, but each project must be explicitly deployed.

2.7 Requirements of a context

A context bundle declares in its `services.json` (schema 3.1.0 and later) what its services need to work, beyond the Cube tool bundles that it depends on. The installer checks them with `doctor` and, during `setup`, resolves what it can.

Only `setup` and `doctor` look at these requirements. `machine-bootstrap`, `machine-install`, `project-add`, `project-deploy` and `project-deploy-installed` install and deploy contexts without checking or resolving them, so a context added or updated with one of them can lack its Python packages or its tool bundles. Run `cube agentic-ai doctor <project_top_folder> --fix` after them. It resolves the dependencies and reports the rest.

Kind	Meaning	What the installer does
`prerequis`	Something the user must ensure, for example a program on the machine.	Reports the remediation text and link. Never acts on it.
`dependenc`	Something the installer can resolve: a Cube bundle, a Python package, an extension.	Resolves it with the bundle-declared resolution (`setup`, `doctor --fix`). VS Code extensions are listed as manual by checks because a terminal cannot see the active profile. By default, the agent or user runs the commands; `setup --install-vscode-extensions` is the opt-in that runs `code` for the declared extensions.

A requirement applies to the services that it declares, or to all services of the bundle. An unmet prerequisite that covers all the services of a dependency blocks that dependency. The services concerned are reported as blocked in

`data.blockedServices`. The overall `readiness` is `READY`, `CHECK_MANUALLY` (some checks cannot be done from a terminal) or `ACTION_REQUIRED`.

The eSW context declares Python 3.10 or later for HAL/LL guidance. See *Python of the eSW context*.

Build and debug-DAP both declare `ides: ["vscode"]`. The command `setup` leaves them unavailable for targets other than VS Code. Build declares `stmicroelectronics.stm32-vscode-extension`. Debug-DAP declares that extension plus `ozzafar.debugmcpextension` (DebugMCP). A terminal cannot see whether an extension is installed in the active VS Code profile, so read-only `doctor` reports these checks as `manual`. `doctor --fix --vscode-profile "<name>"` verifies the named profile and installs missing declared extension dependencies there. Without an explicit profile, `--fix` leaves extensions manual. Existing extensions are not updated, and unavailable profiles are not created. `setup` reports the install command for the selected profile. With `--install-vscode-extensions`, it runs `code` for the declared extensions in the Default profile or each `--vscode-profile`. Otherwise the agent or user runs the commands or uses the Extensions view. Reload the VS Code window after the first installation. DebugMCP is required for debug tools and for flashing through the Programmer context in a project containing debug-DAP. A context can also declare the IDE it needs (`ides` in `services.json`, and the catalog entry). A context that needs VS Code is not deployed for another IDE target, and `setup` and `doctor` report it as unavailable instead of failing the others.

2.7.1 Python of the eSW context

The eSW context runs Python scripts. HAL/LL guidance (service S2) needs Python 3.10 or later, found as `py -3`, `python3` or `python`. HAL project migration is not delivered in this release. If a selected service declares additional Python packages, `setup` and `doctor --fix` can install them into the Python chosen for the project. The installer never installs Python itself.

`setup --python` makes the choice when `setup` creates the project:

`--python`	Python environment used by the eSW context
`auto` (default)	The Python found as `py -3`, `python3` or `python`. If that Python is a virtual environment or is managed by the operating system, the environment of `venv` is used instead.
`venv`	An environment inside the installed eSW bundle version, `cube-agentic-ai-esw/<version>/pybin/.venv` in the bundle installation directory (`cube --get-current-value CUBE_BUNDLE_PATH`). No other Python is changed.
`"<absolute path>"`	That Python executable, 3.10 or later.

The choice is saved in the project, one file per eSW version: `.cube-ai/esw-python/cube-agentic-ai-esw@<version>.json`. `venv` and a path are saved when the installation starts, even if pip then fails. The `auto` choice is saved once the packages are installed, as the Python it used. The command `doctor --fix` and the HAL/LL guidance scripts use the saved choice. The command `doctor` has no `--python` option. The file is not in a project lock and is not meant to be committed.

Important

The choice is made once. `setup` refuses a project that is already deployed (`SETUP_PROJECT_ALREADY_DEPLOYED`), so no installer command changes the Python of a deployed project.

Each project has its own choice. Projects that choose `venv` for the same eSW version share its environment. An environment belongs to one eSW version. When a project moves to another eSW version, that version has no choice file yet and starts from the project's most recent one: a project on `venv` stays on `venv`, with the environment of the new version.

Update eSW from version N to N+1 in a project that uses `venv`:

1. `cube agentic-ai project-remove cube-agentic-ai-esw <project_top_folder>`. The choice files stay in `.cube-ai/esw-python/`.
2. `cube agentic-ai project-add cube-agentic-ai-esw@<N+1> <project_top_folder> --services all`, with the `--ide` and `--agentic-system` options of the project. N+1 is deployed, but its declared requirements

have not been checked by `project-add`.

3. `cube agentic-ai doctor <project_top_folder> --bundles cube-agentic-ai-esw --fix`. It checks the new version's requirements using the saved Python choice and applies available fixes. Inspect `data.readiness`, `data.fixes[]`, and `data.nextSteps`. A Python prerequisite can require manual action. Package installation applies only when a selected service declares packages.

When a newer eSW version is installed on the machine, `doctor` gives these three commands, ready to paste, with the warning `NOT_LATEST_MACHINE_VERSION`. If step 3 reports unmet requirements, follow its remediation, then run step 3 again. For Python, check the interpreter version and path. If declared packages fail to install, check pip's proxy, certificate, and timeout settings. The environment of N stays in the bundle installation directory until N is removed from the machine with `cube bundle remove cube-agentic-ai-esw@<N>`, once no project uses it.

The same `doctor --fix` also creates the environment again when the eSW version was removed from the machine and installed again, or when the Python that created it was uninstalled (install Python 3.10 or later first if no other one is left).

2.8 Lock files and reproducibility

Two lock files provide reproducibility:

- `.settings/bundles-lock.store.json` (project top folder)
  - Owned by the `cube` bundle manager, shared with any other Cube-aware tool (e.g. an IDE).
  - Records resolved bundle versions, including dependencies.
- `.cube-ai/services-lock.json`
  - Owned by the installer.
  - Records deployed context bundle versions, selected services, and effective deployment target.

`project-lock-show` returns a machine-readable state that can be reused as a preset for `project-deploy --preset`.

2.9 Orchestration catalog

During project deployment, the installer assembles a routing catalog used by the agentic AI system.

Current feature level:

- The `cube-agentic-ai-routing` skill contains the catalog of services declared by contexts deployed to the project.
- The default `usage-router` mode generates the catalog without overlap arbitration.
- Pass `--routing-mode usage-router-with-arbitration` to opt in to arbitration when context responsibilities overlap.
- This skill is the authoritative routing source for STM32/STM32Cube/STM32 AI requests.
- Proper routing requires the agentic system to invoke it before planning or tool selection.
- The deploy commands expose a host-instruction update payload that asks to enforce this behavior in the host instruction layer.
- It is regenerated whenever deployment state changes (`project-add`, `project-deploy`, `project-deploy-installed`, `project-remove`).

The generated entry point depends on the deployment target:

Target	Routing skill entry point
GitHub Copilot	`.github/skills/cube-agentic-ai-routing/SKILL.md`
Claude Code	`.claude/skills/cube-agentic-ai-routing/SKILL.md`
Canonical layout	`.cube-ai/skills/cube-agentic-ai-routing/SKILL.md`

2.9.1 CMake build guidance without the Build Agentic AI context

When the Build Agentic AI context is not deployed to the project, the routing skill gains a short section, `CMake build without the Build Agentic AI context`. It is guidance, not a service, and does not override other installed connector skills, security policies or user approval requirements. It tells the agent that it may configure and build a CMake project with its presets, but must first check which `arm-none-eabi-gcc` will run and its version, instead of using the first compiler found on `PATH`. The toolchain file name in a preset, such as `GCC.14.3.1.cmake`, states an intended version and does not select a compiler. The agent reports any difference and asks before building, and it checks the compiler recorded in the CMake cache when it reuses a build directory.

The section is absent when the Build Agentic AI context is deployed, because its skills then govern builds. It is regenerated with the routing catalog.

2.9.2 STM32Cube user-journey skill

When the STM32CubeMX2 context (with CMake generation) and the build context are both deployed, the installer also generates the optional skill `cube-agentic-ai-user-journey-mx2-cmake-build` beside the routing catalog. It keeps separate requests consistent, from project creation to CMake generation, build, and debug:

- The `.ioc2` file and the generated CMake project are placed at the folder opened in VS Code by default.
- A request can cover several phases in one turn. The agent completes only the requested phases, in order, after checking their prerequisite gates. Peripheral, pin, clock, or middleware changes to an existing project use the routing catalog alone and do not trigger journey gates.
- Connector skills own each phase's destination, format, checks, and approval rules. They take precedence over the journey guidance.
- Before building, the CMake project must be **cubified** by the STM32Cube for VS Code extension. The agent asks you to run **Developer: Reload Window** from the Command Palette, then checks the `.settings/` and `.vscode/settings.json` entries written by the extension.
- For a single-phase request, the agent performs only that phase and tells you which constraints keep the later phases possible. If you choose another location, it is respected, and the agent explains the consequence for build and debug.
- Debug steps are included only when the DAP debug context is also deployed.
- Before building, the MX2 generation check must report CMake output at the workspace root, and the extension-generated cubification metadata must be present. The agent does not create that metadata itself or install missing bundles or extensions on the user's behalf.

The skill is regenerated with the routing catalog and removed when either required context is removed. It is not recorded in `services-lock.json`.

2.10 Additive deployment behavior

`project-add` and `project-deploy` are additive primitives:

- A successful run applies the requested deployment on top of current project state.
- Deploying a smaller service subset after a larger one does not remove previously deployed services.
- Use `project-remove` to explicitly remove services or a context from the project.

This additive behavior also applies to service selection in `services-lock.json`: newly selected services are merged with existing ones until removal is requested.

3 Prerequisites

Before running any `cube agentic-ai` command, the following must be in place on your machine.

3.1 cube bundle manager

The installer is a Cube bundle. The `cube` CLI must be installed and available on `PATH`.

To verify:

```
cube --version
```

If `cube` is not found, complete the bootstrap first: https://dev.st.com/stm32cube-agentic-ai/stm32cube-agentic-ai-start/index.html

3.2 Installer bundle

The installer bundle must also be installed. If bootstrap did not install it, run:

```
cube bundle --yes install cube-agentic-ai-installer
cube agentic-ai --version
```

The installer uses its Cube-managed Node.js runtime. A separate system Node.js installation is not required. Before deploying contexts, create or choose an existing project directory. The folder does not need to contain generated firmware or a Cube project yet.

3.3 Operating system

OS	Tested versions	Architecture
Windows®	Windows® 10 and 11, 64-bit	x86_64
Linux®	Ubuntu® 22.04, 24.04, 26.04 — Fedora® 43, 44	x86_64
macOS®	macOS® 15 and 26	Intel® (x86_64) and Apple® Silicon (aarch64)

Warning

Ubuntu® 20.04 is not supported due to an incompatible glibc version.

3.4 Disk space

At least **500 MB** free disk space in the bundle installation directory. The exact location depends on your OS and the bootstrap configuration. Run `cube --get-current-value CUBE_BUNDLE_PATH` to find it on your machine.

3.5 Network access

Network access is required for:

- Downloading bundles from the Cube registry during `machine-install` and `machine-bootstrap`.

- Downloading additional Python packages, if declared by a selected service, during `setup` and `doctor --fix`, from the configured pip package index (proxy, certificate authority, mirror).
- Connecting to remote MCP server endpoints declared by installed contexts at runtime.

Offline operation is possible for `show-catalog`, `list-supported-targets`, `project-lock-show`, and `project-deploy-installed` when the required local bundles and their dependencies are already available. Remote MCP services still need network access when used by the agent.

Each context can have additional prerequisites, such as editor extensions or tools not declared as bundle dependencies. Installing a context does not install these automatically. `cube agentic-ai setup` and `cube agentic-ai doctor` check the ones that the context declares and report what to do: the remediation of a prerequisite, or the `code --install-extension` command of an extension.

GitHub Copilot in VS Code requires the GitHub Copilot extension, and Claude Code in VS Code requires Anthropic's Claude Code extension. Install the appropriate agent extension yourself. The installer does not install or manage either one. The optional `--install-vscode-extensions` flag applies only to extensions declared by toolkit contexts. See the GitHub Copilot extension Marketplace page and the Claude Code for VS Code guide.

The eSW HAL/LL guidance service requires Python 3.10 or later. The connector does not install Python. `setup` checks the interpreter and can install additional packages declared by a selected service. Use `setup --python auto` (default), `--python venv`, or `--python "<absolute Python path>"` when `setup` creates the project. The choice of a deployed project is kept. The selection is recorded per project and ESW bundle version under `.cube-ai/esw-python/`. Projects selecting `venv` for the same ESW version share that bundle version's environment. See *Python of the eSW context*.

Build and debug contexts require VS Code and declare their extensions as dependencies. By default, setup reports the commands for the selected VS Code profile. The installer cannot inspect an active profile from the terminal. `setup --install-vscode-extensions` is optional and requires the `code` command on `PATH`. Otherwise, install from the Extensions view.

3.6 PATH

The directory where `cube` is installed must be on your shell `PATH`. The bootstrap script adds this automatically. On Windows, open a new PowerShell window after installation for the change to take effect.

4 Commands Reference

All commands share the same output format and exit-code contract.

Output format: JSON on stdout (success and failure), except help/version output and interactive prompts. **Exit codes:** `0` = success, non-zero = failure. **Prerequisite for all commands:** `cube` must be installed and on `PATH`.

4.1 Global options

Place global options before the command name, for example:

```
cube agentic-ai --verbose progress machine-bootstrap --dry-run
```

Option	Default	Description
`--verbose none|progress|st`	`none`	Control diagnostics on stderr. `none`: silent; `progress`: per-item progress for multi-item commands; `standard`: progress plus informational and error messages; `debug`: all diagnostics.
`--help`	off	Print human-readable help. Use `<command> --help` for command-specific options.
`--version`	off	Print the installer version.

4.2 Output envelope

Non-interactive command execution writes exactly one JSON object to stdout. The envelope fields are identical regardless of success or failure. Interactive commands can print prompts before the final JSON result. For JSON-only automation, use non-interactive service selection and deployment modes, and `project-remove --confirm-always` when confirmation would otherwise be required. Help and version requests print text instead of an envelope.

4.2.1 Success

```
{
  "ok": true,
  "command": "<stable-command-id>",
  "timestamp": "<ISO-8601 UTC>",
  "data": {}
}
```

Field	Type	Description
`ok`	`true`	Always `true` on success.
`command`	string	Stable command identifier, e.g. `"machine-bootstrap"`.
`timestamp`	string	ISO-8601 UTC timestamp of command execution, e.g. `"2026-08-06T10:00:00.000Z"`.
`data`	object	Command-specific payload. Shape varies per command; each *command section below* documents its fields and status codes.

4.2.2 Failure

```
{
  "ok": false,
  "command": "<stable-command-id>",
  "timestamp": "<ISO-8601 UTC>",
```

(continues on next page)

(continued from previous page)

```
  "error": {
    "code": "UPPERCASE_ERROR_CODE",
    "message": "Human-readable description of the failure.",
    "cause": "Root-cause detail for diagnostics.",
    "details": {}
  }
}
```

Field	Type	Description
`ok`	`false`	Always `false` on failure.
`command`	string	Stable command identifier.
`timestamp`	string	ISO-8601 UTC timestamp of command execution.
`error.code`	string	Stable uppercase error code, e.g. `CATALOG_NOT_FOUND`. The full list with recovery guidance is in *Troubleshooting → Error codes*.
`error.message`	string	Human-readable failure description.
`error.cause`	string	Root-cause detail intended for diagnostics.
`error.details`	object?	Optional structured context (present only when additional machine-readable detail is available).

4.2.3 Parsing guidance

- Read `ok` first to branch between success and failure paths.
- On success, the command-specific payload is always under `data`.
- On failure, the structured reason is always under `error.code` and `error.message`. Do not try to parse `error.message` programmatically. Use `error.code` for branching.
- `ok: true` does not always mean every sub-operation succeeded. Multi-item commands can report partial outcomes inside `data`. Inspect command-specific aggregate status fields and per-item statuses as documented in each command section.
- `timestamp` is informational. Do not use it for flow control.
- In non-interactive command execution, no other text is written to stdout. Progress messages and diagnostics go to stderr.

4.3 Key concepts

See *Concepts* for the conceptual model (bundle lifecycle, SemVer/version selection, machine vs project scope, lock files, orchestration catalog, and additive deployment behavior).

4.4 Commands

`setup` and `doctor` are the entry commands: they cover the whole journey of installing the toolkit and checking that it works. The other commands are the building blocks they use, for advanced control (pinned versions, selected services, presets, removal, upgrades).

Only `setup` and `doctor` handle the requirements that contexts declare: Python packages, tool bundles such as the STM32CubeMX2 application, and VS Code extensions. The building blocks (`machine-bootstrap`, `machine-install`, `project-add`, `project-deploy`, `project-deploy-installed`) never check or resolve them. After one of them, run `cube agentic-ai doctor <project_top_folder> --fix` to install what the deployed contexts need, and read its `data.nextSteps` for what only you can do.

4.4.1 setup

Install the toolkit on the machine and deploy it into a project in one run: install the contexts the target can use, resolve the dependencies they declare, create the project folder, deploy the contexts into it, and diagnose the result. The request is the consent: nothing is asked during the run.

```
cube agentic-ai setup [options]
```

Option	Default	Description
`--project <path>`	`<home>/my-agenti`	Project top folder, created when missing. The home folder and file system roots are refused (`PROJECT_FOLDER_UNSAFE`).
`--context <bundle,...>`	all contexts the target can use	Install only these catalog bundles. The installer bundle, which carries the housekeeping and doctor skills, is always added. A context that needs VS Code fails with another IDE target (`CONTEXT_REQUIRES_VSCODE`).
`--python <value>`	`auto`	Python used for context packages: `auto`, `venv`, or the absolute path of a Python executable. The choice is saved per project and ESW bundle version in `.cube-ai/esw-python/<bundle>@<version>.json`. `venv` uses the ESW bundle-version environment; projects selecting `venv` for that version share it, while each project can independently select a system Python or path.
`--vscode-<name>`	the Default profile	VS Code target only. Profile that receives the extensions; repeat the option for several profiles. The exact profile name is used; invalid names are refused.
`--ide <id>`	`vscode`	IDE target. See *Target selection*.
`--agentic <id>`	`github-co`	Agentic system target.
`--canonic`	off	Deploy under the `.cube-ai/` canonical layout only.
`--upgrade [true|fal`	`false`	`true`: upgrade eligible context bundles on the machine before deploying this new project, as `machine-bootstrap` does. `setup` refuses a project that already has a services lock, so this option does not update deployed project contexts. Use `machine-bootstrap --upgrade-installed-in-range true`, then the replacement commands from `doctor`, for an existing project. The installer bundle follows the installer version and is never upgraded by `setup`.
`--install`	off	Opt-in. The installer runs `code` itself to install the extensions that the installed contexts declare, in each `--vscode-profile` or in the Default profile. It applies to the VS Code target only (`--ide vscode`, the default, without `--canonical`): with another IDE it is skipped, and a log line, the warning `VSCODE_INSTALL_SKIPPED`, a next step and `data.vscodeInstall.reason` say why. It is also skipped when no installed context declares an extension or `code` is not on the `PATH`. It lists each profile first and skips one that cannot be listed (it never creates a profile), leaves an installed extension as it is (never `--force`), and reports each extension in `data.vscodeInstall`.
`--dry-run`	off	Report what `setup` would do and change nothing. See the limits below.

Behavior:

- A project folder may already contain user files, but if it has a services lock (`.cube-ai/services-lock.json`) setup refuses it with `SETUP_PROJECT_ALREADY_DEPLOYED`, before anything is installed. Use setup only for a folder not yet deployed by this installer. To change contexts in a deployed project, use `project-add` or `project-remove`, then `doctor <project_top_folder> --fix`.
- A bundle that fails to install does not stop the others. In that case, `data.status` is `PARTIAL`. Read `data.bundles[]`, `data.deploymentFailures[]` and `data.nextSteps[]`. The next steps give complete `project-add` and bundle-scoped `doctor --fix` commands to retry a failed installation or deployment. Setup itself cannot be rerun after any context was deployed. If every bundle installation fails, setup returns `ALL_BUNDLES_FAILED` before creating the project folder, and can be retried after fixing the cause.
- Python choices are project-scoped and made once, when `setup` creates the project. Setup cannot change the

choice of a deployed project. Setup passes the target project folder to declared Python checks and resolutions, and stores the selected interpreter under `.cube-ai/esw-python/<bundle>@<version>.json`. The Python executable itself is never installed. A new ESW bundle version without a choice file starts from the project's most recent choice. Another project can choose a different interpreter. See *Python of the eSW context*.

- An ESW setup checks and applies its idempotent Python dependency resolution even when the declared package ranges already pass, so the requested choice is recorded for that project. `--dry-run` does not write the setting.
- Progress goes to stderr: a line per step and per bundle, and a line every 10 seconds while a bundle installs or a resolution runs. A fresh installation can take several minutes: allow a long timeout, or run it in the background. Running it again resumes when it stopped before deploying. Once it has deployed, finish with `doctor <project_top_folder> --fix`. stdout carries only the JSON result. `--verbose none` silences stderr.
- All services of each context are deployed. To select services or pin versions, use `project-add`.

Outcome:

- **Returns:** `data.status` (`ALL_DEPLOYED` or `PARTIAL`), `data.readiness` (`READY`, `CHECK_MANUALLY` or `ACTION_REQUIRED`), `data.bundles[]`, `data.deployed[]`, `data.deploymentFailures[]`, `data.resolutions[]`, `data.licenses[]`, `data.requirements[]`, `data.blockedServices[]`, `data.unavailableContexts[]`, `data.notices[]`, and `data.nextSteps[]`. `ALL_DEPLOYED` describes bundle deployment, not requirement readiness. Read `data.readiness` and `data.nextSteps` for remaining work.
- `data.licenses[]` reports the identifiers declared by installed bundles. The context bundles declare `SLA0044 Rev 7 - February 2026`. STM32CubeCLI remains under SLA0048.
- **VS Code target:** when an installed context declares an extension, `data.vscodeExtensions[]` lists the declared extensions. `data.vscodeProfiles[]` gives, for each profile, a `verifyCommand` to list its extensions, one `installCommands` entry per declared extension (VS Code leaves an installed one unchanged), and an `openCommand`. `data.openCommand` opens the project in the **first** profile of `data.vscodeProfiles[]` (the Default profile when none is named): run the `verifyCommand` of that profile first, and do not run the open command if the profile is not found, because VS Code would create an empty profile of that name. `data.recommendations` reports the update of `.vscode/extensions.json`: `status` is `ADDED`, `UNCHANGED` or `NOT_UPDATED`. A file that has comments or an unexpected shape is not rewritten: `status` is `NOT_UPDATED`, `reason` says why, and the extensions to add are the `data.vscodeExtensions[]` entries. `RECOMMENDATIONS_NOT_WRITTEN` is raised only when writing the file fails. During setup, `code` runs only with `--install-vscode-extensions`; otherwise the agent or user runs the returned commands after verifying the profile.
- **Who runs `code`:** setup returns extension commands unless `--install-vscode-extensions` is supplied. Doctor returns commands unless **both** `--fix` and an explicit `--vscode-profile "<name>"` are supplied. That combination verifies the named profile and installs missing declared extension dependencies there. Doctor never installs into an implicit Default profile. The commands follow the VS Code command line reference:

Purpose	Command	Notes
List the extensions of a profile (`verifyCommand`)	`code --list-extensions --profile "<name>"`	Without `--profile`, the Default profile. For a profile that does not exist it reports that it was not found and creates nothing. Compare the extension ids without case.
Install an extension (`installCommand`)	`code --install-extension <publisher.extension> --profile "<name>"`	An extension that is already installed is left as it is. `--force` is never used, because it would update an extension that the user already has.
Open the project (`openCommand`)	`code "<project folder>" --profile "<name>"`	For a profile that does not exist, VS Code creates an empty profile of that name, so run it only after the verification.

Every path and profile name is in double quotes, so the commands can be pasted in PowerShell, cmd, bash and zsh. VS Code Insiders is started with `code-insiders`, which `setup` does not look for.

- **No declared extension:** nothing needs `code`, so `data.vscodeProfiles` is empty, there is no `data.openCommand`, and the next step says to open the project folder in VS Code. A `--vscode-profile` option

has no effect then, and a next step says so.

- **`code` not on the PATH:** `data.vscodeCli.onPath` is `false` and the warning `VSCODE_CLI_NOT_FOUND` is raised. Run none of the VS Code commands: the extensions were not checked, so they are not necessarily missing. Install them from the Extensions view, or accept the proposal that VS Code makes when the folder opens. To get `code`: on macOS, run the Command Palette command "Shell Command: Install 'code' command in PATH" (named "Install 'code' and optionally 'copilot' commands in PATH" in recent versions); on Windows and Linux, reinstall VS Code or add its `bin` folder to the `PATH` (Windows default: `AppData\Local\Programs\Microsoft VS Code\bin`).
- **Routing instruction:** `data.hostInstructionUpdate` holds the instruction to add to the host instruction file so that the agent consults the routing skill. `data.hostInstructionUpdate.instructionFile` names that file for the selected agentic system (`.github/copilot-instructions.md` for GitHub Copilot, `CLAUDE.md` for Claude Code, `AGENTS.md` for Cursor). A next step repeats it.
- **Other IDE targets:** the next step is to start your agent in the project folder.
- **Exit code:** `0` whenever the installation ran, including with unmet prerequisites. Read `data.readiness`.
- **Machine level:** installs Cube bundles and runs the commands that bundles declare as resolutions.
- **Project level:** creates the project folder, deploys the contexts, writes both project locks, recommends extensions, and stores the ESW Python choice under `.cube-ai/esw-python/` when an ESW Python dependency is resolved.

Dry run limits (`--dry-run`): the requirements of a bundle are declared in the bundle itself and can be read only once it is installed. A dry run therefore **cannot show the requirements of a bundle that would be installed**: those bundles are listed in `data.requirementsAfterInstall`, and the real run can still end with unmet prerequisites (for example Python) that the plan does not show.

- For a bundle that would be updated, requirements are read from the version installed today, not from the new one (each requirement carries `readFromVersion`).
- Licenses of bundles that are not installed yet are unknown (warning `LICENSES_UNKNOWN_BEFORE_INSTALL`).
- Versions to install come from the online index (warning `ONLINE_INDEX_UNREACHABLE` when it cannot be read).
- With `--dry-run`, `data.mode` is `DRY_RUN`, each entry of `data.bundles[]` is `INSTALL`, `UPDATE`, `KEEP` or `UNKNOWN`, and `data.applyCommand` is the command that applies the plan.
- Every command that `setup` and `doctor` print (`applyCommand`, the `--fix` command, the VS Code commands) is safe to copy and paste in PowerShell, cmd, bash and zsh: each path and profile name is in double quotes, spaces and Chinese, Japanese or Korean characters included. A value that a shell would interpret inside double quotes (a double quote, `$`, a backtick, `%`, `!`, a line break, a trailing backslash, or consecutive backslashes such as a UNC path) is never written wrongly: the command is left out and the next step says to run the same command yourself.

Example:

```
cube agentic-ai setup --dry-run
cube agentic-ai setup --project-folder ~/work/my-board
cube agentic-ai setup --contexts cube-agentic-ai-st-docs,cube-agentic-ai-sidekick-kb --project-
↪folder ~/work/my-board
```

4.4.2 doctor

Check the requirements that the installed contexts declare: tools, Cube bundles, Python packages, extensions. Use it when an issue arises, or to verify an installation.

```
cube agentic-ai doctor [<project_top_folder>] [options]
```

Argument / option	Default	Description
project_top_folder	machine scope	Check the contexts and services deployed in this project. Omit it to check the contexts installed on the machine, with all their services.
--bundles <bundle,...>	all	Check only these catalog bundles.
--fix	off	Resolve the unmet dependencies, then check again.
--vscode-profile <name>	the Default profile	VS Code profile of the project, one at most, written in the extension commands of data.vscodeProfile. The exact name is used; invalid names are refused (INVALID_VSCODE_PROFILE_NAME).

Requirement kinds:

- **prerequisite**: something the user must ensure. The installer reports its remediation text and link but never acts on it.
- **dependency**: something the installer can resolve. `--fix` resolves it, unless an unmet prerequisite of the same services blocks it.

Statuses: `satisfied`, `unmet`, `manual` (check by hand), `unknown` (the check could not run). Read-only extension checks are `manual` and carry `extensionId`. With `--fix --vscode-profile "<name>"`, successful verification/installation makes them `satisfied` for that invocation; failed installations are `unmet`, while an unavailable profile or `code` leaves them `unknown`. A subsequent read-only run returns extension checks to `manual` because it does not inspect the active window.

Doctor starts with an activity line on **stderr**, before checking the contexts. Progress and diagnostics never go to stdout. stdout remains JSON-only. Pass `--verbose none` to suppress stderr progress and diagnostics explicitly.

Outcome:

- **Returns:** `data.readiness`, `data.contexts[].requirements[]`, `data.blockedServices[]` (services that cannot work because of an unmet requirement), `data.skipped[]` (contexts not checked, with the reason: `NOT_INSTALLED`, `NOT_DEPLOYED`, `BUNDLE_VERSION_NOT_PRESENT_ON_MACHINE` or `AI_METADATA_INVALID`), `data.warnings[]`, `data.nextSteps[]`, and `data.fixes[]` with `--fix`.
- **Skipped contexts:** a skipped context is not known to be fine, so `data.readiness` is `ACTION_REQUIRED` while one is skipped. The only exception is a catalog context that is not installed when the whole machine is checked without `--bundles`, which is a choice (`setup --contexts`). For `NOT_DEPLOYED`, `data.nextSteps` gives `project-add` and `doctor --fix` commands for this folder and preserves the target recorded by another deployed context when available. `BUNDLE_VERSION_NOT_PRESENT_ON_MACHINE` is an error: the version deployed in the project is not installed on the machine (it was removed), so the scripts and the catalog that its services need may be missing and it cannot be checked. The entry carries `deployedVersion` and, when one is installed, `installedVersion`; its next step starts with `ERROR` and offers two ways out: restore the exact version with `cube bundle install <bundle>@<deployed version>`, or replace it with the latest installed one, `project-remove` then `project-add <bundle>@<installed version> --services all`; either way followed by `doctor <project_top_folder> --bundles <bundle> --fix`, because neither command resolves requirements (a restored ESW version, for example, has lost its Python environment).
- **No deployed contexts:** when the project has never been deployed, `doctor` suggests `setup`. If it has a services lock left after its contexts were removed, `setup` refuses it. The field `data.nextSteps` says so and suggests adding a context with `project-add <bundle> <project_top_folder>` using the intended target, then running `doctor --fix`.
- **Not on the latest version:** several versions of a bundle can coexist on a machine. When the version deployed in the project is installed but a newer one is installed too, the context is checked as usual, carries the warning `NOT_LATEST_MACHINE_VERSION` and `latestInstalledVersion`, and a next step gives the same replacement commands, ending with the `doctor --fix` command. It is a notice, not an error. `setup` never moves a deployed context by itself.

- **VS Code extensions:** `data.vscodeProfile` gives verify/install/open commands for the named profile, or Default when omitted. With `--fix` and an explicit profile, doctor runs verification and installs missing dependency extensions automatically. `data.vscodeInstall` reports the profile result (`PROCESSED`, `PROFILE_UNAVAILABLE`, or `CODE_FAILED`) and each extension (`INSTALLED`, `ALREADY_INSTALLED`, or `FAILED`); matching `data.fixes[]` entries carry `type: "vscode-extension"` and the status afterwards. It never creates a profile, uses `--force`, opens a window, or reloads VS Code. Without an explicit profile, extensions stay manual even with `--fix`; run the returned commands manually if desired. Reload the VS Code window after installation.
- **Exit code:** `0` whenever the diagnosis ran, including with unmet requirements. A failure to run it returns `ok: false`.
- **Machine level:** read-only without `--fix`. With `--fix`, installs Cube bundles (never recorded in a project lock) and runs declared resolutions. With an explicit `--vscode-profile`, it also installs declared extension dependencies in that profile. A project-scoped Python choice needs a project folder. Use project-scope doctor for that dependency. Without a project folder, that check is `unknown`, not a claim that Python packages are missing.
- **Project level:** read-only without `--fix`. With `--fix`, it applies only declared dependency resolutions. An ESW resolution can write its Python choice under that project's `.cube-ai/esw-python/`. It does not deploy contexts or change project locks.

Example:

```
cube agentic-ai doctor ~/work/my-board
cube agentic-ai doctor ~/work/my-board --bundles cube-agentic-ai-esw --fix
cube agentic-ai doctor ~/work/my-board --vscode-profile "STM32 dev"
cube agentic-ai doctor ~/work/my-board --vscode-profile "STM32 dev" --fix
```

4.4.3 machine-bootstrap

Install all STM32Cube Agentic Context bundles supported by the installer on the machine. Any required Cube tool dependency bundles are installed automatically by the `cube` bundle manager.

Scope note:

- `--upgrade-installed-in-range` applies to installer-catalog context bundles only.
- Transitive tool dependency bundles are resolved and installed by the `cube` bundle manager according to its standard dependency resolution policy.

```
cube agentic-ai machine-bootstrap [options]
```

Option	Default	Description
`--dry-run`	off	Report what would be installed without installing
`--upgrade-in` `true|false`	`false`	When `true`, also upgrades already installed in-scope bundles if a newer eligible version is available. When `false`, keeps the default behavior: install missing bundles and fix non-compliant installed bundles, without proactively upgrading already-compliant ones.

Outcome:

- **Returns:** command-level `data.status`, plus per-bundle status (`WOULD_INSTALL`, `WOULD_UPDATE`, `ALREADY_INSTALLED`, `INSTALLED`, `UPDATED`, or `FAILED`) in `data.results[]` for every context in the set.
- **Partial failure:** if at least one context is installed, updated, already installed, or would be installed/updated, the command returns `ok: true`, `data.status: "PARTIAL"`, and any failed context is reported as `status: "FAILED"` in `data.results[]` with a `reason`.
- **All failed:** if every processed context fails, the command returns `ok: false` with `error.code: "ALL_BUNDLES_FAILED"`. Per-bundle failure details are available in `error.details.results`.
- **Machine level:** context bundles are installed, kept as-is, or updated. Required tool dependencies are resolved and installed by the `cube` bundle manager.

- **Project level:** none.
- **Requirements:** not checked or resolved. Use `setup`, or run `doctor --fix` after the deployment.

Boolean value handling:

- For `--upgrade-installed-in-range [true|false]`, accepted values are case-insensitive: `true`, `false`, `1`, `0`, `yes`, `no`, `y`, `n`.
- If `--upgrade-installed-in-range` is provided without a value, it is interpreted as `true`.

Command status codes (in `data.status`):

Status code	Meaning
`ALL_DEPLOYED`	Every processed context was installed, updated, or already installed.
`PARTIAL`	At least one context succeeded and at least one context failed. Inspect `data.results[]`.
`NONE_DEPLOYED`	No context was processed, for example because the catalog was empty.
`DRY_RUN`	`--dry-run` is on; no install or update was performed. Inspect `data.results[]` for the planned per-bundle outcome.

Per-bundle status codes (in each `data.results[]` entry's `status` field):

Status code	Meaning
`INSTALLED`	Bundle was not present and has been successfully installed.
`UPDATED`	Bundle was already present and has been upgraded to a newer eligible version.
`ALREADY_INSTALLED`	Bundle was already present on the machine; nothing changed.
`WOULD_INSTALL`	`--dry-run` is on; bundle would be installed but was not.
`WOULD_UPDATE`	`--dry-run` is on; bundle would be upgraded/remediated but was not.
`FAILED`	Installation attempt failed. A `reason` field accompanies the entry with the error detail.

Idempotency and version update behaviour

`machine-bootstrap` behavior depends on the catalog `versionSpec` and on `--upgrade-installed-in-range`:

Catalog `versionSpec`	Matching rule	Action when `--upgrade-installed-in-range=false`	Action when `--upgrade-installed-in-range=true`
absent	any installed version accepted	Missing bundles are installed. Already-installed bundles are left unchanged.	Missing bundles are installed. Already-installed bundles can be upgraded when a newer eligible version is available.
exact `1.2.3`	must match exactly	Missing or non-compliant bundles are moved to `1.2.3`.	Same as `false`.
range `^1.0`	must satisfy the range	Missing or non-compliant bundles are moved to a compliant version.	Missing or non-compliant bundles are moved to a compliant version; already-installed compliant bundles can be upgraded to a newer eligible version.

The command remains machine-scoped and idempotent for repeated runs with the same options and unchanged bundle availability.

Idempotency by situation:

Situation	Outcome
All catalog contexts already installed at a version satisfying their spec	All `ALREADY_INSTALLED` (unless `--upgrade-installed-in-range=true` and a newer eligible version is available).
Newer within-range version available on server; installed version still satisfies spec; `--upgrade-installed-in-range=false`	`ALREADY_INSTALLED`. No change is applied.
Newer within-range version available on server; installed version still satisfies spec; `--upgrade-installed-in-range=true`	`UPDATED`. Highest eligible online version is installed.
Installed version does NOT satisfy the spec (range raised, exact version changed)	The bundle is moved to a compliant version; reported as `UPDATED` if it already existed, otherwise `INSTALLED` (dry-run: `WOULD_UPDATE` or `WOULD_INSTALL` respectively).
New contexts added to the catalog	New entries `INSTALLED`; existing entries `ALREADY_INSTALLED` or `UPDATED` depending on flag and available eligible versions.

4.4.4 machine-install

Install one STM32Cube Agentic Context bundle on the machine. The installer verifies that the bundle is supported before proceeding. To install another kind of bundle, use `cube bundle install` directly.

If the context requires a STM32Cube tool delivered as a bundle, the `cube` bundle manager installs that tool dependency automatically — no separate install command is needed.

```
cube agentic-ai machine-install <bundle_name>
```

Argument	Description
`bundle_name`	Context bundle identifier, for example `cube-agentic-ai-esw` or `cube-agentic-ai-esw@<version>`

Outcome:

- **Returns:** install status and resolved context bundle version.
- **Machine level:** the context bundle is installed or reused. Required Cube tool dependencies are installed by `cube`.
- **Project level:** none.
- **Requirements:** not checked or resolved. Use `setup`, or run `doctor --fix` after the deployment.

4.4.5 machine-discovery

Show what is currently installed on the machine. Context bundles installed through this installer are marked `IN_SCOPE`. Tool dependency bundles required directly by installed contexts are marked `DIRECT_AI_DEPENDENCY`. All remaining bundles are marked `OUT_OF_SCOPE`.

```
cube agentic-ai machine-discovery [options]
```

Option	Default	Description
`--json`	on	Generate a machine catalog JSON file in addition to stdout command JSON
`--no-json`	off	Disable machine catalog JSON file generation
`--ai-only`	off	Exclude `OUT_OF_SCOPE` bundles from stdout result and generated catalog file
`--json-file <path>`	`machine-catalog.json`	Output path for the generated machine catalog JSON file

Outcome:

- **Returns:** command-result JSON on stdout with discovered bundles (`name`, `version`, `category`). Category values are `IN_SCOPE`, `DIRECT_AI_DEPENDENCY`, or `OUT_OF_SCOPE`.
- **Generated file:** when `--json` is enabled (default), writes a JSON file containing only the bundle list array (no command envelope fields like `ok` or `error`).
- **Validation:** passing `--json-file` together with `--no-json` fails with a structured option-combination error.
- **Environment scope:** discovery uses `CUBE_BUNDLE_PATH` as the machine root and intentionally ignores `CUBE_BUNDLE_PATH_EXTRA` for listing and metadata classification.
- **Machine level:** none (read-only).
- **Project level:** none.

Machine-discovery error codes:

Error code	Meaning
`INVALID_OPTION_COMBINATION`	`--json-file` was passed while `--no-json` disabled file generation.
`INVALID_USAGE`	`--json-file` is empty while JSON file generation is enabled.
`CATALOG_NOT_FOUND`	Context catalog file is missing.
`CATALOG_PARSE_ERROR`	Context catalog file is not valid JSON.
`CATALOG_INVALID`	Context catalog fails schema validation.
`CUBE_LIST_ERROR`	`cube bundle list` failed while reading machine inventory.
`CUBE_BUNDLE_INFO_ERROR`	Failed to read installed context metadata for direct dependency classification.
`JSON_FILE_WRITE_FAILED`	Machine catalog JSON output file could not be created or written.
`MACHINE_DISCOVERY_FAILED`	Unexpected machine-discovery failure not covered by a specific typed error code.

4.4.6 show-catalog

Display the installer context catalog to discover which STM32Cube Agentic Context bundles it supports.

Purpose note:

- This command reads the catalog bundled with the installer. It does not query the online registry.

```
cube agentic-ai show-catalog [options]
```

Option	Default	Description
`--short true|false`	`true`	`true`: return compact items (`bundleName`, `description`). `false`: return full catalog entries.

Boolean value handling:

- For `--short [true|false]`, accepted values are case-insensitive: `true`, `false`, `1`, `0`, `yes`, `no`, `y`, `n`.

- If `--short` is provided without a value, it is interpreted as `true`.

Outcome:

- **Returns:** command-result JSON on stdout with `data.short` and `data.items`.
- **Machine level:** none (read-only).
- **Project level:** none.

Prerequisites:

- `cube` must be installed and on `PATH`.
- The installer bundle's local catalog must be available. No network connectivity is required.

Show-catalog error codes:

Error code	Meaning
`CATALOG_NOT_FOUND`	Context catalog file is missing.
`CATALOG_PARSE_ERROR`	Context catalog file is not valid JSON.
`CATALOG_INVALID`	Context catalog fails schema validation.
`SHOW_CATALOG_FAILED`	Unexpected show-catalog failure not covered by a specific typed error code.

4.4.7 list-supported-targets

Display the deployment targets supported by this installer release, that is the accepted values of the `--ide` and `--agentic-system` options used by the project commands.

```
cube agentic-ai list-supported-targets
```

This command takes no option.

Outcome:

- **Returns:** command-result JSON on stdout with `data.ides`, `data.agenticSystems`, `data.defaults` and `data.canonical`.
- **Machine level:** none (read-only).
- **Project level:** none.

Each entry of `data.ides` and `data.agenticSystems` carries the `id` to pass to the option and a human-readable `label`. `data.defaults` reports the values applied when the option is omitted.

`data.canonical` documents the `--canonical` option: instead of the native layout selected by `--ide` and `--agentic-system`, `--canonical` deploys a generic folder structure under the canonical root folder (`.cube-ai/`). The agentic system in use can then reshuffle that structure per its own needs.

Values are read from the installer deployment-targets catalog. A target appears only when it is marked supported for the release. This release supports the IDE targets `vscode` and `none`, and the agentic-system targets `github-copilot` and `claude-code`. To use Claude Code without an IDE binding, pass `--ide none --agentic-system claude-code`.

List-supported-targets error codes:

Error code	Meaning
`CATALOG_VALIDATION_FAILED`	Target matrix catalog is missing or fails schema validation.
`NO_SUPPORTED_TARGET`	Target matrix catalog flags no supported IDE or agentic-system target.
`LIST_SUPPORTED_TARGETS_FAILED`	Unexpected failure not covered by a specific typed error code.

4.4.8 project-add

Add one STM32Cube Agentic Context bundle to a project: installs it on the machine if needed, then deploys its files into the project folder and assembles the routing file.

```
cube agentic-ai project-add <bundle_name> <project_top_folder> [options]
```

Argument	Description
bundle_name	Context bundle reference: cube-agentic-ai-<connector>, cube-agentic-ai-<connector>@<version>, or an explicit semver range after @
project_top_folder	Absolute or relative path to the project top folder

Option	Default	Description
--services <id,...>	—	Deploy only the listed service IDs (plus required ones). Special value: all (deploy all services). Empty value is rejected.
--interactive [true|false]	false	Prompt for service selection when --services is omitted. Ignored when --services is provided. If --interactive is omitted, default and required services are deployed. Interactive prompts are written to stdout before the final command result, so this mode is intended for human terminal sessions rather than JSON-only stdout parsing. In prompts, y and yes are equivalent and case-insensitive; pressing Enter applies the displayed default.
--ide <id|none|>	vscode	IDE target for artifact placement
--agentic-system <id>	github-copilot	Agentic system target
--canonical	off	Force deployment under .cube-ai/ only; overrides --ide and --agentic-system placement
--autodetect [off|advisory|enforce]	off	Detect IDE and agentic system from the project environment. advisory: when --ide or --agentic-system is missing, return detected candidates as an expected advisory-only result and stop before deployment. enforce: apply detected values for any target not set explicitly, or fail on ambiguity. Explicit --ide/--agentic-system flags always take precedence over detection.
--bundle-version-policy [flexible|lock-driven]	flexible	Version-resolution policy for unpinned bundle_name (name without @version). flexible: reuse the highest installed semantic version. lock-driven: reuse project-locked version when available; fail with BUNDLE_LOCKED_VERSION_NOT_INSTALLED if that locked version is not installed locally. Explicit pinned name@version remains authoritative.
--routing-mode <mode>	usage-router-with-arbitration	Routing catalog assembly mode. usage-router: legacy catalog plus second-choice guidance when at least two services are installed, without overlap arbitration. usage-router-with-arbitration: add policy-driven arbitration when context responsibilities overlap.

Boolean value handling:

- For --interactive [true|false], accepted values are case-insensitive: true, false, 1, 0, yes, no, y, n.
- If --interactive is provided without a value, it is interpreted as true.

Outcome:

- **Returns:** deployment result with selected services, artifact paths, and deployment layout status.
- **Advisory mode:** when --autodetect-targets advisory stops early, the JSON result uses status: ADVISORY_ONLY and installStatus: SKIPPED and does not install or deploy anything.
- **Machine level:** bundle installed if not already present.

- Version selection is based on the CLI `bundle_name` reference passed to `project-add` (`name`, `name@version`, or `name@<semver range>`). An explicit reference is honored.
- If `bundle_name` is unpinned (no `@` reference), the command reuses the highest installed semantic version for that bundle name.
- An unpinned `project-add <bundle>` intentionally does not enforce installer-catalog `versionSpec` ranges, so it can select any installed version. Use an explicit `name@<semver range>` when the catalog constraint must be respected, including after a failed `setup` installation.
- Under `lock-driven`, unpinned `project-add <name>` uses the version already locked in the project when available. If that locked version is no longer installed locally, the command fails with `BUNDLE_LOCKED_VERSION_NOT_INSTALLED`.
- Explicit pinned references (`name@version`) remain authoritative and bypass policy fallback.
- **Project level:** deployment folder updated, project lock updated, routing catalog regenerated, and `mcp.json` merged.
- **Requirements:** not checked or resolved: the Python packages, tool bundles and VS Code extensions that the context declares are not installed. Run `cube agentic-ai doctor <project_top_folder> --bundles <bundle_name> --fix` afterwards.
- **Idempotency note:** idempotency is guaranteed only for repeated successful executions with identical inputs. After any failure or partial execution, the command resumes from the current state and may apply changes on top of existing artifacts.

Machine install status codes (in `data.installStatus`):

Status code	Meaning
`INSTALLED`	Bundle was not present and has been installed as part of this command.
`REUSED`	A satisfying version was already installed; no install was performed.

Cube project lock status codes (in `data.cubeProjectLockStatus`):

Status code	Meaning
`RECORDED`	Bundle reference recorded in `.settings/bundles.store.json`; `bundleVersion` is the version resolved there.
`EXCLUDED`	Bundle listed in `cube-project-lock-exclusions.json` (today only `cube-agentic-ai-installer`); tracked only in `services-lock.json` with the installed version.

See *Lock files* for the rationale.

Tool dependency output (in `data.dependencies`):

- Each entry reports the dependency `name`, locked `version` (omitted when removed), `lockStatus` (`ADDED`, `UPDATED`, `UNCHANGED`, or `REMOVED`), and `machineStatus` (`INSTALLED` or `REUSED`).
- `previousVersion` is included when the project-lock version changed or the dependency was removed.
- `newerInstalledVersion` identifies a newer installed version that satisfies the declared range but is not selected by the project lock. The command remains successful and adds `TOOL_DEPENDENCY_PINNED_BY_PROJECT_LOCK` to `data.warnings`, with the refresh hint in `data.warningMessages`. Run `cube bundle update --project` from the workspace when you intend to refresh its tool dependencies.

Target resolution priority for `project-add`:

The effective `--ide`, `--agentic-system`, and `--canonical` values are resolved in this strict priority order:

Priority	Source	Condition
1 (highest)	Explicit CLI flag	Flag was passed on the command line. Always wins.
2	Autodetection	`--autodetect-targets enforce` is set and the dimension was not resolved by a CLI flag. When `advisory`, detected candidates are reported but not applied.
3 (lowest)	ST default	No CLI flag or autodetection result resolved the dimension. Defaults: `--ide vscode`, `--agentic-system github-copilot`, `--canonical` off.

Canonical layout interaction:

`--canonical` is orthogonal to `--ide` and `--agentic-system`. When `--canonical` is specified:

- Target resolution still runs for `--ide` and `--agentic-system` (following the priority above).
- After resolution, `--canonical` overrides the deployment family: all artifacts are written under `.cube-ai/` regardless of the resolved IDE and agentic system values.
- Target specific host folders such as `.github/skills/` or `.vscode/` are not created or updated for deployed artifacts.
- The services lock records `ide: none` and `agenticSystem: none` to reflect that no native placement was performed.

For the full list of `--ide`/`--agentic-system` pairings and file placement rules, see *Target selection*.

Deployment layout status codes (in `data.deploymentLayout.status`):

Status code	Meaning
`PLACED_BY_INSTALLER`	Artifacts were placed directly in the locations expected by the selected IDE and agentic system. No further action required.
`AGENT_RESHUFFLE_REQUIRED`	`--canonical` was used: artifacts were placed in the portable `.cube-ai/` layout instead of the native locations. The agentic system must reorganize them to match its own recommended folder structure before use. The `reshuffleHint` field in the response provides guidance. If files are moved after deployment, `project-remove` still uses the canonical paths recorded in the lock and moved files may require manual cleanup.

4.4.9 project-deploy

Deploy supported STM32Cube Agentic Context bundles into a project in one pass.

```
cube agentic-ai project-deploy <project_top_folder> [options]
```

Argument	Description
`project_top_folder`	Absolute or relative path to the project top folder

Option	Default	Description
`--mode interactive|`	`interacti`	Propose bundles one by one, or deploy all at once. Implicitly `all` when `--preset` is provided. Interactive prompts are written to stdout before the final command result, so this mode is intended for human terminal sessions rather than JSON-only stdout parsing.

continues on next page

Table 3 – continued from previous page

Option	Default	Description
`--preset <config_file>`	—	Path to a JSON preset file that specifies exact context bundle versions and service provenance. Accepts the output of `project-lock-show` directly. When provided, interactive prompts and `--services-mode` are suppressed.
`--preset-no-`	off	Exact preset replay mode. Valid only with `--preset`: groups context services by provenance version and dispatches one pinned deployment per version group. When used without `--preset`, command fails with `PROJECT_DEPLOY_INVALID_OPTIONS`.
`--auto-insta true|false`	`true`	Install missing bundles automatically as prerequisites
`--services-m default|all|`	`default`	Service selection strategy. Ignored when `--preset` is provided.
`--ide <id|none|oth`	`vscode`	IDE target
`--agentic-sy <id>`	`github-co`	Agentic system target
`--canonical`	off	Force canonical deployment layout; overrides `--ide` and `--agentic-system` placement
`--autodetect off|advisory`	`off`	Detect IDE and agentic system from the project environment. `advisory`: report suggestions only. `enforce`: apply detected values for any target not set explicitly. Explicit `--ide`/`--agentic-system` flags always take precedence over detection.
`--dry-run`	off	Report what would be deployed without deploying
`--bundle-ver flexible|loc`	`flexible`	Forwarded to child `project-add` calls. In non-preset mode, `lock-driven` reuses project-locked versions when available and falls back to `flexible` for first introduction of a bundle. In preset mode, bundle references are already pinned (`name@version`) and are authoritative over the policy.
`--routing-mo <mode>`	`usage-rou`	Forwarded unchanged to each child `project-add` call. See the `project-add` option table for the meaning of each value.

Boolean value handling:

- For `--auto-install-missing [true|false]`, accepted values are case-insensitive: `true`, `false`, `1`, `0`, `yes`, `no`, `y`, `n`.
- If `--auto-install-missing` is provided without a value, it is interpreted as `true`.

Interactive yes/no handling (`--mode interactive`):

- Each bundle prompt is interpreted as: `Enter` = yes (default), `y`/`yes` = yes, any other non-empty answer = no.
- Parsing is case-insensitive.
- Prompts are handled through stdin/stdout and are printed before the final result JSON. Callers that need machine-readable stdout must combine `--mode all` with `--services-mode default` or `--services-mode all`, or use `--preset`.

Bundle version input model for `project-deploy`:

- This command has no positional `<bundle_name>` argument.
- Without `--preset`, bundle names and optional version specs come from the installer context catalog.
- With `--preset`, each bundle entry carries an exact pinned version (`name` + `version`) from the preset file.

Outcome:

- **Returns:** full deployment summary with per-bundle service selections, artifact paths, and deployment layout status.
- **Machine level:** missing context bundles are installed if `--auto-install-missing` is on. Required tool dependency bundles are installed by `cube`.
- Version-selection behavior:

  - In non-preset mode, per-bundle deployment is executed through `project-add` using bundle name references from the installer catalog. This intentionally does not enforce catalog `versionSpec` during deployment, so users can deploy newer bundle versions even when the local installer catalog is outdated.
  - As a consequence, when multiple versions are installed and no exact version is pinned in the effective bundle reference passed to `project-add`, the highest installed semantic version for that bundle name is reused.
  - In preset mode, the exact version declared in the preset entry is used (`name@version`) and is authoritative.
  - `project-deploy` forwards `--bundle-version-policy` to child `project-add` calls.
  - With `--bundle-version-policy lock-driven` in non-preset mode, bundles already present in project lock are replayed with their locked version; bundles not yet in project lock fall back to `flexible` for first introduction.

- **Project level:** deployment folder updated, project lock updated, routing catalog regenerated, and `mcp.json` merged.
- **Requirements:** not checked or resolved, for any deployed context. Run `cube agentic-ai doctor <project_top_folder> --fix` afterwards.
- **Idempotency note:** idempotency is guaranteed only for repeated successful executions with identical inputs. After any failure or partial execution, the command resumes from the current state and may apply changes on top of existing artifacts.

Routing-authority output fields:

When `ok: true`, `data.hostInstructionUpdate` provides a post-deploy action payload that helps enforce authoritative routing behavior in the host agentic system:

Field	Type	Description
`data.hostInstructionUpdate.requiredStatement`	string	Mandatory statement to place in host instructions so the `cube-agentic-ai-routing` skill is invoked before planning and its generated catalog selects the applicable connector skills and MCP servers.
`data.hostInstructionUpdate.callToAction`	string	Explicit action message requesting to add the mandatory statement to the host instruction file.

Target resolution with `--preset`:

The effective `--ide`, `--agentic-system`, and `--canonical` values are resolved in this strict priority order for every `project-deploy` invocation, whether or not `--preset` is used:

Priority	Source	Condition
1 (highest)	Explicit CLI flag	Flag was passed on the command line. Always wins.
2	Preset file field	`--preset` was provided and the preset contains the corresponding top-level field (`ide`, `agentic-system`, or `canonical`). Only used when the CLI flag was **not** passed.
3	Autodetection	`--autodetect-targets enforce` is set and the dimension was not resolved by a CLI flag or preset field. When `advisory`, detected candidates are reported but not applied.
4 (lowest)	ST default	No CLI flag, preset field, or autodetection result resolved the dimension. Defaults: `--ide vscode`, `--agentic-system github-copilot`, `--canonical` off.

Examples:

- `--ide none` passed + preset has `"ide": "vscode"` → effective IDE is `none` (CLI wins).
- No `--ide` passed + preset has `"ide": "none"` → effective IDE is `none` (preset fallback).
- No `--ide` passed + preset has no `ide` field → effective IDE is `vscode` (ST default).
- `--ide none` passed + no preset → effective IDE is `none` (CLI wins, ST default for others).

Canonical layout interaction:

`--canonical` is orthogonal to `--ide` and `--agentic-system`. When `--canonical` is specified (via CLI or preset):

- Target resolution still runs for `--ide` and `--agentic-system` (following the priority above).
- After resolution, `--canonical` overrides the deployment family: all artifacts are written under `.cube-ai/` regardless of the resolved IDE and agentic system values.
- Target specific host folders such as `.github/skills/` or `.vscode/` are not created or updated for deployed artifacts.
- The services lock records `ide: none` and `agenticSystem: none` to reflect that no native placement was performed.
- `--canonical` in a preset can be overridden by passing `--canonical` explicitly on the CLI (CLI priority 1 always wins).

For the full list of `--ide`/`--agentic-system` pairings and file placement rules, see *Target selection*.

Output `ok` field and partial success:

`ok: true` does **not** mean every bundle succeeded. The command continues after a per-bundle failure so that other bundles are not blocked. Check `data.status` and inspect `data.bundles[].outcome` to know the result for each bundle:

data.status	ok	Meaning
ALL_DEPLOYED	true	Every accepted bundle was deployed successfully.
PARTIAL	true	At least one bundle succeeded and at least one failed. Partially deployed bundles remain in the project.
NONE_DEPLOYED	true	No bundles were accepted (all rejected interactively, or the catalog/preset contained no contexts).
DRY_RUN	true	`--dry-run` was active; no mutations were performed.
—	false	Command-level failure (bad project root, catalog unavailable, preset invalid) that prevented any bundle from being attempted.

When `ok: true` and `status: PARTIAL`, each failed bundle has `outcome: "FAILED"` and an `error` object in its `data.bundles[]` entry.

Deployment warning output:

- `data.warnings` contains warning codes propagated from child `project-add` operations, such as lock-consistency warnings.
- `TOOL_DEPENDENCY_PINNED_BY_PROJECT_LOCK` means a newer compatible installed tool dependency is not selected because the project lock preserves its current version.
- Each successful `data.bundles[]` entry exposes the same dependency array under `detail.dependencies`.
- `data.warningMessages` contains human-readable, non-fatal deployment warnings. It is omitted when there are no such warnings.
- If every attempted bundle fails, warning messages discovered during candidate resolution are returned under `error.details.warningMessages` instead.
- When present, these fields are included in the JSON result regardless of `--verbose`. With `--verbose standard` or `--verbose debug`, warning messages are also written to stderr.

For `project-deploy-installed`, installed-inventory resolution detects catalog entries marked `required` that have no eligible installed version. The command returns each missing bundle in a human-readable `data.warningMessages` entry, but leaves it out of the candidate set and does not install it. The warning explains that installed-only behavior is being honored. Run `machine-bootstrap` to install catalog bundles, then rerun `project-deploy-installed` when appropriate. This warning does not change candidate selection or make the command fail.

4.4.10 project-deploy-installed

Deploy into a project using **only bundles already installed on the machine** as the candidate set. Use this command when you have deliberately installed exactly the bundles you want (via `machine-bootstrap` or `machine-install`) and want to deploy only those — nothing more, nothing less. Unlike `project-deploy`, which proposes from the full supported set, this command is driven entirely by your machine inventory, so there are no unexpected bundles and no automatic installs.

```
cube agentic-ai project-deploy-installed <project_top_folder> [options]
```

Argument	Description
`project_top_folder`	Absolute or relative path to the project top folder

Option	Default	Description
`--mode interactive|a`	`interacti`	Propose candidates one by one, or deploy all
`--services-mo default|all|i`	`default`	Service selection strategy applied per selected bundle
`--ide <id|none|othe`	`vscode`	IDE target
`--agentic-sys <id>`	`github-co`	Agentic system target
`--canonical`	off	Force canonical deployment layout; overrides `--ide` and `--agentic-system` placement
`--autodetect- off|advisory|`	`off`	Detect IDE and agentic system from the project environment. `advisory`: report suggestions only. `enforce`: apply detected values for any target not set explicitly. Explicit `--ide`/`--agentic-system` flags always take precedence over detection.
`--dry-run`	off	Report what would be deployed without deploying
`--routing-mod <mode>`	`usage-rou`	Forwarded unchanged to each child `project-add` call. See the `project-add` option table for the meaning of each value.

Out-of-scope bundles discovered on the machine are never deployment candidates for this command.

Target resolution priority for `project-deploy-installed`:

The effective `--ide`, `--agentic-system`, and `--canonical` values are resolved in this strict priority order:

Priority	Source	Condition
1 (highest)	Explicit CLI flag	Flag was passed on the command line. Always wins.
2	Autodetection	`--autodetect-targets enforce` is set and the dimension was not resolved by a CLI flag. When `advisory`, detected candidates are reported but not applied.
3 (lowest)	ST default	No CLI flag or autodetection result resolved the dimension. Defaults: `--ide vscode`, `--agentic-system github-copilot`, `--canonical` off.

Canonical layout interaction:

`--canonical` is orthogonal to `--ide` and `--agentic-system`. When `--canonical` is specified:

- Target resolution still runs for `--ide` and `--agentic-system` (following the priority above).
- After resolution, `--canonical` overrides the deployment family: all artifacts are written under `.cube-ai/` regardless of the resolved IDE and agentic system values.

- The services lock records `ide: none` and `agenticSystem: none` to reflect that no native placement was performed.

For the full list of `--ide`/`--agentic-system` pairings and file placement rules, see *Target selection*.

`project-deploy-installed` has no `--preset` option. The only sources for target resolution are CLI flags, autodetection, and ST defaults (no preset field can override a CLI flag here).

Selection behavior:

- `--mode` controls bundle candidate selection (`interactive` prompts per bundle, `all` accepts all candidates).
- `--services-mode` controls per-bundle service selection (`default`, `all`, `interactive`).
- When the bundle reference does not pin an exact version and several installed versions of the same bundle are eligible, the highest installed semantic version is selected for deployment.
- This command is effectively pinned by construction (`name@version` candidates), so it does not expose `--bundle-version-policy`.

Outcome:

- **Returns:** deployment summary with per-bundle service selections, artifact paths, and deployment layout status.
- Successful bundle details include `detail.dependencies` with project-lock and machine-install statuses for transitive tool dependencies.
- A newer compatible installed version that is not used because of the project lock adds `TOOL_DEPENDENCY_PINNED_BY_PROJECT_LOCK` to `data.warnings`. Deployment remains successful.
- **Machine level:** none (no bundle install).
- **Project level:** deployment folder updated, project lock updated, routing catalog regenerated, and `mcp.json` merged.
- **Requirements:** not checked or resolved, for any deployed context. Run `cube agentic-ai doctor <project_top_folder> --fix` afterwards.
- **Idempotency note:** idempotency is guaranteed only for repeated successful executions with identical inputs. After any failure or partial execution, the command resumes from the current state and may apply changes on top of existing artifacts.

Routing-authority output fields:

When `ok: true`, `data.hostInstructionUpdate` provides the same post-deploy action payload as `project-deploy`:

Field	Type	Description
`data.hostInstructionUpdate.requiredStatement`	string	Mandatory statement to place in host instructions so the `cube-agentic-ai-routing` skill is invoked before planning and its generated catalog selects the applicable connector skills and MCP servers.
`data.hostInstructionUpdate.callToAction`	string	Explicit action message requesting to add the mandatory statement to the host instruction file.

4.4.11 project-lock-show

Display the current project lock state, including deployed context bundle versions, selected services, and the exact versions of required tool dependency bundles.

The command output is machine-readable JSON and is a valid preset file for `project-deploy --preset`. It captures context bundle versions and resolved tool dependency bundle versions so the same environment can be replayed on another machine or project.

```
cube agentic-ai project-lock-show <project_top_folder>
```

Argument	Description
`project_top_folder`	Absolute or relative path to the project top folder

Outcome:

- **Returns:** machine-readable lock state including context bundle versions, resolved tool dependency bundle versions, selected services, deployment target per bundle, and managed file ownership records.
- **Machine level:** none (read-only).
- **Project level:** none.
- **State model:** reports the current merged state per bundle from `services-lock.json`. If `project-add` or `project-deploy` is run multiple times for the same bundle with different service selections, all selected services remain listed until removed by `project-remove`.
- **Warnings and integrity:** when consistency issues are detected (for example context version drift between locks), output remains `ok: true` and includes warning information. If present, `services-lock.json.integrity` is also reported so automation can detect degraded lock reliability.
- **Warning field location:** warnings are emitted in the command stdout JSON success payload under `data.warnings`.

4.4.12 project-remove

Remove one deployed STM32Cube Agentic Context bundle from a project, reverting the effect of `project-add`.

```
cube agentic-ai project-remove <bundle_name> <project_top_folder> [options]
```

Argument	Description
`bundle_name`	Context bundle identifier to remove from the project
`project_top_folder`	Absolute or relative path to the project top folder

Option	Default	Description
`--service <id,...>`	—	Remove only the listed service IDs. If omitted, all deployed services of the bundle are removed.
`--confirm`	off	Bypass the confirmation prompt when removing a required service while other services from the same context remain. Not needed when removing the whole bundle or all deployed services.
`--dry-run`	off	Report what would be removed without removing anything
`--routing <mode>`	`usage-rou`	Routing catalog assembly mode used when regenerating the routing catalog after removal. See the `project-add` option table for the meaning of each value.

Outcome:

- **Returns:** list of removed files, updated lock paths, and remaining deployed services (if partial removal).
- **Machine level:** none (bundle remains installed on the machine).
- **Project level:** selected deployment files removed, project lock updated, routing catalog regenerated.

Project-remove error codes:

Error code	Meaning
BUNDLE_NOT_DEPLOYED	The requested bundle has no entry in .cube-ai/services-lock.json for this project.
SERVICES_VALIDATION_FAILED	--services is empty or includes service IDs not currently deployed for this bundle.
PROJECT_REMOVE_REQUIRED_SERVICE_CONFIRMATION_REQUIRED	A partial removal includes a service recorded with the |required provenance tag; rerun with --confirm-always or confirm the interactive prompt.
PROJECT_REMOVE_MANAGED_PATHS_MISSING	One or more managed artifact paths are missing from lock metadata or missing on disk during remove preflight.
PROJECT_REMOVE_PERMISSION_PRECHECK_FAILED	One or more managed artifact paths cannot be removed due to filesystem permission checks.
PROJECT_REMOVE_IO_FAILED	Deletion failed during the remove commit phase after preflight succeeded.
SERVICES_LOCK_UNRELIABLE_WRITE_FAILED	After partial deletion failure, writing degraded-state marker (integrity.state = UNRELIABLE) to services lock failed.
SERVICES_LOCK_READ_FAILED	.cube-ai/services-lock.json is present but unreadable or invalid JSON.
SERVICES_LOCK_WRITE_FAILED	Writing .cube-ai/services-lock.json failed.
PROJECT_REMOVE_FAILED	Unexpected project-remove failure not covered by a more specific typed error code.

For recovery steps, see *Troubleshooting → Error codes*.

Behavior notes:

- **Removal authority:** project-remove uses <project_top_folder>/.cube-ai/services-lock.json as the source of truth to determine removable services and managed files.
- **Required services:** each selectedServiceProvenance entry in the lock ends in either |required or |optional, copied from the deployed bundle manifest. When --services would remove a required service while another service from the same context remains, the command warns that the remaining services may be impeded and requires explicit confirmation. Removing the whole bundle or all deployed services does not require this confirmation. The tag is metadata only. The installer does not store a dependency graph.
- **Path authority model:** removal candidates are derived from lock-recorded per-service managed artifact records (managedArtifacts).
- **Out-of-band content:** files or services added manually outside installer commands are not tracked by services-lock.json and are not removed automatically.
- **Requirements are kept:** what setup and doctor --fix installed for the context stays, because other projects can use it: Cube bundles such as the STM32CubeMX2 application, Python packages and the ESW environment of the bundle version (removed with that version by cube bundle remove), and VS Code extensions. The project keeps its .cube-ai/esw-python/ choice files, which a later version of the context reuses, and the .vscode/extensions.json recommendations.
- **Canonical reshuffle caveat:** if deployment was done with --canonical and files were later moved to non-canonical locations, project-remove resolves removal candidates from canonical-managed paths. Moved files may remain and require manual cleanup.
- **Tainted lock state:** after remove-time I/O failures with partial deletion, or when lock-source drift is detected, services-lock.json may be marked tainted (integrity.state = UNRELIABLE). Subsequent mutating commands still run but emit warning information.
- **No manifest consistency check:** the command does not validate lock-versus-bundle-manifest consistency before removal.
- **Project scope only:** this command never uninstalls bundles from machine inventory. It only updates project deployment state.

4.4.13 get-user-manual

Access the installer user manual.

```
cube agentic-ai get-user-manual [--format <tldr|markdown|html>]
```

Option	Description
`--format tldr`	Return a concise quick reference directly in JSON (`data.content`). Default when `--format` is omitted.
`--format markdown`	Return the on-disk paths to the Markdown source files (for AI agent use)
`--format html`	Open the HTML documentation in the system default browser (for human use)

Outcome:

- **Returns:** one of: concise TLDR content (`tldr`), Markdown folder path(s) (`markdown`), or HTML entry point path (`html`).
- **Machine level:** none.
- **Project level:** none. (`--format html` opens the browser as a side effect.)

5 Target selection

The installer resolves two target dimensions:

- `--agentic-system` controls where **skills, agents, instructions, the routing catalog, and the MCP configuration file** are placed (for example `.github/` and `.mcp.json` for GitHub Copilot CLI, `.claude/` and `.mcp.json` for Claude Code). The agentic system is the MCP client, so it decides which MCP file is read.
- `--ide` changes the MCP configuration file only when the agentic system reads a file owned by that IDE. Today this applies to GitHub Copilot inside VS Code, which reads `.vscode/mcp.json`. The IDE also decides which IDE extensions are proposed (`--install-vscode-extensions`).

When `--ide none` is used, no IDE-native folder is created.

5.1 Supported targets

Run `cube agentic-ai list-supported-targets` to see the targets marked supported in your release: `--ide vscode|cursor|none` and `--agentic-system github-copilot|cursor|claude-code`. The CLI accepts no other target ID.

Only the six pairs in the table below are accepted. `project-add`, `project-deploy` and `setup` reject any other pair, for example `--ide cursor --agentic-system github-copilot`, with `TARGET_COMBINATION_NOT_SUPPORTED`. `--canonical` skips this check because it writes no host-specific file.

5.2 File placement by dimension

Flags	MCP config file	Agentic content
`--ide vscode --agentic-system github-copilot` (default)	`.vscode/mcp.json`	`.github/`
`--ide none --agentic-system github-copilot` (Copilot CLI)	`.mcp.json`	`.github/`
`--ide cursor --agentic-system cursor` (Cursor IDE)	`.cursor/mcp.json`	`.cursor/` (rules as `.cursor/rules/*.mdc`)
`--ide none --agentic-system cursor` (Cursor CLI)	`.cursor/mcp.json`	`.cursor/` (rules as `.cursor/rules/*.mdc`)
`--ide vscode --agentic-system claude-code` (Claude Code extension)	`.mcp.json`	`.claude/`
`--ide none --agentic-system claude-code` (Claude Code CLI)	`.mcp.json`	`.claude/`
`--canonical`	`.cube-ai/mcp.json`	`.cube-ai/`

A project has one target. Once a context is deployed, `project-add`, `project-deploy` and `setup` reject any other `--ide`, `--agentic-system` or `--canonical` with `TARGET_CHANGED_IN_DEPLOYED_PROJECT`, and change nothing. To move a project to another target, run `project-remove` for each deployed context, then deploy again with the new target.

5.3 Rule of thumb

GitHub Copilot in VS Code requires the GitHub Copilot extension, and Claude Code in VS Code requires Anthropic's Claude Code extension. The installer does not install or manage either extension. Install them separately from the GitHub Copilot extension Marketplace page and the Claude Code for VS Code guide. The optional `--install-vscode-extensions` flag applies only to extensions declared by toolkit contexts.

- **Pick the agentic system you chat with**: it decides every deployed file, including the MCP config file.
- **Add `--ide vscode`** when that agentic system runs inside VS Code: VS Code extensions are then proposed, and GitHub Copilot uses `.vscode/mcp.json`.
- **Same host for IDE and agentic system** (e.g. Cursor): use matching values `--ide cursor --agentic-system cursor`.
- **Portable layout only**: use `--canonical` when you want all deployed files under `.cube-ai/` and are willing to reshuffle them later. Target specific host folders such as `.github/skills/` or `.vscode/` are not created or updated for deployed artifacts.

Deployment status follows the same split:

- `PLACED_BY_INSTALLER` means files were written in the selected native locations.
- `AGENT_RESHUFFLE_REQUIRED` means `--canonical` was used and the portable layout still needs host-specific reorganization.

6 Lock Files

The installer maintains two complementary lock files in every project to record exactly what was deployed and which versions are in use. Together they make project setups fully reproducible.

6.1 Overview

File	Location	Owned by	Purpose
Cube project lock	`<project>/.settings/bundles-lock.store.json`	`cube` bundle manager (shared with any other Cube-aware tool, e.g. an IDE)	Records every bundle reference — context bundles and their tool dependencies — with exact resolved versions
Services lock	`<project>/.cube-ai/services-lock.json`	Installer	Records deployed context service state, deployment target, ownership, paths, and hashes for managed files

The two files are separate by design: bundle version resolution is the `cube` bundle manager's responsibility (`.settings/bundles-lock.store.json`, at the project top folder), while service selection is the installer's (`services-lock.json`).

6.2 Cube project lock

The Cube project lock is created and updated by the `cube` bundle manager every time you run a command that adds a context to a project (`project-add`, `project-deploy`, `project-deploy-installed`). You never edit it directly. It lives at the project top folder (`<project>/.settings/`), the same location any other Cube-aware tool (such as an IDE) uses, so a project only ever has one Cube bundle lock.

It records:

- Every context bundle added to the project, with the exact resolved version, except the bundles excluded from the Cube project lock (see below).
- Every tool bundle installed as a transitive dependency of a context, with the exact resolved version and platform.

This is the file that drives reproducible replay: when you run `project-deploy --preset` with a lock exported by `project-lock-show`, the installer uses the exact version list from the lock to restore the environment on a new machine or project folder.

6.2.1 Bundles excluded from the Cube project lock

`cube` runs a command with the bundle version recorded in the Cube project lock of the current folder. If the installer bundle (`cube-agentic-ai-installer`) were recorded there, every `cube agentic-ai ...` started inside the project would keep running that old installer, even after `cube bundle update cube-agentic-ai-installer` reports "Everything is up to date."

The installer therefore never records the bundles listed in its `cube-project-lock-exclusions.json` data file in the Cube project lock. Today the list contains only `cube-agentic-ai-installer`. These bundles are still deployed normally and are tracked only in `services-lock.json`, with the installed version used for the deployment.

- `project-add` reports `cubeProjectLockStatus` in its JSON output: `RECORDED` or `EXCLUDED`.

- `project-lock-show` lists an excluded bundle as a `connector` with its services-lock version and raises no `CONNECTOR_MISSING_FROM_RESOLVED_LOCK` warning for it.
- A project with only excluded bundles deployed has no `.settings/` Cube lock.

6.2.2 Projects with a legacy Cube lock

Projects deployed by an older installer version may still have their own Cube bundle lock under `<project>/.cube-ai/.settings/bundles.store.json`. This legacy layout is no longer supported: `project-add`, `project-remove`, and `project-lock-show` fail fast with `LEGACY_CUBE_PROJECT_LOCK_UNSUPPORTED` instead of silently reconciling it with the shared `.settings/` lock. Reconcile the bundle references into `<project>/.settings/bundles.store.json` (or remove the legacy folder once its references are reflected there) before re-running the command.

6.3 Services lock (`services-lock.json`)

The services lock is created by the installer on the first `project-add` or `project-deploy` invocation and updated on every subsequent deployment. It is a JSON file.

It records, for each deployed context bundle:

- The exact bundle version that was deployed (taken from the Cube project lock, or the installed version for bundles excluded from it).
- The service IDs that were selected for deployment.
- The `selectedServiceProvenance` entry for each service, tagged with `|required` or `|optional` according to the bundle manifest.
- The effective deployment target (`ide`, `agenticSystem`, `fileFamily`) at the time of deployment.
- The per-service managed artifact records, including identity, destination, version, and hash (used by `project-remove` and collision preflight).
- The logical owner, source, destination, kind, and SHA-256 content hash of each managed artifact.
- MCP server ownership records, including the project manifest path, server-map key path, server name, and whether the installer introduced the key.
- The timestamp of the last deployment.

The recorded tag is used during partial `project-remove` operations. If a requested removal includes a `|required` service and leaves another service from the same context deployed, the user must explicitly confirm because the remaining services may be impeded. Full bundle removal, or removal of all deployed services, does not require confirmation. This tag is not a dependency graph and does not identify which remaining service might be affected. For backward compatibility, an untagged legacy provenance entry is treated as `|optional` (`required: false`).

Service merge behavior:

- The services lock stores the current selected-services set per bundle.
- If you run `project-add` or `project-deploy` multiple times for the same bundle with different service selections, newly selected services are merged with the existing entry.
- `project-lock-show` therefore reflects the current merged deployment status for each context bundle.

6.3.1 Managed artifact replay and conflicts

When a deployment encounters an existing destination, the installer consults `services-lock.json` before reporting a conflict:

- An unchanged artifact owned by the same logical context service and artifact may be replaced. The installer logs an informational update and considers the artifact deployed.
- A destination owned by another context is rejected with `ARTIFACT_CROSS_CONNECTOR_CONFLICT`.

- A destination claimed by another incoming artifact is rejected with `ARTIFACT_DUPLICATE_DESTINATION`.
- A destination not present in the services lock is rejected with `ARTIFACT_USER_OWNED_CONFLICT`.
- A destination owned by another service or artifact in the same context is rejected with `ARTIFACT_CONNECTOR_SERVICE_CONFLICT`.
- An installer-managed file whose current contents differ from its recorded hash is rejected with `ARTIFACT_CONTENT_HASH_MISMATCH` only when the incoming service version is the same as the recorded version. A different version of the same logical service is treated as an intentional upgrade and may replace the previous artifact.

This makes preset replay idempotent for unchanged installer-managed artifacts while protecting user files and context boundaries. The lock records the hash and writing version of the bytes last written by the installer. It does not claim ownership of manually created files.

Ownership-aware replay requires services-lock schema `1.1.0`. Older or unknown lock schemas must be replaced through a clean deployment before replay. The installer does not infer ownership or hashes from legacy path-only records.

`project-remove` behavior with lock authority:

- `project-remove` uses `services-lock.json` as the project source of truth to determine which services are currently deployed for a context.
- The command removes installer-managed deployment artifacts derived from that lock state.
- MCP keys are removed from recorded manifests only when the lock marks them installer-owned and no remaining service or bundle owns the same key. The source bundle is not required to be installed for this cleanup.
- Legacy locks without MCP ownership records leave MCP keys unchanged.
- Manually added services/files that were not deployed through installer commands are outside lock tracking and are not removed automatically.
- The command does not perform a consistency audit between `services-lock.json` and bundle manifest files before removal.
- Removal is project-scoped: machine-installed bundles remain installed. Use `cube bundle remove` only for machine-level uninstall operations.

The services lock is the commit marker: its presence and currency signals that deployment completed successfully. A missing or stale entry indicates an interrupted deployment.

6.3.2 Tainted services-lock concept

- In rare remove-time I/O failures with partial deletion, the installer can mark `services-lock.json` as tainted (`integrity.state = UNRELIABLE`).
- The installer can also set this marker when lock-source drift is detected (for example context version mismatch between `services-lock.json` and the Cube resolved lock view).
- A tainted services lock means project state may be unstable and requires extra care for subsequent mutating commands.
- Mutating commands still run, but emit warning information indicating the unreliable lock state.

6.3.3 Inspecting the services lock

Use `project-lock-show` to display the full project lock state as preset-compatible machine-readable JSON:

```
cube agentic-ai project-lock-show <project_top_folder>
```

The output includes both context entries and tool dependency entries. For multi-service contexts, selected services are explicitly listed.

6.4 Reproducing a project environment

6.4.1 Committing lock files to source control (recommended)

The simplest and most reliable way to reproduce a project environment is to commit both lock files to your source control repository:

```
.settings/bundles-lock.store.json              ← Cube bundle versions (project top folder)
.cube-ai/services-lock.json                   ← Service selections + deployment target + managed␣
↪artifact ownership
```

On a new machine or after a fresh clone, if you also commit the deployed service artifacts (skills, agents, instructions, prompts, MCP config, routing catalog), only the machine-level bundle installs need restoring:

```
# Restore bundle installs on the machine (reads the Cube project lock)
cube --cwd <project_top_folder> bundle install --project

# Install what the contexts need on this machine (Python packages, tool bundles)
cube agentic-ai doctor <project_top_folder> --fix
```

The project files are already present from the clone, and the Cube wrapper re-installs the exact bundle versions recorded in the lock. The requirements that contexts declare are not in either lock: `doctor --fix` resolves them for this machine and lists the rest, such as VS Code extensions.

Do not commit `.cube-ai/esw-python/`: it records the Python chosen on one machine, which can be an absolute path that another machine does not have. Without it, `doctor --fix` applies the default choice (`auto`). `setup` does not run in a cloned project, which is already deployed.

Bundles excluded from the Cube project lock, including the installer, are not restored by that command. Install them separately when needed. Their deployed project files remain in the clone.

If the deployed artifacts are **not** committed (e.g. they are gitignored), add a second step to redeploy them:

```
# Export a preset from the source project, then replay it
cube agentic-ai project-lock-show <project_top_folder> > my-preset.json
cube agentic-ai project-deploy <project_top_folder> --preset my-preset.json
```

Default preset replay reconciles a context's selected services to its preset bundle version. Add `--preset-no-reconcile` to preserve mixed-version service provenance exactly. Replay is additive: services already present in the destination but absent from the preset are not removed automatically.

`services-lock.json` contains the context bundle versions, service selections, deployment target, and managed artifact ownership. However, it does **not** match the preset file schema and does **not** record tool dependency versions — those are captured only in the Cube project lock. Use `project-lock-show` output as the preset source for `project-deploy --preset`.

6.4.2 When to use `project-lock-show` exported preset instead

If your team works across **different operating systems** or if a context declares a **version range** for one of its tool dependencies (rather than an exact version), re-resolving on a different machine could theoretically pick a different tool version.

In that case, export an explicit preset that locks tool versions verbatim:

```
# On the source machine
cube agentic-ai project-lock-show <project_top_folder> > my-preset.json

# On the target machine
cube agentic-ai project-deploy <new_project_folder> --preset my-preset.json
```

The exported preset adds explicit `"kind": "tool-dependency"` entries with exact versions, so the Cube wrapper installs those specific versions rather than re-resolving from the network.

Preset target fields and deploy behavior:

- If the exported preset contains unambiguous top-level `ide`, `agentic-system`, and `canonical`, `project-deploy --preset` uses them when no explicit target flags are passed.
- If those target fields are missing or ambiguous in the preset, pass explicit target flags (`--ide`, `--agentic-system`, and optionally `--canonical`).
- If target flags are not passed and preset target fields are absent, deployment continues with normal target resolution (`--autodetect-targets enforce` can apply detected values; otherwise ST defaults are used).

For most teams on the same platform, committing both lock files and using `cube bundle install --project` to restore the Cube project lock is the simplest and most reliable path. The exported preset is the right choice when you need a single portable file to share across machines, platforms, or Cube versions.

6.5 Key rules

- **Never edit lock files manually.** All mutations go through installer commands.
- **Commit both lock files** to source control so every team member and CI pipeline can reproduce the environment without re-running interactive prompts.
- **Absence of `services-lock.json`** in a project means no installer deployment has completed yet. Run `project-add` or `project-deploy` to initialise it.

7 Project Housekeeping

This page explains how to maintain and evolve a project deployment over time using:

- *project-add*
- *project-remove*
- *project-deploy*
- *project-deploy-installed*

It focuses on operational choices. It does not repeat conceptual definitions from *Concepts* or full option syntax from *Commands reference*.

Important

These commands deploy and remove contexts. They never check or resolve the requirements that contexts declare (Python packages, tool bundles such as the STM32CubeMX2 application, VS Code extensions). Only `setup` and `doctor` do. After adding, updating or replaying a context, run `cube agentic-ai doctor <project_top_folder> --fix`. An update is a remove, then an add with the same services (`--services all` for a project created with `setup`), then that `doctor --fix`.

Note

Naming convention used below: `bundleName` identifies a context bundle, and `serviceId` identifies a service inside that bundle.

7.1 Why there is no `project-update` command

There is no single `project-update` command by design.

Project evolution is not one workflow. Teams may want to:

- Add a new context without touching existing ones.
- Intentionally keep a mixed-version service state for compatibility reasons.
- Reconcile to a single pinned version from a preset.
- Replay an exact historical mixed-version state.
- Restrict unpinned updates to lock-known versions.

Instead of forcing one policy, the installer provides explicit primitives so users can choose between freedom and control at each step.

7.2 The housekeeping levers

Think in five independent decisions.

Decision	Flexible choice	Controlled choice
What to add or refresh	*project-add* for one context, *project-deploy* for many	*project-deploy-installed* to limit candidates to installed pinned versions
How to change an existing deployment	Additive updates with *project-add* or *project-deploy* (narrower follow-up scope does not remove prior deployment state, so leftovers can remain)	Clean replacement by removing first with *project-remove*, then re-adding or re-deploying; unchanged installer-managed artifacts can also be replayed safely
Removal granularity	Remove a selected service subset (service IDs) with *project-remove* and `--services <serviceId,...>` (more flexible, but can preserve mixed-version leftovers before a later add)	Remove the entire context (`bundleName`) with *project-remove* before re-adding (more controlled, avoids carry-over and mixing risk)
How bundle version is selected for `bundleName`	Unpinned `bundleName` with flexible policy (highest installed eligible version)	Pinned `bundleName@version`, or unpinned `bundleName` with lock-driven policy (reuse project-locked version)
How preset replay behaves	*project-deploy* with `--preset` (reconcile default)	*project-deploy* with `--preset --preset-no-reconcile` (exact per-service replay)

Note

For version selection, pinned references (`bundleName@version`) are always explicit and authoritative. Policy choices apply to unpinned references (`bundleName`).

7.3 Practical scenarios

7.3.1 1) Catalog is behind, but a newer bundle is installed intentionally

You can still deploy newer installed versions.

For this example, assume a catalog entry specifies `1.3.0` for `bundleName=cube-agentic-ai-<connector>`, and both `1.3.0` and `1.4.0` are installed. The current release's catalog uses open-ended minimum ranges instead of exact pins.

- Install or update the context bundle on the machine.
- Deploy with explicit pinning (`name@version`) for deterministic intent.
- Or deploy unpinned (`name`) when you intentionally accept local inventory influence.

This is useful when the installer catalog has not yet been refreshed but your team validated a newer context bundle.

Example:

```
# The catalog entry specifies 1.3.0, but you validated 1.4.0 and deploy it explicitly
cube agentic-ai project-add cube-agentic-ai-<connector>@1.4.0 <project_top_folder>

# Unpinned deploy reuses the highest installed semantic version (1.4.0 here)
cube agentic-ai project-add cube-agentic-ai-<connector> <project_top_folder>
```

Possible command-result excerpt for the unpinned call:

```
{
        "data": {
                "bundleRef": "cube-agentic-ai-<connector>@1.4.0"
```

(continues on next page)

(continued from previous page)

```
        }
}
```

7.3.2 2) Intentionally mix services from different bundle versions

Mixed-version service state is allowed.

- Add or deploy context services over time with different bundle versions.
- Do not remove services unless you want to shrink or reset the state.

This supports gradual migrations where not all services move at once.

Example:

```
# First run unpinned, so current highest selected is 1.3.0
cube agentic-ai project-add cube-agentic-ai-<connector> <project_top_folder> --services␣
↪serviceId-S1,serviceId-S2

# Later, add one service from explicitly pinned newer 1.4.0
cube agentic-ai project-add cube-agentic-ai-<connector>@1.4.0 <project_top_folder> --services␣
↪serviceId-S3
```

Possible lock excerpt:

```
{
        "bundleName": "cube-agentic-ai-<connector>",
        "selectedServiceProvenance": ["serviceId-S1@1.3.0|required", "serviceId-S2@1.3.0|optional
↪", "serviceId-S3@1.4.0|optional"]
}
```

7.3.3 3) Reconcile a preset to a single effective bundle version per context

Default preset deployment (*project-deploy* with `--preset`) is reconcile mode.

- Use this when you want a clean, deterministic target state from preset inputs.
- In preset mode, preset bundle references are pinned and authoritative.

Example:

```
cube agentic-ai project-deploy <project_top_folder> --preset preset.json
```

Preset excerpt:

```
{
        "bundles": [
                {
                        "name": "cube-agentic-ai-<connector>",
                        "version": "1.3.0",
                        "serviceProvenance": ["serviceId-S1@1.3.0|required", "serviceId-S2@1.4.
↪0|optional"],
                        "kind": "connector"
                }
        ]
}
```

In preset JSON, field `name` is the context bundle name, and each `serviceProvenance` entry is `<serviceId>@<bundleVersion>|<required|optional>`. Untagged legacy entries are treated as optional.

Possible reconciled result excerpt:

```
{
        "bundleName": "cube-agentic-ai-<connector>",
        "bundleVersion": "1.3.0",
        "selectedServiceProvenance": ["serviceId-S1@1.3.0|required", "serviceId-S2@1.3.0|optional
↪"]
}
```


In reconcile mode, services selected from the preset are deployed against the preset bundle version (`1.3.0` here).

7.3.4 4) Keep preset diversity exactly as recorded

Use *project-deploy* with `--preset --preset-no-reconcile`.

- The deploy expands execution by `(bundle, version)` groups.
- Services keep their recorded provenance versioning instead of being reconciled.

Use this when historical fidelity matters more than homogenization.

Operational equivalence:

- `project-deploy --preset --preset-no-reconcile` is equivalent to running one pinned *project-add* per `(bundle, version)` group from `serviceProvenance`.

Example:

```
cube agentic-ai project-deploy <project_top_folder> --preset preset.json --preset-no-reconcile
```

Equivalent grouped execution excerpt:

```
project-add cube-agentic-ai-<connector>@1.3.0 --services serviceId-S1
project-add cube-agentic-ai-<connector>@1.4.0 --services serviceId-S2
```

7.3.5 5) Keep unpinned updates under lock control

Use lock-driven policy for unpinned references.

- With unpinned names, lock-driven reuses the bundle version already locked in the project when available (for example `1.3.0` from the earlier unpinned add).
- With explicit `name@version`, the explicit pin remains authoritative.

This is a good middle ground between strict pinning and full flexibility.

Example:

```
cube agentic-ai project-add cube-agentic-ai-<connector> <project_top_folder> --bundle-version-
↪policy lock-driven
cube agentic-ai project-deploy <project_top_folder> --mode all --bundle-version-policy lock-
↪driven
```

Possible nominal result excerpt:


```
{
        "ok": true,
        "data": {
                "bundleRef": "cube-agentic-ai-<connector>@1.3.0"
        }
}
```

Note

If the required locked version is not installed locally, the command fails with `BUNDLE_LOCKED_VERSION_NOT_INSTALLED`. In this example, the missing locked version would be `cube-agentic-ai-<connector>@1.3.0`.

7.4 Connector upgrades and tool dependencies

Connector bundles and their Cube tool dependencies have separate scopes. `machine-bootstrap` upgrades catalog connectors on the machine. It does not refresh tool versions already selected by an existing project lock. In this section, *workspace* means the project top folder targeted by the command (`<project_top_folder>`). Both terms refer to the same project-level lock and deployment scope. A fresh workspace resolves the newest installed tool version that satisfies the connector's dependency range, while an existing workspace keeps its locked version for reproducibility.

`project-add`, `project-deploy`, and `project-deploy-installed` report each tool dependency's project-lock status (`ADDED`, `UPDATED`, `UNCHANGED`, or `REMOVED`) and machine status (`INSTALLED` or `REUSED`). If a newer installed version satisfies the dependency range but is not selected because of the project lock, the command adds `TOOL_DEPENDENCY_PINNED_BY_PROJECT_LOCK` to `data.warnings` and the human-readable refresh hint to `data.warningMessages`. To refresh tool dependencies intentionally, run `cube bundle update --project` from the project top folder. Review and retain the resulting lock changes.

7.5 Recommended operating policy

Default policy for updates:

1. Remove the entire context first with *project-remove*.
2. Re-add or redeploy it with the intended context version and service set.

This full remove + full add pattern is the safest default because it minimizes leftover artifacts from additive operations.

Preset replay is idempotent for unchanged installer-managed artifacts. The installer may replace an existing artifact when its recorded owner is the same logical context service/artifact and its content hash still matches the services lock. User-owned, cross-context, duplicate, or same-version modified managed files remain conflicts and must be resolved before deployment.

Advanced policy (intentional mixed-version control):

1. Use partial service operations only when you explicitly need mixed-version service granularity.
2. After each operation, verify the repository and lock state carefully (`services-lock.json`, generated artifacts, and target folders).

Important

Canonical deployments require extra care in this policy.

- With canonical deployment (`--canonical`), installer-managed paths are recorded in the lock at deployment time.
- If an agentic system (or any post-process) reshuffles files afterward, those lock paths are no longer the real final locations unless the reshuffling owner also updates lock metadata.
- Without that lock rework, later remove operations can target outdated canonical paths, and manual cleanup may still be required for moved files.

For machine-constrained rollout, prefer *project-deploy-installed*.

7.6 Related pages

- *Usage* for concise end-to-end command sequences.
- *Commands reference* for complete option and status details.
- *Lock files* for lock-state interpretation and reproducibility.
- *Concepts* for core terminology and model boundaries.

8 Usage

Workflows ordered from full onboarding to targeted maintenance operations.
Replace `<project_top_folder>` with the absolute path to your project top folder.

Important behavior for all deployment commands:

- `project-add` and `project-deploy` are additive.
- Re-deploying a smaller service subset does not remove previously deployed services.
- Use `project-remove` when you need to remove services or reset a context deployment before reapplying it.
- `project-add` and `project-deploy` accept `--bundle-version-policy flexible|lock-driven` (default: `flexible`).
- With `--bundle-version-policy flexible`, when no exact version is specified and several installed versions match, `project-add` and `project-deploy` reuse the highest installed semantic version that satisfies the requested bundle reference.
- With `--bundle-version-policy lock-driven`, `project-add` and `project-deploy` reuse the version already recorded in the project lock for that bundle. For a bundle not yet locked, they fall back to flexible selection. A locked version that is missing locally causes `BUNDLE_LOCKED_VERSION_NOT_INSTALLED`.
- In `project-deploy --preset` mode, bundle references are already pinned by the preset, so `--bundle-version-policy` has no operational effect.
- `project-deploy-installed` does not expose `--bundle-version-policy` because its candidates are already pinned by construction (`name@version` from installed inventory).

8.1 0. Quick start: set up the toolkit in one command

Use `setup` for the standard journey. It installs the contexts on the machine, resolves what they need, creates the project folder, deploys into it, and reports what is left to do. Read `data.nextSteps` in the result. `setup` works only for a project folder that has never been deployed: it refuses a folder that has a services lock (`SETUP_PROJECT_ALREADY_DEPLOYED`). In a deployed project, use the workflows of sections 5 to 8.

```
# Preview first: nothing is changed. Requirements of bundles not installed yet cannot be shown.
cube agentic-ai setup --dry-run

# Install and deploy into the default folder <home>/my-agentic-ai-project
cube agentic-ai setup

# Choose the project folder, and a dedicated Python environment for the Python packages
cube agentic-ai setup --project-folder <project_top_folder> --python venv

# Install only some contexts
cube agentic-ai setup --contexts cube-agentic-ai-st-docs,cube-agentic-ai-sidekick-kb --project-
↪folder <project_top_folder>

# Also upgrade the installed contexts that have a newer version in the catalog range
cube agentic-ai setup --upgrade-installed-in-range

# Let the installer run `code` to install the VS Code extensions that the contexts declare (VS␣
↪Code target only)
cube agentic-ai setup --install-vscode-extensions --vscode-profile "My profile"
```

When `data.readiness` is not `READY`, or when something fails later, check the installation:

```
# Read-only diagnosis of the contexts deployed in the project
cube agentic-ai doctor <project_top_folder>

# Resolve the unmet dependencies, then check again
cube agentic-ai doctor <project_top_folder> --fix
```

The workflows below use the building blocks directly: pinned versions, selected services, presets, upgrades and removal.

8.2 1. Recommended: full onboarding on a fresh machine

Use this two-command flow for first-time setup on a new machine, then interactive deployment into a project. Required Cube tool dependency bundles are installed automatically.

```
# Install all STM32Cube Agentic Context bundles on the machine (first time on a new machine).
# Required Cube tool dependencies are installed automatically.
cube agentic-ai machine-bootstrap

# Deploy the selected contexts into your project.
# The command proposes each candidate. Accept or skip it.
# Missing contexts and their required Cube tools are installed automatically.
cube agentic-ai project-deploy <project_top_folder> --mode interactive
```

8.3 2. One-shot non-interactive onboarding (CI or scripted)

```
# Install baseline
cube agentic-ai machine-bootstrap

# Deploy all catalog bundles without prompts, using default service selections
cube agentic-ai project-deploy <project_top_folder> --mode all --services-mode default
```

`machine-bootstrap` defaults to `--upgrade-installed-in-range false`: missing bundles are installed, but already-installed satisfying bundles are not proactively upgraded.

Use `--upgrade-installed-in-range true` when you intentionally want to refresh installed in-scope bundles to newer eligible versions as part of bootstrap:

```
# Install missing bundles and also upgrade already-installed in-scope bundles when eligible
cube agentic-ai machine-bootstrap --upgrade-installed-in-range true
```

8.4 3. Reproducible deployment from a preset file

Use a preset when you need a fully scripted, version-locked deployment (CI pipelines, team onboarding).

```
# preset.json pins exact context bundle versions and service provenance
# {
#   "bundles": [
#     { "name": "cube-agentic-ai-esw", "version": "<context-version>", "serviceProvenance": ["S1@
↪<context-version>|required"], "kind": "connector" },
#     { "name": "cube-agentic-ai-st-docs", "version": "<context-version>", "serviceProvenance": [
```

(continues on next page)

(continued from previous page)

```
↪"S1@<context-version>|required"], "kind": "connector" },
#     { "name": "tool-a", "version": "<tool-version>", "kind": "tool-dependency" }
#   ]
# }

cube agentic-ai project-deploy <project_top_folder> --preset preset.json
```

No prompts are shown. Context service selection is derived from `serviceProvenance` in the preset.

Target selection in preset mode follows strict priority (CLI flag > preset field > autodetect enforce > ST default):

- When passed, `--ide`, `--agentic-system`, and `--canonical` always take precedence over preset target fields.
- When not passed, preset top-level `ide`, `agentic-system`, and `canonical` are used if present.
- If neither CLI flags nor preset fields provide a dimension, `--autodetect-targets enforce` can resolve it. Otherwise the ST default applies (`vscode`, `github-copilot`).

```
# Explicit target override while using a preset
cube agentic-ai project-deploy <project_top_folder> --preset preset.json \
  --ide vscode --agentic-system github-copilot
```

8.5 4. Reproduce an existing project environment

Capture the exact deployment of a working project and replay it elsewhere.

```
# On the reference machine: export the current deployment as a preset file
cube agentic-ai project-lock-show <project_top_folder> > my-preset.json

# On the target machine (or target project): replay the recorded service versions
cube agentic-ai project-deploy <project_top_folder> --preset my-preset.json --preset-no-reconcile
```

Use `--preset-no-reconcile` when services from one context were deployed from different versions. Without that option, preset replay reconciles each context's selected services to its preset bundle version:

```
cube agentic-ai project-deploy <project_top_folder> --preset my-preset.json
```

The output of `project-lock-show` is a valid preset file for `project-deploy --preset`.

8.6 5. Add a single context to a project

```
# Install the context bundle on the machine (no-op if already installed)
cube agentic-ai machine-install cube-agentic-ai-esw

# Deploy it into the project with interactive service selection
cube agentic-ai project-add cube-agentic-ai-esw <project_top_folder> --interactive

# Install what the context needs (Python packages, tool bundles): project-add does not
cube agentic-ai doctor <project_top_folder> --bundles cube-agentic-ai-esw --fix
```

Omit `--interactive` to deploy the default and required services without prompting.

8.7 6. Add a context with explicit service selection

```
# Deploy only services S1 and S2 of a multi-service bundle
cube agentic-ai project-add cube-agentic-ai-esw <project_top_folder> --services S1,S2

# Install what these services need
cube agentic-ai doctor <project_top_folder> --bundles cube-agentic-ai-esw --fix
```

If the same context was already deployed with other services, this command adds `S1,S2` to the existing selected set. It does not remove previously deployed services.

8.8 7. Update a deployed context to a newer version (upgrade)

Use explicit remove then re-add when you want a clean transition to a new context version and service set.

```
# Remove all deployed services for the context from the project
cube agentic-ai project-remove cube-agentic-ai-esw <project_top_folder>

# Install and deploy the selected version
cube agentic-ai project-add cube-agentic-ai-esw@<target-version> <project_top_folder> --services␣
↪all

# Install what the new version needs. For ESW, its Python environment follows the project's␣
↪choice.
cube agentic-ai doctor <project_top_folder> --bundles cube-agentic-ai-esw --fix
```

Use `--services all` for a project created with `setup`, which deploys every service. Otherwise pass the service IDs that `project-lock-show` lists. Add the `--ide` and `--agentic-system` options of the project when it does not use the default target. You can also use `project-deploy --preset` to apply a fully version-locked upgrade plan, followed by the same `doctor --fix`.

For eSW in a project that uses `--python venv`, the `doctor --fix` step creates the environment of the new version: see *Python of the eSW context*.

8.9 8. Roll back to an older context version (downgrade)

```
# Remove the current context deployment
cube agentic-ai project-remove cube-agentic-ai-esw <project_top_folder>

# Reinstall/reuse and deploy an older pinned version
cube agentic-ai project-add cube-agentic-ai-esw@<target-version> <project_top_folder> --services␣
↪S1,S2

# Install what that version needs
cube agentic-ai doctor <project_top_folder> --bundles cube-agentic-ai-esw --fix
```

This explicit remove + add sequence avoids mixing artifacts from different deployed versions.

8.10 9. Two-step workflow: install first, deploy later

Useful when machine setup and project setup are done by different people or at different times.

```
# --- Machine setup (done once, by the operator) ---
cube agentic-ai machine-bootstrap

# --- Project setup (done per project) ---
# Deploy only from what is already installed. No downloads are attempted.
cube agentic-ai project-deploy-installed <project_top_folder> --mode interactive

# Install what the deployed contexts need. This step may download dependencies.
cube agentic-ai doctor <project_top_folder> --fix
```

8.11 10. Deploy to a specific IDE and agentic system

```
# Default: VS Code + GitHub Copilot
cube agentic-ai project-deploy <project_top_folder> --mode all

# No IDE binding, Claude Code as agentic AI system
cube agentic-ai project-deploy <project_top_folder> --mode all \
  --ide none --agentic-system claude-code

# Canonical layout only (no IDE-specific placement)
cube agentic-ai project-deploy <project_top_folder> --mode all --canonical
```

When you use `--canonical`, it takes priority over `--ide` and `--agentic-system` placement. In other words, deployment is written only to `.cube-ai/` (portable layout), not to native IDE/agentic folders. If an agentic system later reshuffles those files, `project-remove` still targets the canonical paths recorded in the lock. Moved files require manual cleanup.

For a compact explanation of the two target dimensions and the common deployment patterns, see *Target selection*.

The tested pairings are VS Code + GitHub Copilot and no IDE binding + Claude Code. Other accepted target IDs are not tested. Run `cube agentic-ai list-supported-targets` to inspect the supported targets of your installer release. No target is production-validated.

8.12 11. Inspect what is installed and deployed

```
# What is installed on the machine?
cube agentic-ai machine-discovery

# Show only AI-relevant bundles (exclude OUT_OF_SCOPE)
cube agentic-ai machine-discovery --ai-only

# Same discovery but without generating machine-catalog.json
cube agentic-ai machine-discovery --no-json

# Generate machine catalog JSON at a custom path
cube agentic-ai machine-discovery --json-file ./artifacts/machine-catalog.json

# What is locked in a specific project?
cube agentic-ai project-lock-show <project_top_folder>
```

Notes:

- `machine-discovery` always returns command-result JSON on stdout.
- By default it also writes `machine-catalog.json` containing only the discovered bundle list array.
- `--ai-only` filters out bundles classified as `OUT_OF_SCOPE`.
- `--json-file` requires JSON file generation to be enabled. Combining `--json-file` with `--no-json` fails with `INVALID_OPTION_COMBINATION`.

8.13 12. Dry run before committing

```
# Preview what machine-bootstrap would install
cube agentic-ai machine-bootstrap --dry-run

# Preview what project-deploy would deploy (all mode - no prompts)
cube agentic-ai project-deploy <project_top_folder> --mode all --dry-run

# Preview with interactive selection (prompts shown, nothing deployed)
# Accepted bundles appear as SKIPPED_DRY_RUN in the result
cube agentic-ai project-deploy <project_top_folder> --mode interactive --dry-run
```

For `machine-bootstrap --dry-run`, entries that would change an already-installed bundle are reported as `WOULD_UPDATE`. Entries missing on the machine are reported as `WOULD_INSTALL`.

8.14 13. Remove a context from a project

```
# Remove all deployed services of a context (bundle entry removed from project)
cube agentic-ai project-remove cube-agentic-ai-esw <project_top_folder>

# Remove only specific services (other services remain deployed)
cube agentic-ai project-remove cube-agentic-ai-esw <project_top_folder> --services S3,S4

# Preview what would be removed before committing
cube agentic-ai project-remove cube-agentic-ai-esw <project_top_folder> --services S3 --dry-run
```

Use `project-remove` before reapplying a smaller service set if your intent is to replace rather than extend the current deployment.

Important for `project-remove`:

- Removal decisions come from `<project_top_folder>/.cube-ai/services-lock.json`.
- Services or files added manually outside installer deployment are not tracked and are not removed.
- If deployment used `--canonical` and files were later reshuffled to non-canonical paths, `project-remove` still targets canonical-managed paths. Moved files may need manual cleanup.
- The command does not perform lock-versus-manifest consistency verification before removing artifacts.
- The command is project-scoped. It does not uninstall bundles from machine inventory.
- After partial delete I/O failures, the services lock may be marked tainted (`integrity.state = UNRELIABLE`). In that state, run a dry-run first before any further mutating command.

Risk reduction recommendation:

- For `project-remove`, run `--dry-run` first as a general safety step. Dry-run executes preflight checks (candidate/path validation) without changing project state.

8.15 14. Access the user manual

```
# Default output is TLDR quick reference on stdout
cube agentic-ai get-user-manual

# Explicit TLDR request
cube agentic-ai get-user-manual --format tldr

# Get the Markdown file paths so an AI agent can read the manual
cube agentic-ai get-user-manual --format markdown

# Open the HTML version in a browser (human use)
cube agentic-ai get-user-manual --format html
```

9 Troubleshooting

9.1 Diagnostics and tips

By default, `setup` and `doctor` show progress on stderr. Other commands are silent. Explicit `--verbose none` silences progress and diagnostics. Non-interactive execution writes only its JSON result to stdout. Interactive commands also print prompts, and help/version output is plain text. Pass `--verbose <level>` to activate diagnostic output on stderr.

```
cube agentic-ai --verbose standard project-add <bundle> <project>   # step-by-step progress
cube agentic-ai --verbose debug   project-add <bundle> <project>   # full sub-step trace
```

Level	What appears on stderr
`none`	Nothing. Default for commands other than `setup` and `doctor`.
`progress`	Default for `setup` and `doctor`; also enables per-item progress for other commands.
`standard`	Progress plus INFO, ERROR, and CRITICAL messages
`debug`	All of the above plus resolved paths, per-file copy trace, raw values

`--verbose` is a root-level flag. Example:

```
# Diagnose a failing project-add
cube agentic-ai --verbose debug project-add cube-agentic-ai-programmer /my/project 2>debug.log
```

Both stdout and stderr can be captured independently:

```
cube agentic-ai --verbose standard project-add <bundle> <project> > result.json 2>progress.log
```

On failure, every command writes a JSON object with `ok: false` and a structured `error` block to stdout. The exit code is non-zero (see *Commands → Output envelope*).

Other useful practices:

- Use `--dry-run` with `machine-bootstrap`, `project-deploy`, `project-deploy-installed`, or `project-remove` to preview changes. `machine-install` and `project-add` do not have this option.
- If a command hangs, `cube` may be waiting on a network operation — check connectivity and retry.
- After a partial `project-remove` failure, or when lock-source drift is detected, `services-lock.json` may be marked tainted (`integrity.state = UNRELIABLE`): inspect the actual project state. See *Lock files* for details.
- If command output includes warning code `CONNECTOR_VERSION_MISMATCH`, run `cube agentic-ai project-lock-show <project>` and inspect lock state before the next mutating operation.

9.2 Error codes

Codes specific to one command are noted in the **Meaning** line.

9.2.1 ADVISORY_ONLY

Meaning: `--autodetect-targets advisory` found missing target dimensions, reported the detected candidates, and intentionally stopped before installation or deployment. This is expected behavior, not an error.

Action: Review the detected `ide` and `agentic-system` candidates in the JSON output, then rerun with explicit `--ide` and `--agentic-system` values, or switch to `--autodetect-targets enforce` if you want the installer to apply detected values automatically.

9.2.2 ALL_BUNDLES_FAILED

Meaning: (`setup`, `machine-bootstrap`, `project-deploy`, `project-deploy-installed`) All accepted or processed bundles were attempted and every one failed. No contexts were deployed. Setup stops before its project-folder creation step. Per-bundle causes are in `error.details.results` for machine bootstrap/setup or `error.details.failedBundles` for deploy commands.

Action:

1. Run with `--verbose debug` to see the per-bundle error details.
2. For project deploy commands, inspect `data.bundles[].error.code` when the command returns a partial success, or `error.details.failedBundles` when all bundles failed.
3. For `machine-bootstrap` or `setup`, inspect `data.results[]` on partial success, or `error.details.results` when all bundles failed.
4. Fix the root cause for each bundle and retry the same command. After `setup` returns `PARTIAL`, use the project-scoped `project-add` and `doctor --fix` commands in its `data.nextSteps`. Setup cannot be rerun once it has deployed contexts.

9.2.3 ARTIFACT_DUPLICATE_DESTINATION

Meaning: Two incoming artifacts resolve to the same target path or skill directory.

Action: Review the reported owners and destination, then rename or move the user-owned file, change the selected deployment, or skip one of the conflicting artifacts before retrying.

9.2.4 ARTIFACT_USER_OWNED_CONFLICT

Meaning: An existing destination is not recorded as installer-managed and is treated as user-owned or unknown.

Action: Rename or move the file, or change the selected deployment before retrying. The installer does not silently overwrite user files.

9.2.5 ARTIFACT_CONNECTOR_SERVICE_CONFLICT

Meaning: The destination is already managed by another service or artifact from the same context.

Action: Fix the context artifact mapping or change the selected deployment. The installer does not allow one service or artifact to replace another within the same context.

9.2.6 ARTIFACT_CROSS_CONNECTOR_CONFLICT

Meaning: The destination is recorded as managed by a different context.

Action: Keep the existing context, remove it before deploying the new context, or change the incoming artifact destination. The installer does not allow one context to replace another context's managed artifact. Report this conflict to the bundle owner if it is unexpected.

9.2.7 ARTIFACT_CONTENT_HASH_MISMATCH

Meaning: The destination belongs to the incoming logical artifact, the incoming service version is the same as the recorded version, but its current contents no longer match the hash recorded when the installer last deployed it.

Action: Review the file for intentional user changes. Preserve or move those changes, then retry deployment. A different version of the same logical service is an intentional upgrade and is not reported with this error.

9.2.8 SERVICES_LOCK_SCHEMA_UNSUPPORTED

Meaning: The project services lock does not use the hash-backed schema required for ownership-aware replay.

Action: Perform a clean deployment that creates the current services-lock schema, then retry the replay. The installer does not infer ownership from legacy path-only records.

9.2.9 PROJECT_REMOVE_REQUIRED_SERVICE_CONFIRMATION_REQUIRED

Meaning: A partial `project-remove` request includes a service recorded as required, while another service from the same context will remain deployed. Removing it may impede the remaining services.

Action: Confirm the interactive prompt, or rerun the command with `--confirm-always`. Removing the whole context, or removing all deployed services, does not require confirmation. Legacy lock entries without a required/optional tag are treated as optional.

9.2.10 ARTIFACT_ENTRYPOINT_MISSING

Meaning: A selected skill, agent, instruction, or prompt source/entrypoint is missing.

Action: Check the bundle source path and selected `fileFamilies` entry.

9.2.11 ARTIFACT_METADATA_INVALID

Meaning: Required kind-specific metadata cannot be validated or normalized.

Action: Inspect the artifact frontmatter and target profile requirements.

9.2.12 ARTIFACT_KIND_UNDEFINED

Meaning: The context bundle contains inconsistent artifact metadata. The installer cannot determine a valid semantic kind for one of the declared artifacts.

Action: Reinstall or update the context bundle. If the problem persists, report the bundle name, version, and error details to the bundle owner. Bundle metadata must use supported kinds (`skill`, `agent`, `instruction`, `prompt`, or `resource`) consistently.

9.2.13 ARTIFACT_NAME_INVALID

Meaning: An artifact identity does not satisfy the registered or host naming rules.

Action: Check `artifacts.name` and the selected target profile.

9.2.14 ARTIFACT_REFERENCE_KIND_MISMATCH

Meaning: A `skill://`, `agent://`, `instruction://`, or `prompt://` reference does not match the declared artifact kind.

Action: This indicates inconsistent bundle metadata or `capabilities.md` content. Reinstall or update the context bundle and report the bundle name, version, reference, and error details to the bundle owner.

9.2.15 ARTIFACT_REFERENCE_MISSING

Meaning: A selected artifact is not referenced in the filtered `capabilities.md` content for the deployed service set.

Action: Check that the selected service's `capabilities.md` section references its declared artifact with the matching `skill://`, `agent://`, `instruction://`, or `prompt://` scheme. Report the bundle name, service, and artifact identity to the bundle owner if the reference is missing from the context content.

9.2.16 ARTIFACT_REFERENCE_UNKNOWN

Meaning: A semantic artifact reference has no corresponding entry in the installer-provided replacement map.

Action: This indicates an incomplete or inconsistent context bundle. Reinstall or update the bundle and report the bundle name, version, reference, and error details to the bundle owner. It is not a user-file naming issue.

9.2.17 ARTIFACT_RESOURCE_OWNS_RESOURCE

Meaning: A context bundle declares a `kind: "resource"` artifact with its own non-empty `resources` list. Resource artifacts are supporting files and cannot own other resources.

Action: Reinstall or update the context bundle. If the problem persists, report the bundle name, version, artifact identity, source path, and nested resource list to the bundle owner. The bundle owner must move those nested resources to the owning `skill`, `agent`, `instruction`, or `prompt` artifact's `resources` list.

9.2.18 BUNDLE_INSTALL_FAILED

Meaning: `cube` could not install one or more bundles (network error, registry unavailable, or version not found).

Action:

1. Check network connectivity.
2. Verify the bundle name and version: `cube agentic-ai machine-discovery` lists what is available locally. The command `cube bundle list-online` lists the registry.
3. Retry. If the error persists, check the Cube registry status.

9.2.19 BUNDLE_LOCKED_VERSION_NOT_INSTALLED

Meaning: (`project-add`, `project-deploy`) With `--bundle-version-policy lock-driven`, the command requires the version already locked for the bundle, but that exact version is not installed on the machine.

> **Note**
>
> `project-deploy-installed` does not accept `--bundle-version-policy`. Its candidates are already pinned (`name@version` from installed inventory).

Action:

1. Inspect project lock state with `cube agentic-ai project-lock-show <project>` and identify the required bundle version.
2. Install the missing version (`cube bundle install <bundle>@<version>`), then rerun the command.
3. If you intentionally want to proceed without the locked version, rerun with `--bundle-version-policy flexible`.

9.2.20 BUNDLE_NOT_DEPLOYED

Meaning: (`project-remove`) The bundle is not deployed in the project.

Action: Run `cube agentic-ai project-lock-show <project>` to see what is currently deployed, then retry with a valid bundle name.

9.2.21 BUNDLE_NOT_FOUND

Meaning: A requested bundle is not installed on the machine and could not be located.

Action: Verify the bundle name with `cube agentic-ai show-catalog`, then run `cube agentic-ai machine-install <bundle_name>`, or use `cube agentic-ai project-deploy` with `--auto-install-missing` (default `true`).

9.2.22 BUNDLE_NOT_IN_AI_CATALOG

Meaning: (`machine-install`, `setup --contexts`, `doctor --bundles`) The bundle name is not a supported STM32Cube Agentic Context bundle.

Action: Run `cube agentic-ai show-catalog` to see supported contexts. Use `cube agentic-ai machine-discovery` to see what is already installed. To install another kind of bundle, use `cube bundle install <bundle_name>` directly.

9.2.23 CATALOG_INVALID

Meaning: (`show-catalog`, `machine-bootstrap`, `machine-install`, `machine-discovery`, `project-add`, `project-deploy`) Context catalog data is invalid.

Action: Reinstall the installer bundle or restore a valid catalog version.

9.2.24 CATALOG_NOT_FOUND

Meaning: (`show-catalog`, `machine-bootstrap`, `machine-install`, `machine-discovery`, `project-add`, `project-deploy`) The context catalog is unavailable.

Action: Reinstall the installer bundle and retry.

9.2.25 CATALOG_PARSE_ERROR

Meaning: (`show-catalog`, `machine-bootstrap`, `machine-install`, `machine-discovery`, `project-add`, `project-deploy`) The context catalog could not be read.

Action: Reinstall the installer bundle, or fix the malformed catalog file if working in a development checkout.

9.2.26 CATALOG_VALIDATION_FAILED

Meaning: A catalog file under `data/` failed schema validation (missing field, wrong type, or unknown version). Indicates a corrupted or incompatible installer bundle.

Action: Re-install the installer bundle:

```
cube bundle install cube-agentic-ai-installer --yes
```

9.2.27 CUBE_BUNDLE_INFO_ERROR

Meaning: (`machine-discovery`) The installer could not inspect an installed context's Cube tool dependencies.

Action: Reinstall the affected context bundle and retry. If the issue persists, run with `--verbose debug` and report the bundle name and error details to support.

9.2.28 CUBE_LIST_ERROR

Meaning: (`machine-discovery`) `cube bundle list` failed while reading installed machine inventory.

Action: Verify `cube` is installed and functional, then retry. Use `cube agentic-ai --verbose debug machine-discovery` for details.

9.2.29 CUBE_NOT_FOUND

Meaning: The `cube` binary is not on `PATH`.

Action: Complete the bootstrap first: https://dev.st.com/stm32cube-agentic-ai/stm32cube-agentic-ai-start/index.html On Windows, open a new terminal after installation so the updated `PATH` takes effect.

9.2.30 INVALID_OPTION_COMBINATION

Meaning: (`machine-discovery`) `--json-file` was passed while `--no-json` disabled machine catalog file generation.

Action: Remove `--json-file`, or re-enable file generation by omitting `--no-json`.

9.2.31 INVALID_USAGE

Meaning: The CLI arguments or options are invalid, for example an unknown command, an invalid option value, a missing required argument, or an empty `machine-discovery --json-file` value.

Action: Read `error.message` and run `cube agentic-ai <command> --help` to check the syntax. For `machine-discovery --json-file`, provide a non-empty path value.

9.2.32 JSON_FILE_WRITE_FAILED

Meaning: (`machine-discovery`) Machine catalog file output could not be written.

Action: Verify target folder write permissions, ensure parent directories can be created, or choose another `--json-file` path.

9.2.33 LEGACY_CUBE_PROJECT_LOCK_UNSUPPORTED

Meaning: (`project-add`, `project-remove`, `project-lock-show`) The project still has a Cube bundle lock created by an older installer version under `<canonicalRoot>/.settings/bundles.store.json` (e.g. `.cube-ai/.settings/`), instead of the project's shared `.settings/` folder. This legacy layout is no longer supported and the command is rejected without modifying any lock file.

Action:

1. Inspect the legacy lock at the path given in `error.cause`/`error.details.legacyLockPath` and the project's shared `.settings/bundles.store.json` (if any).
2. Reconcile the bundle references into the shared `.settings/bundles.store.json` (or remove the legacy folder if its references are already reflected there).
3. Rerun the command.

9.2.34 LIST_SUPPORTED_TARGETS_FAILED

Meaning: (`list-supported-targets`) Unclassified failure while reading the supported deployment targets.

Action: Rerun with `--verbose debug`, capture stdout/stderr, and report the failure with the emitted `error.cause`.

9.2.35 LOCK_READ_ERROR

Meaning: The project services lock file (`.cube-ai/services-lock.json`) is present but cannot be read or parsed.

Action: Back up and inspect the file, then restore a valid copy from source control when available. Do not delete it blindly: it records managed-file ownership required for safe replay and removal.

9.2.36 MACHINE_DISCOVERY_FAILED

Meaning: (`machine-discovery`) Unclassified failure.

Action: Rerun with `--verbose debug`, capture stdout/stderr, and report the failure with the emitted `error.cause` and `error.details`.

9.2.37 MANUAL_NOT_FOUND

Meaning: (`get-user-manual`) Documentation files could not be located in the installed bundle.

Action: Reinstall the installer bundle to restore its embedded manual. When working from a source checkout, regenerate the HTML and bundled Markdown through the supported synchronization script:

```
# from the installer source
bash update-user-manual.sh
```

9.2.38 MCP_CONFIG_PARSE_FAILED

Meaning: A bundle's MCP configuration file or project's MCP manifest could not be parsed as JSON/JSONC.

Action:

1. The file may have syntax errors (missing commas, unescaped quotes, invalid comments).
2. Check the file path in the error details for the exact location.
3. For VS Code workspace manifests (`.vscode/mcp.json`), ensure the server map is valid JSON under top-level `servers`.
4. If you copied a `settings.json`-style wrapper (`{"mcp":{"servers":...}}`) into `.vscode/mcp.json`, rewrite it as top-level `servers`.
5. Fix the JSON syntax and retry, or delete the file to reset it.
6. If the error is in a bundle-provided file, the bundle installation may be corrupted — re-install it.

9.2.39 MCP_FORMAT_INCOMPATIBLE

Meaning: MCP configuration files use incompatible formats or structure (e.g., source is `.json` but target is `.yaml`, or the servers key is not found).

Action:

1. Ensure both the bundle MCP file and project MCP manifest use the same file format.
2. Ensure the MCP payload contains a recognized server map: `servers`, `mcpServers`, or `mcp.servers`.
3. When the project MCP manifest already exists, the installer preserves that existing key path and merges bundle servers into it.
4. When the project MCP manifest is created, the installer uses the selected target's required key path (for example `servers` for VS Code or `mcpServers` for Claude Code).
5. A sidecar file (`mcp-st<ext>`) may have been created next to the target for manual inspection.

9.2.40 MCP_MERGE_CONFLICT

Meaning: The bundle tries to define a server with the same name as an existing server in the project, but the configurations differ.

Action:

1. This protects your project from losing custom server configurations.
2. A sidecar file (`mcp-st<ext>`) has been created next to your project's MCP manifest.
3. Manually merge the two files, keeping the configuration that is correct for your project.
4. Delete the sidecar file after merging.
5. Retry the deploy command.

Note

The installer normalizes MCP shape only when it creates a missing target manifest. Existing project manifests keep their detected key path so user-managed MCP files are not migrated during a merge.

9.2.41 MCP_SERIALIZATION_FAILED

Meaning: The merged MCP configuration could not be serialized back to JSON. This is an internal error and should not occur in normal operation.

Action: Check the error details and report it as a bug.

9.2.42 NO_SUPPORTED_TARGET

Meaning: (`list-supported-targets`) The target matrix catalog flags no supported IDE or agentic-system target.

Action: Re-install the installer bundle to restore a valid `data/deployment-targets.json`.

9.2.43 PATH_TRAVERSAL_DETECTED

Meaning: A source or destination path escapes its permitted bundle or project root.

Action: Remove path traversal segments and retry.

9.2.44 PRESET_INVALID

Meaning: (`project-deploy`) The preset file is not valid JSON, does not satisfy the expected schema, or contains a service provenance entry that cannot be replayed. A valid tagged entry uses `<serviceId>@<bundleVersion>|required` or `<serviceId>@<bundleVersion>|optional`. A legacy entry without the tag remains valid and is treated as optional.

Action: Inspect the file for JSON syntax errors and malformed service provenance. If the preset was generated by `project-lock-show`, re-export it because it may have been manually edited. Each context entry must have `name`, `version`, and `serviceProvenance`. Each provenance entry must contain a usable service ID and bundle version. Tool-dependency entries must not include `serviceProvenance`.

9.2.45 PRESET_NOT_FOUND

Meaning: (`project-deploy`) The path passed to `--preset` does not exist or is not a file.

Action: Verify the path. To generate a valid preset from an existing project, use `cube agentic-ai project-lock-show <project> > preset.json`.

9.2.46 PROJECT_DEPLOY_INVALID_OPTIONS

Meaning: (`project-deploy`) An invalid option combination was provided. This currently applies when `--preset-no-reconcile` is used without `--preset`.

Action:

1. Provide a preset file with `--preset <config_file>` when using `--preset-no-reconcile`.
2. Or remove `--preset-no-reconcile` and run standard deploy mode.

9.2.47 PROJECT_NOT_FOUND

Meaning: The provided `project_top_folder` does not exist or is not a directory.

Action: Verify the path or create the directory first. Generated firmware and Cube project metadata are not required before deploying agentic contexts.

9.2.48 PROJECT_REMOVE_FAILED

Meaning: (`project-remove`) Removal could not be completed due to a generic internal failure.

Action: Re-run with `--verbose debug`, then follow the more specific error code if one is provided.

9.2.49 PROJECT_REMOVE_IO_FAILED

Meaning: (`project-remove`) A filesystem I/O failure occurred during deletion of managed paths (for example lock, permission change, or other runtime FS error).

Partial deletion may already have occurred before the failure.

Action:

1. Re-run with `--verbose debug` and inspect `failedPath`, `deletedPaths`, and `pendingPaths` in error details.
2. Resolve the failing path condition (permissions, file lock, path state).
3. Re-run `project-remove`.
4. If needed, manually reconcile partially removed files using the reported path lists.

9.2.50 PROJECT_REMOVE_MANAGED_PATHS_MISSING

Meaning: (`project-remove`) One or more managed files expected by the computed removal plan are not present at their managed paths.

This can happen after canonical deployment when files were manually reshuffled from `.cube-ai/` to non-canonical locations.

Action:

1. Re-run with `--verbose debug` to inspect the computed removal paths.
2. Check whether files were moved after deployment (especially after `--canonical` runs).
3. Move files back to the expected managed paths.
4. Re-run `project-remove`.

9.2.51 PROJECT_REMOVE_PERMISSION_PRECHECK_FAILED

Meaning: (`project-remove`) Preflight detected insufficient permissions for one or more managed removal targets.

Action:

1. Re-run with `--verbose debug` and inspect the failing path details.
2. Check file and directory permissions/ACLs for the managed targets.
3. Close processes that may lock files.
4. Fix permissions and re-run `project-remove`.

9.2.52 ROUTING_SECOND_CHOICE_POLICY_INVALID

Meaning: The orchestrator's bundled second-choice routing policy does not match its schema or contains duplicate tier or service IDs. This is an internal policy-data error.

Action: Update or reinstall the installer bundle so its orchestrator policy and validator are from the same release. If the error persists, report the error code, installer version, and diagnostic details to support.

9.2.53 ROUTING_SECOND_CHOICE_SCHEMA_UNSUPPORTED

Meaning: The orchestrator cannot load the schema version declared by its bundled second-choice routing policy.

Action: Update or reinstall the installer bundle so its orchestrator policy and validator are from the same release. If the error persists, report the error code, installer version, and diagnostic details to support.

9.2.54 SERVICES_LOCK_UNRELIABLE_WRITE_FAILED

Meaning: During `project-remove`, at least one managed file was already deleted, then another deletion failed. When the installer tried to mark `services-lock.json` as `integrity.state = UNRELIABLE`, writing that marker also failed.

Action:

1. Treat the lock state as potentially stale.
2. Inspect current project files and compare with lock content.
3. Repair or regenerate lock state before further mutating operations.

9.2.55 SERVICES_SCHEMA_UNSUPPORTED

Meaning: (`project-add`, `project-deploy`) The selected context bundle uses a deployment format that this installer release does not support. With `--canonical`, supported contexts can still deploy to the portable `.cube-ai/` layout.

Action: Update the installer and context bundle, then retry. If the error persists, report the context bundle name, version, and error details to support.

9.2.56 SERVICES_VALIDATION_FAILED

Meaning: A service selection passed via `--services` is invalid (unknown service ID or explicit empty value).

Action: For `project-add`, inspect the context's `--list-ai-services` output or its `services.json`. Omit `--services` for default and required services, or pass `--services all` to deploy every service. For `project-remove`, inspect `project-lock-show` and use IDs currently deployed in that project. Omit `--services` to remove the whole context.

9.2.57 SHOW_CATALOG_FAILED

Meaning: (`show-catalog`) The installer could not display the supported contexts.

Action:

1. Re-run with `--verbose debug` and capture stdout/stderr.
2. If the error is one of `CATALOG_NOT_FOUND`, `CATALOG_PARSE_ERROR`, or `CATALOG_INVALID`, follow the catalog-specific action above.
3. If the command still returns `SHOW_CATALOG_FAILED`, report the failure with `error.cause` and `error.details`.

9.2.58 TARGET_AMBIGUOUS

Meaning: `--autodetect-targets enforce` was used but detection found multiple possible targets and could not resolve to one unambiguously.

Action: Provide explicit `--ide` and `--agentic-system` flags, or switch to `--autodetect-targets advisory` to see the suggestions without failing.

9.2.59 TARGET_CHANGED_IN_DEPLOYED_PROJECT

Meaning: The project already has deployed contexts, and the command asked for another `--ide`, `--agentic-system` or `--canonical`. A project has one target: deploying to another would leave the files and MCP servers of the first deployment behind, outside the lock. The command stops before writing any file.

Action: Read `error.details.recordedTarget` and use the same `--ide` and `--agentic-system`. To move the project to the requested target, run `project-remove` for each deployed context, then deploy again with the new target.

9.2.60 TARGET_COMBINATION_NOT_SUPPORTED

Meaning: The `--ide` value was not validated with the selected `--agentic-system`. The command stops before writing any file.

Action: Pick one of the IDEs listed in `error.details.supportedIdes`, for example `--ide none` for a CLI session. See *Target selection* for the accepted pairs.

9.2.61 TARGET_NOT_DETECTED

Meaning: `--autodetect-targets enforce` was used but no IDE or agentic-system marker was found in the project folder. The installer refuses to fall back silently to defaults.

Common situations:

- The project folder is empty or freshly created (no `.vscode/`, `.github/`, `.cursor/`, etc.).
- `--ide` or `--agentic-system` was omitted while targeting a tool whose folder markers do not exist yet.

Action:

1. Check `details.targetType` in the error JSON (`ide` or `agenticSystem`) to know which dimension failed.
2. Provide the failing dimension explicitly, or create the expected marker first (e.g. open the project in VS Code so `.vscode/` is created). You can mix explicit and autodetected dimensions:

```
cube agentic-ai project-add <bundle> <project> --autodetect-targets enforce --agentic-system␣
↪github-copilot
cube agentic-ai project-add <bundle> <project> --autodetect-targets enforce --ide vscode
```

3. To inspect candidates without failing, use advisory mode first:

```
cube agentic-ai project-add <bundle> <project> --autodetect-targets advisory
```

9.2.62 TARGET_PROFILE_UNSUPPORTED

Meaning: The requested IDE/agentic-system tuple has no supported projection profile.

Action: Use a supported target or deploy with the canonical layout.

9.2.63 USER_JOURNEY_POLICY_INVALID

Meaning: The orchestrator's bundled user-journey policy, which defines skills such as `cube-agentic-ai-user-journey-mx2-cmake-build`, does not match its schema or contains duplicate journey, skill, or step IDs. This is an internal policy-data error.

Action: Update or reinstall the installer bundle so its orchestrator policy and validator are from the same release. If the error persists, report the error code, installer version, and diagnostic details to support.

9.2.64 USER_JOURNEY_SCHEMA_UNSUPPORTED

Meaning: The orchestrator cannot load the schema version declared by its bundled user-journey policy.

Action: Update or reinstall the installer bundle so its orchestrator policy and validator are from the same release. If the error persists, report the error code, installer version, and diagnostic details to support.

9.3 Codes of `setup` and `doctor`

9.3.1 VS Code extension repair failures

With `doctor --fix --vscode-profile "<name>"`, inspect `data.vscodeInstall` and `data.fixes` even when `ok` is true:

Result	Cause and action
`PROFILE_UNAVAILABLE`	The named profile could not be listed. Check its exact name in VS Code and the returned details. Create it manually if needed, then retry. No extension was installed there.
`CODE_FAILED`	`code` was absent, failed to start, or timed out. Fix PATH/launcher access or use that profile's Extensions view. The extension state is unknown, not known missing.
`Extension FAILED`	The download/install command failed. Read its `details` for network, marketplace or permission errors, fix the cause and retry the same command. Already-installed extensions are not reinstalled or updated.

Without an explicit profile, `--fix` intentionally leaves extensions manual. Supply the exact profile name to authorize automatic repair. These results describe the named profile, not whether the active VS Code window has loaded the extension. Reload that window after installation.

9.3.2 CONTEXTS_EMPTY

Meaning: (`setup`, `doctor`) `--contexts` or `--bundles` was given without any bundle name.

Action: Name at least one catalog bundle, or omit the option. `cube agentic-ai show-catalog` lists the names.

9.3.3 CONTEXT_REQUIRES_VSCODE

Meaning: (`setup`) A context named with `--contexts` needs VS Code with STM32CubeIDE for Visual Studio Code, and the target is another IDE or the canonical layout. `error.details.bundleNames` lists them. Without `--contexts`, such contexts are skipped and listed in `data.unavailableContexts` instead of failing the run.

Action: Remove the context from `--contexts`, or use `--ide vscode` without `--canonical`.

9.3.4 BUNDLE_IDE_NOT_SUPPORTED

Meaning: (`project-add`) The bundle lists the IDEs it supports in its `services.json`, and the IDE of the request is not one of them, or `--canonical` was given. A bundle that lists no IDE supports every IDE. `error.details` has the bundle, its IDEs and the requested IDE.

Action: Use an `--ide` that the bundle lists, without `--canonical`, or leave the context out.

9.3.5 INVALID_PYTHON_CHOICE

Meaning: (`setup`) `--python` is not `auto`, `venv`, or the absolute path of an existing Python executable.

Action: Use `--python auto`, `--python venv`, or `--python "<absolute path to python>"`.

9.3.6 INVALID_VSCODE_PROFILE_NAME

Meaning: (`setup`, `doctor`) A `--vscode-profile` name uses characters other than letters of any script (Chinese, Japanese and Korean included), digits, spaces, dots, underscores and hyphens, starts with something else than a letter or digit, or is longer than 64 characters.

Action: Use the exact name of the profile as VS Code shows it. Create the profile in VS Code first if it does not exist.

9.3.7 PROJECT_FOLDER_UNSAFE

Meaning: (`setup`) `--project-folder` is the home folder or a file system root.

Action: Choose a dedicated folder, for example `<home>/my-agentic-ai-project` (the default).

9.3.8 PROJECT_FOLDER_CREATE_FAILED

Meaning: (`setup`) The project folder cannot be created: its path exists and is not a folder, or the permissions do not allow it. `error.cause` has the path and the reason.

Action: Pick another path, or fix the permissions, then rerun `setup`.

9.3.9 SETUP_PROJECT_ALREADY_DEPLOYED

Meaning: (`setup`) The project folder already has a services lock: contexts were deployed there before. `setup` works only for a project folder that has never been deployed. `error.cause` has the path of the lock.

Action: Choose another project folder for a new deployment. To change this project's contexts, use `project-add` or `project-remove`, then `doctor <project_top_folder> --fix`. If the desired context version is not installed yet, run `machine-bootstrap --upgrade-installed-in-range true` first.

9.3.10 SETUP_FAILED and DOCTOR_FAILED

Meaning: An unexpected error stopped `setup` or `doctor` before it could report.

Action: Rerun with `--verbose debug` and capture stdout and stderr. Report `error.cause` if it persists.

9.3.11 Warnings of `setup` and `doctor`

These codes appear in `data.warnings[]` and do not fail the command.

Code	Meaning	Action
`REQUIREMENTS_DECLARATION_INVALID`	The `requirements` of a bundle do not match the schema. They are ignored, and the bundle is checked without them.	Update the bundle. Report it to its owner if it is up to date.
`IDES_DECLARATION_INVALID`	The `ides` declaration of a bundle is not valid, or its `services.json` is older than 3.1.0. It is ignored.	Update the bundle.
`IDE_DECLARATION_DIFFERS`	The bundle lists IDEs that the installer catalog entry does not list, or lists none. The bundle's own list decides which deployments are refused. The catalog only decides which contexts `setup` installs.	None for the user. Update the installer to get the matching catalog.
`VSCODE_INSTALL_SKIPPED`	(`setup --install-vscode-extensions`) The option did not apply: the target is not VS Code (another `--ide`, or `--canonical`), no installed context declares an extension, or `code` is not on the `PATH`. `data.vscodeInstall.reason` says which. Nothing was installed by the installer.	Use `--ide vscode` without `--canonical`, or run the commands of `data.vscodeProfiles` yourself.
`VSCODE_INSTALL_INCOMPLETE`	(`setup --install-vscode-extensions`) A profile could not be listed (usually a wrong name; the installer never creates a profile), `code` failed, or an extension did not install. `data.vscodeInstall.profiles[]` has the cause of each.	Fix the profile name, or install what is missing with the commands of `data.vscodeProfiles` or from the Extensions view.
`VSCODE_CLI_NOT_FOUND`	(`setup`) An installed context declares a VS Code extension and the `code` command is not in a `PATH` directory, so the extensions could not be checked or installed from the terminal. They are not necessarily missing.	Install them from the Extensions view of VS Code, or accept the proposal that VS Code makes when the folder opens. To get `code`: on macOS, run the Command Palette command "Shell Command: Install 'code' command in PATH" (named "Install 'code' and optionally 'copilot' commands in PATH" in recent versions); on Windows and Linux, reinstall VS Code or add its `bin` folder to the `PATH`. See the VS Code command line reference.
`RECOMMENDATIONS_NOT_WRITTEN`	Writing `.vscode/extensions.json` failed (permissions, locked file). A file with comments or an unexpected shape does not raise it: it is left unchanged and reported as `NOT_UPDATED` in `data.recommendations`.	Add the extensions of `data.vscodeExtensions` to its `recommendations` yourself.
`LICENSES_UNKNOWN_BEFORE_INSTALL`	(`setup --dry-run`) The licenses of bundles that are not installed yet cannot be read.	Run `setup` and read `data.licenses`.
`ONLINE_INDEX_UNREACHABLE`	(`setup --dry-run`) The online index could not be read, so the versions to install are unknown.	Check the network connectivity and retry.