STM32Cube for Agentic AI Documentation

Release 1.0.0

Nov 03, 2026

Contents

1 STM32Cube for Agentic AI 4
1.1 Quick start . . . 4
1.2 Explore the toolkit . . . 4
1.3 Try it out . . . 4
1.4 Project-based deployment . . . 5
1.5 Bundles and scope . . . 5

2 Choose your installation 6
2.1 Installation decisions . . . 6
2.2 Select editor and agent . . . 7
2.3 Typical installation . . . 7
2.4 Custom installation . . . 7
2.5 Check installation readiness . . . 7

3 STM32Cube Agentic Contexts 8
3.1 STM32Cube for Agentic AI installer . . . 8
3.1.1 Purpose and scope . . . 8
3.1.2 Download and deploy . . . 8
3.1.3 Assets and capabilities . . . 8
3.2 STM32CubeCLI and project housekeeping . . . 9
3.2.1 Purpose and scope . . . 9
3.2.2 Responsibilities . . . 9
3.3 STM32Cube Build Context . . . 9
3.3.1 Purpose and scope . . . 9
3.3.2 Troubleshooting . . . 10
3.3.3 Typical prompts . . . 10
3.4 STM32 DAP Debug Context . . . 10
3.4.1 Purpose and scope . . . 10
3.4.2 Debug workflows . . . 10
3.4.3 Troubleshooting . . . 11
3.4.4 Typical prompts . . . 11
3.5 STM32 Errata Hunter Context . . . 11
3.5.1 Purpose and scope . . . 11
3.5.2 Context scope . . . 11
3.5.3 Assets and capabilities . . . 11
3.5.4 Cube commands . . . 11
3.5.5 Typical prompts . . . 11
3.6 STM32Cube Embedded Software Context . . . 12
3.6.1 Purpose and scope . . . 12
3.6.2 Context scope . . . 12
3.6.3 Assets and capabilities . . . 12
3.6.4 Cube commands . . . 13
3.6.5 Typical prompts . . . 13
3.7 STM32CubeMX2 Context . . . 13
3.7.1 Purpose and scope . . . 13
3.7.2 Context scope . . . 14
3.7.3 Assets and capabilities . . . 14
3.7.4 Cube commands . . . 14
3.7.5 Typical prompts . . . 14
3.8 STM32CubeProgrammer Context . . . 14
3.8.1 Purpose and scope . . . 14
3.8.2 Assets and capabilities . . . 15
3.8.3 Cube commands . . . 15

3.8.4 Troubleshooting . . . 15
3.8.5 Typical prompts . . . 15
3.9 STM32 Sidekick Knowledge Base Context . . . 15
3.9.1 Purpose and scope . . . 15
3.9.2 Context scope . . . 15
3.9.3 Assets and capabilities . . . 15
3.9.4 Cube commands . . . 16
3.9.5 Typical prompts . . . 16
3.10 ST Official Documentation Context . . . 16
3.10.1 Purpose and scope . . . 16
3.10.2 Context scope . . . 16
3.10.3 Assets and capabilities . . . 16
3.10.4 Cube commands . . . 16
3.10.5 Typical prompts . . . 17

4 Toolkit status 18
4.1 Supported IDEs and AI coding agents . . . 18
4.1.1 Supported combinations . . . 18
4.1.2 Contexts that need VS Code . . . 19
4.1.3 How each agent reads the deployed files . . . 19
4.1.4 Approve the MCP servers before first use . . . 20
4.1.5 One target per project . . . 20
4.1.6 Not supported . . . 21
4.2 Prerequisites . . . 21
4.2.1 System requirements . . . 21
4.2.2 VS Code build and debug . . . 21
4.2.3 CMake without the Build Context . . . 21
4.2.4 Related guidance . . . 21
4.3 Supported contexts and versions . . . 22
4.4 Known limitations . . . 23
4.5 Why is STM32Cube for Agentic AI released as a beta? . . . 23
4.5.1 What this means for you . . . 23
4.5.2 Known limitations . . . 24

5 FAQ and troubleshooting 25
5.1 How should I work with an AI coding assistant? . . . 25
5.2 Why use the toolkit? . . . 25
5.3 How does custom installation differ from setup? . . . 25
5.4 How can I use an existing venv? . . . 26
5.5 What does `doctor` check? . . . 26
5.6 Why are dependencies missing after setup? . . . 26
5.7 How can I diagnose eSW Python setup? . . . 26
5.8 Where should contexts be installed? . . . 27
5.9 Why is a context unavailable? . . . 27
5.10 Why generate at the workspace root? . . . 27
5.11 What do service IDs mean? . . . 27
5.12 Do old bundle versions accumulate? . . . 28
5.13 Why did updating a bundle not update my project? . . . 28
5.14 Can the toolkit automatically build or debug with other IDEs? . . . 28
5.15 Why can the build or debug check fail in VS Code? . . . 28
5.16 What should I do when PowerShell shows `^U`? . . . 28
5.17 Why does MX2 fail in a terminal? . . . 29
5.18 Why did setup stop on an existing deployment? . . . 29

1 STM32Cube for Agentic AI

Beta release. See *Why is STM32Cube for Agentic AI released as a beta?*.

Ask an AI coding agent to turn an application idea into STM32 firmware: configure peripherals, check documentation and silicon errata, build and program the firmware, and verify its behavior on a connected board. STM32Cube for Agentic AI provides the technical contexts that *connect the agent to the STM32Cube ecosystem*.

1.1 Quick start

1. Check *Prerequisites* and choose an editor and AI coding agent in *Supported IDEs and AI coding agents*.
2. Follow the installation guide to install and deploy contexts into a project folder.
3. Open that folder with the selected agent and ask: `List the STM32 services deployed in this project.` Use one service for a read-only question and ask the agent to cite its source or evidence. See *Try it out* for a first-use example.

1.2 Explore the toolkit

Choose the contexts needed for a project. The six supported editor-and-agent combinations describe host setups, not the number of contexts. See *STM32Cube Agentic Contexts* for the context inventory.

1.3 Try it out

Copy and paste this prompt into your favorite coding assistant, augmented with the *STM32Cube for Agentic AI contexts*:

```
I have an STM32C5 Nucleo plugged in. Build me a small temperature logger on it.

Sample the MCU's internal temperature sensor with ADC1 through DMA and print
one line per second over the virtual COM port at 115200, formatted exactly as
"temp=NN.N C". Use the factory calibration values so the reading is in real
degrees, not raw counts.

Blink the user LED at 2 Hz so I can see the system is alive. When I press the
user button, pause the logging and print "logging paused". Press it again to
resume and print "logging resumed".

Before you settle the design, check whether any documented silicon errata
affect the way you are using the ADC or DMA on this device. If any do,
implement the recommended workaround and tell me the errata IDs you applied.

Leave me a short handoff note describing what was configured and why.

Get it running on the board and show me it works.
```

Note

The toolkit is in beta: see *Why the toolkit is in beta*. We use feedback to improve the contexts and publish updates continuously. Context bundles can be updated independently. The installer is the central place to discover and deploy them.

1.4 Project-based deployment

Deploy contexts into a project or workspace, not into the user home folder. This lets each project use its own à-la-carte context selection. Only the context bundles are installed at machine scope. Each project records its deployed versions and services.

1.5 Bundles and scope

STM32Cube Agentic Contexts are distributed as independently versioned bundles through STM32CubeCLI. A context can use packaged data, project files, or a remote MCP service. The selected AI agent remains responsible for the conversation and approved actions.

For bundle concepts, see the STM32CubeCLI documentation. For legal information and installation, see the installation guide.

2 Choose your installation

STM32Cube for Agentic AI is modular. Install context bundles once on the machine, then deploy the contexts and services needed by each project. See *STM32Cube Agentic Contexts* to compare their capabilities.

Choose **Typical** to deploy all contexts compatible with your selected target. Choose **Custom** to select installed contexts and their services. Both routes support all six native editor-and-agent combinations. This page helps choose a route. Follow the installation guide for prerequisites and commands.

2.1 Installation decisions

Select the editor and agent, then choose a route. The target determines which contexts are available. The route determines whether you deploy all compatible contexts or choose them individually.

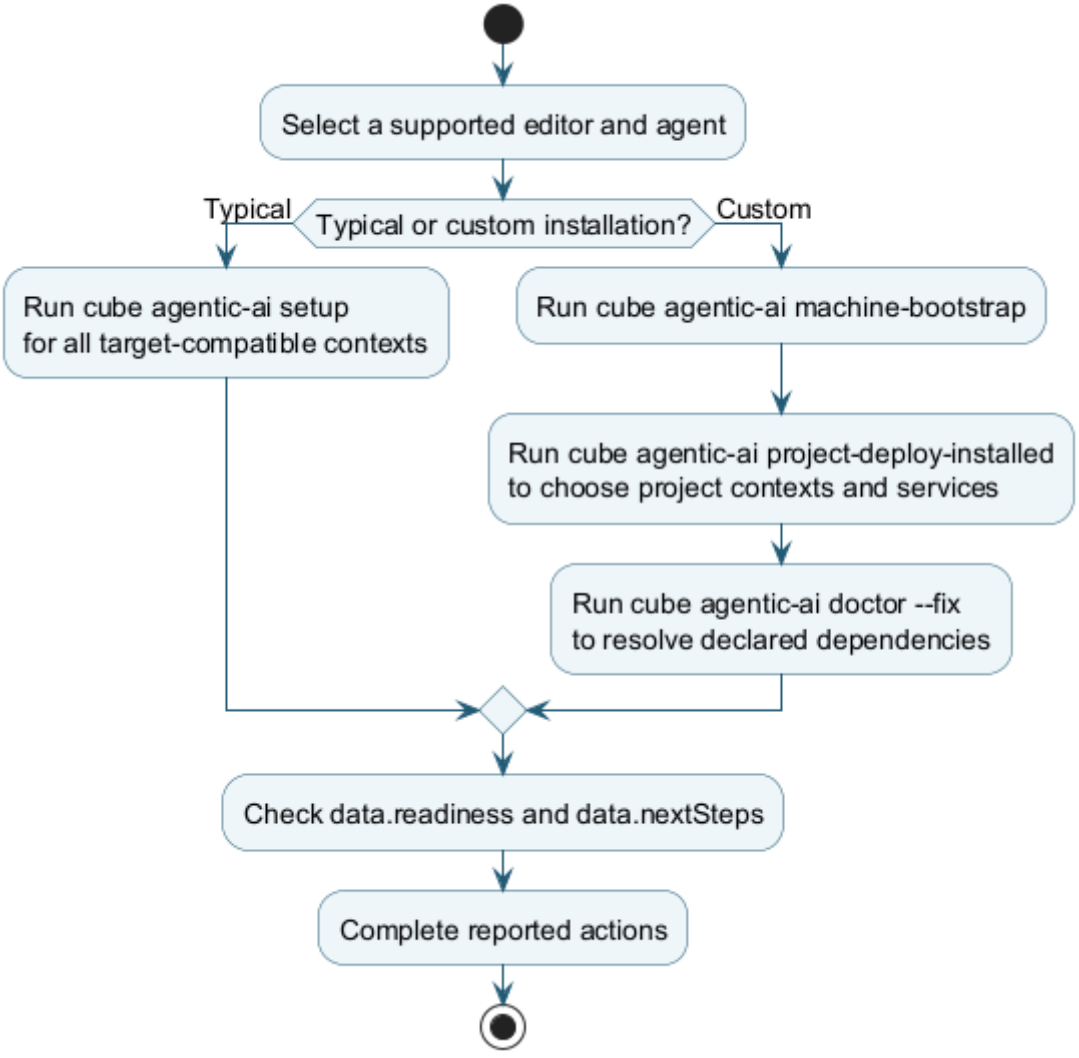


Fig. 1: Choose a target and installation route

2.2 Select editor and agent

Choose one of the supported editor-and-agent combinations in *Supported IDEs and AI coding agents*. The installer does not install the editor or the AI agent. Build and DAP Debug contexts require the VS Code IDE target.

2.3 Typical installation

Use `setup` for a project not yet deployed by this installer. It installs and deploys all contexts compatible with the selected target. Use the custom route to choose individual contexts or services. For terminal steps, see the typical manual installation. For an agent-led prompt, see the typical agent installation.

The Embedded Software Context requires Python 3.10 or later. Select the Python option during typical setup. The recommended isolated choice is `venv`. See the *Embedded Software Context guide* for more information. The *Embedded Software Context Python troubleshooting FAQ* covers detection, repair, and environment maintenance.

For target-specific extension and profile behavior, follow the installation guide. Avoid choosing a profile here without first confirming which editor target you selected.

2.4 Custom installation

Use the custom route when you need to choose contexts and services separately. It installs contexts on the machine, then deploys selected items to the project and checks declared dependencies. For terminal steps, see the custom manual installation. For an agent-led prompt, see the custom agent installation. Do not omit dependency repair after deployment.

For profile-specific extension repair, see the VS Code-only instructions in the installation guide.

2.5 Check installation readiness

After either route, check `data.readiness` and `data.nextSteps`. A successful deployment is not proof that every dependency is ready. Complete manual prerequisites and rerun `doctor` when needed. For the shared finish checklist and repair commands, see the installation guide. Use `project-add` or `project-remove` rather than `setup` to change an already deployed project. The installer user manual covers command details and maintenance.

3 STM32Cube Agentic Contexts

Each context adds focused STM32Cube knowledge or workflows to an AI coding agent. Select the contexts needed by each project. The table links to the detailed guide for each context.

Context	Scope
STM32Cube for Agentic AI installer	Installs context bundles on the machine and deploys selected context files into a project.
STM32CubeCLI and project housekeeping	Guides STM32CubeCLI and manages project-scoped context deployment.
STM32Cube Build Context	Checks CMake project prerequisites and launches builds through `cube cmake`.
STM32 DAP Debug Context	Prepares, checks, and launches STM32 debug sessions through DebugMCP.
STM32 Errata Hunter Context	Finds and analyzes published silicon errata.
STM32Cube Embedded Software Context	Discovers examples, provides HAL/LL guidance, and assists with application code.
STM32CubeMX2 Context	Creates, configures, and generates STM32CubeMX2 projects.
STM32CubeProgrammer Context	Programs internal and external NVM, erases and reads memory, and programs option bytes on STM32 MCUs. STM32 MPUs are not covered.
STM32 Sidekick Knowledge Base Context	Searches supported knowledge-base sources and returns provenance.
ST Official Documentation Context	Discovers official ST documents and retrieves available assets.

The installer catalog can change as new contexts become available. See *Toolkit status* for the currently supported contexts and versions.

3.1 STM32Cube for Agentic AI installer

3.1.1 Purpose and scope

The STM32Cube for Agentic AI installer downloads and deploys STM32Cube Agentic Context bundles. It provides the `cube agentic-ai` command. Context bundles follow the `cube-agentic-ai-<connector>` naming pattern, and the portable project layout uses the `.cube-ai/` folder.

The installer is not a technical context and does not answer STM32 technical questions. It deploys the required project-housekeeping skill, which manages the contexts and services available in a project and can identify a relevant missing context. Technical task routing remains a separate capability.

3.1.2 Download and deploy

Download places STM32Cube Agentic Context bundles on the machine. Cube installs dependencies declared by each bundle. Additional prerequisites, such as the STM32CubeMX2 application or a VS Code extension, may need to be installed separately and are listed on the relevant context page.

Deploy makes the selected context capabilities available in a project folder. The AI coding agent can then use those capabilities when they match a task. The installer supports GitHub Copilot, Claude Code and Cursor, in an editor or in a terminal. See *Supported IDEs and AI coding agents* for the supported combinations and the options to select them.

3.1.3 Assets and capabilities

Each STM32Cube Agentic Context describes its capabilities and scope. The installer deploys the selected context files and tracks them for updates and removal.

The installer is delivered in the `cube-agentic-ai-installer` bundle. It uses STM32CubeCLI to download bundles and deploy selected capabilities to project folders.

For the difference between general Cube CLI guidance and project housekeeping, see *Cube CLI wrapper and project housekeeping*.

For installation procedure, see the installation guide.

For command details, see the installer user manual.

> **Note**
>
> The installer also embeds its user manual in Markdown format. An AI Coding Agent can read this manual to understand how to use the installer on behalf of the user.

3.2 STM32CubeCLI and project housekeeping

3.2.1 Purpose and scope

The `cube-agentic-ai-cube-wrapper` context guides general STM32CubeCLI use: installing and troubleshooting the CLI, discovering commands, managing Cube bundles and dependencies, handling licenses and downloads, and configuring environment overrides.

The installer provides the `cli.project-housekeeping` skill for managing STM32Cube Agentic AI contexts and services in a project. It lists available or deployed capabilities, guides add, update, and remove operations, and helps identify a relevant missing context.

3.2.2 Responsibilities

Capability	Scope
Cube wrapper context	General STM32CubeCLI and Cube bundle workflows. Installing a bundle on the machine does not make its context available to a project.
Installer project-housekeeping skill	Project-scoped lifecycle for `cube-agentic-ai-*` contexts and their services, including identifying a relevant missing context. It does not answer STM32 technical questions or perform technical task routing.

For Agentic AI connector requests, use project housekeeping rather than the generic `cube bundle install` workflow. `cube bundle install` installs a bundle on the machine. The command `cube agentic-ai project-add` deploys the connector into a project and installs it on the machine if needed. Machine installation and project deployment are separate states.

The housekeeping skill uses the project's routing guidance to identify which technical capabilities apply. It manages what is deployed. It does not replace technical task routing.

For command syntax and project-lifecycle workflows, see the installer user manual.

3.3 STM32Cube Build Context

3.3.1 Purpose and scope

The STM32Cube Build Context checks prerequisites and launches builds for CMake-based STM32 projects through `cube cmake`. It does not program, erase, or debug a target.

Use this context in STM32CubeIDE for Visual Studio Code. It supports the VS Code target only and requires the declared `stmicroelectronics.stm32-vscode-extension` dependency. During setup, add `--install-vscode-extensions` to install declared extensions in the selected profile. For an existing project, run `cube agentic-ai doctor <project> --fix --vscode-profile "<exact profile name>"`. Without an explicit profile and installation request, the installer reports the required commands. The CMake project must be the workspace root

and must be discovered by STM32CubeIDE for Visual Studio Code (cubified). See the STM32CubeIDE for Visual Studio Code documentation. After project generation, run `Developer: Reload Window` once and wait for CMake configuration to finish before asking the agent to build.

The project must contain a root `CMakeLists.txt` and a `CMakePresets.json` file when it uses presets. `setup` installs the context's declared CMake bundle dependency. It checks any project-specific toolchain requirements and does not change build files. Before a build, the user must confirm that the project is trusted for execution or approve a named sandbox.

The context is delivered in the `cube-agentic-ai-build` bundle.

3.3.2 Troubleshooting

For common VS Code project-discovery and build-check failures, see *FAQ and troubleshooting*.

3.3.3 Typical prompts

- `Check the CMake build environment and build this STM32Cube project.`

3.4 STM32 DAP Debug Context

Important

This context requires the DebugMCP VS Code extension for debug sessions and for flashing routed through the debugger. Installing STM32CubeProgrammer alone does not satisfy this dependency. To use the STM32CubeProgrammer Context for flashing without DebugMCP, the DAP Debug Context is not needed in the project.

3.4.1 Purpose and scope

The STM32 DAP Debug Context prepares, checks, and launches debug sessions for STM32 projects through the DebugMCP VS Code extension.

3.4.2 Debug workflows

- Check the debug environment.
- Create or update a debug configuration.
- Check the configuration before launch.
- Launch a debug session through an ST-LINK GDB server.

Use this context in STM32CubeIDE for Visual Studio Code. VS Code is required. The project must be the top-level folder in the VS Code workspace and must be discovered by STM32CubeIDE for Visual Studio Code (cubified) before debugging. See the STM32CubeIDE for Visual Studio Code documentation.

The context also requires the separately installed DebugMCP VS Code extension (`ozzafar.debugmcpextension`), as well as the STM32CubeIDE for Visual Studio Code extension (`stmicroelectronics.stm32-vscode-extension`). The `cube-agentic-ai-debug-dap` bundle supports the VS Code target only. The installer declares both extensions as dependencies. For a VS Code target, `cube agentic-ai setup --install-vscode-extensions` installs them in the selected profile. For an existing project, run `cube agentic-ai doctor <project> --fix --vscode-profile "<exact profile name>"` to install missing declared extensions there. Without the explicit profile and `--fix`, doctor only reports the extension checks and next steps. VS Code itself is a prerequisite and is never installed by the toolkit.

DebugMCP is a third-party extension and provides the debug tools. To avoid requiring it for flashing through the STM32CubeProgrammer Context, omit this debug context. After setup and project generation, run `Developer: Reload Window` once for cubification and tool discovery. A reload is not needed after every debug configuration or MCP update. This context supports ST-LINK probes only. SEGGER/J-Link and generic GDB servers are not supported. Configure the ST-LINK probe according to its vendor instructions. The project also needs a valid STM32 debug configuration and the STM32Cube CMSIS scanner bundle. ST-LINK uses the `stlink-gdbserver` bundle, which setup resolves as a declared dependency.

A launch that writes a firmware image overwrites target flash and resets the device. The user must approve this operation before launch.

The context is delivered in the `cube-agentic-ai-debug-dap` bundle.

3.4.3 Troubleshooting

For common VS Code setup and debugger checks, see *FAQ and troubleshooting*.

3.4.4 Typical prompts

- `Check the debug environment and configure this project for an ST-LINK GDB server.`
- `Check this debug configuration, then launch a session using ST-LINK.`

3.5 STM32 Errata Hunter Context

3.5.1 Purpose and scope

The STM32 Errata Hunter Context searches published silicon errata and provides classification and impact analysis for a selected STM32 part and its peripherals. It reports source identifiers and links when available.

The context requires an identifiable full STM32 ordering code. It does not infer applicability from a family name alone.

3.5.2 Context scope

- `errata-hunter.errata-lookup` resolves the part number and selected peripherals, then returns matching errata and workarounds.

3.5.3 Assets and capabilities

The context is delivered in the `cube-agentic-ai-errata-hunter` bundle, which uses a remote service for errata lookup.

The remote service accepts a full ordering code and optional peripheral names, types, or wildcard patterns. The host AI system should cite the errata source when the service provides a document identifier and link.

3.5.4 Cube commands

The bundle command provides context inspection:

Table 1: :header: “Command”, “Use” :widths: 45, 55

`cube cube-agentic-ai-errata-hunter --help`	Show the context command help.
`cube cube-agentic-ai-errata-hunter --version`	Show the installed bundle version.
`cube cube-agentic-ai-errata-hunter --list-ai-services`	List the errata lookup service.

Errata lookup uses the remote service selected by this context.

3.5.5 Typical prompts

- `Is my current project exposed to known silicon errata? Identify the full STM32 ordering code and configured peripherals first, then cite applicable errata and workarounds.`
- `Check the errata that apply to STM32C562RET6 and SPI1.`
- `Find silicon limitations for the peripherals configured in this project.`
- `Does this STM32 part have an erratum affecting DMA? Cite the errata sheet.`
- `List the applicable workarounds and identify the affected peripheral.`

3.6 STM32Cube Embedded Software Context

3.6.1 Purpose and scope

The STM32Cube Embedded Software (eSW) Context provides example discovery, source analysis, validated HAL2/LL guidance, Python environment checks, and application-code assistance.

In bundle and command names, `esw` stands for embedded software.

The agent can also write or change application code using HAL2/LL, grounded in the project's sources, selected examples, and available guidance. This is not a standalone chatbot or a replacement for the Build, Debug, Programmer, or STM32CubeMX2 contexts. Ask for clarification when source evidence is insufficient. HAL/LL guidance currently covers STM32C5.

Example discovery checks whether a candidate covers the requested combination of peripherals. Example and guidance workflows do not change a project. Application-code workflows can change the selected project files. HAL1-to-HAL2 project migration is not delivered in this release.

3.6.2 Context scope

The installed eSW 1.0.0 service catalog uses these identifiers. Identifiers belong to this context and can differ between bundle versions:

Table 2: Embedded-software service selection

ID	Service	Purpose
S0	`esw.environment-setup`	Required Python environment checks and preparation.
S1	`esw.examples-explorer`	Example discovery and selected-source analysis.
S2	`esw.hal-ll-guidance`	Documented HAL2/LL usage, API reference, and runtime guidance.
S5	`esw.application-code`	Write, change, or complete application code using HAL2/LL.

Run `cube cube-agentic-ai-esw --list-ai-services` to inspect the catalog provided by the installed version before choosing services interactively.

- Find and compare examples by series, board, peripheral, or search terms. Retrieve a selected README and inspect explicitly selected source files.
- Get HAL2/LL documented usage, API reference, and runtime guidance from validated ST documentation.
- Write or change application code using HAL2/LL, grounded in the selected project's sources, examples, and available guidance.

3.6.3 Assets and capabilities

The context is delivered in the `cube-agentic-ai-esw` bundle. Python 3.10 or later is required for HAL/LL guidance. The toolkit does not install Python. Install it from the Python downloads page or use an existing Python executable.

Python setup

For a new project, `setup` supports `--python auto` (the default), `--python venv`, or `--python "<absolute executable path>"`. The installation guide uses `venv` for an isolated environment. The choice is recorded per project and eSW version, and `doctor --fix` reuses it.

If a selected service declares additional Python packages, the installer installs them when their declared version ranges are not met. It does not upgrade a compatible installed version. For interpreter detection, package repair, proxy configuration, and shared-environment maintenance, see *How can I diagnose eSW Python setup?*.

Network access

Example searches use the bundled catalog and do not require network access. Retrieving selected source files from GitHub requires network access and approval. HAL/LL guidance retrieves public ST documentation over HTTPS and caches successful responses locally for a limited time. Python package installation needs access to the configured package index or approved mirror.

3.6.4 Cube commands

The bundle command provides example operations:

Table 3: :header: “Command”, “Use” :widths: 55, 45

`cube cube-agentic-ai-esw --examples-query`	Search the example database. Use filters such as --search, --series, or --board.
`cube cube-agentic-ai-esw --list-ai-services`	List service identifiers, names, and descriptions for this installed version.
`cube cube-agentic-ai-esw --examples-readme`	Retrieve the README for a selected example.
`cube cube-agentic-ai-esw --examples-materialize`	Materialize explicitly named source files for analysis.
`cube cube-agentic-ai-esw --examples-navigate`	Navigate a selected local source projection.
`cube cube-agentic-ai-esw --examples-inspect`	Inspect a selected source file or projection.
`cube cube-agentic-ai-esw --examples-inventory`	List supported example series, boards, and peripherals.
`cube agentic-ai setup --python <auto|venv|path>`	Choose the Python used for eSW HAL/LL guidance in a new project.
`cube agentic-ai doctor <project> --fix`	Check and resolve declared dependencies for an existing project.

HAL/LL guidance uses a supervised lookup:

```
cube cube-agentic-ai-esw --run-with-feedback -- cube cube-agentic-ai-esw --hal-lookup <command> -
↪-target <target> --peripheral <PERIPHERAL>
```

Run `cube cube-agentic-ai-esw --help` for command arguments.

3.6.5 Typical prompts

- `Find a STM32Cube UART example for this board and explain why it matches.`
- `Show the README and initialization files for the closest SPI example.`
- `Compare the source files in these two STM32Cube examples.`
- `Find an example that uses DMA with this peripheral and cite the source files.`
- `Explain the HAL2 initialization sequence for this peripheral.`
- `Explain when this HAL2 API reports transfer completion and what its errors mean.`
- `Add a DMA-based UART transmit path to this HAL2 project, following the closest example and checking for resource conflicts.`

3.7 STM32CubeMX2 Context

3.7.1 Purpose and scope

The STM32CubeMX2 context provides two services for headless STM32CubeMX2 workflows: creating or configuring project models, and generating or exporting projects. It resolves board and MCU references, configures pins, periph-

erals, clocks, software parameters, middleware, utilities, and components, then can validate the project model when requested.

The context does not launch the STM32CubeMX2 graphical user interface, use GUI-backed extension commands, or ask for dialog-based interaction.

3.7.2 Context scope

- `mx2.create-or-configure-project` creates a project from an exact board or MCU reference, or opens and changes an existing `.ioc2` project. Validation is performed when requested.
- `mx2.generate-export` generates or exports a project and verifies output artifacts. CMake with GCC is the default. IAR (EWARM) and AC6 (Open-CMSIS) are also supported.

3.7.3 Assets and capabilities

The context is delivered in the `cube-agentic-ai-mx2-skill` bundle. It requires the STM32CubeMX2 application bundle. That application is a separate prerequisite from the connector. Install it if it is not already available. Use STM32CubeMX2 application version 1.2.0 and ensure it is the only installed `stm32cubemx-application` version to avoid command collisions. See the STM32CubeMX2 command-line reference.

The context uses headless workflows and does not launch the STM32CubeMX2 graphical user interface. CMake/GCC is the default export. IAR and AC6 use their native EWARM and Open-CMSIS formats.

3.7.4 Cube commands

The bundle command provides service discovery:

Command	Use
`cube cube-agentic-ai-mx2-skill --help`	Show the context command help.
`cube cube-agentic-ai-mx2-skill --version`	Show the installed bundle version.
`cube cube-agentic-ai-mx2-skill --list-ai-services`	List the available context services.

The AI coding agent invokes the relevant focused skill. The skill uses the local STM32CubeMX2 command surface with the project path and target details from the request.

3.7.5 Typical prompts

- `Create a project for the STM32 board NUCLEO-C562RE and validate it.`
- `Open this STM32CubeMX2 project and configure USART3 for the requested pins.`
- `Configure the clock tree and add the requested middleware.`
- `Generate a CMake/GCC project and verify its expected artifacts.`
- `Export this project in IAR format and verify its output files.`

3.8 STM32CubeProgrammer Context

3.8.1 Purpose and scope

The STM32CubeProgrammer Context programs internal and external nonvolatile memory (NVM), erases memory, reads memory, and programs option bytes on connected STM32 devices through an MCP service. It is delivered in the `cube-agentic-ai-programmer` bundle.

Device scope: All STM32 MCUs are supported. STM32 MPUs are not covered by this release.

3.8.2 Assets and capabilities

The connector includes its MCP server and declares the `programmer` Cube tool bundle as a dependency. Cube installs that dependency automatically. Connector bundles are available for 64-bit Windows and Linux, and for Intel and Apple silicon Macs. See the STM32CubeProgrammer product page.

The context can perform operations that change device state. The AI Coding Agent must identify the target and confirm the intended operation before erasing memory or programming option bytes. Review the impact of option-byte changes before programming them.

3.8.3 Cube commands

The `programmer` dependency provides the STM32CubeProgrammer CLI. The AI agent invokes the connector's MCP service for internal and external NVM programming, memory erase and read, and option-byte programming.

3.8.4 Troubleshooting

If an operation fails, confirm the selected target and connection, then inspect the MCP result. For agent-host or server-startup checks, see *FAQ and troubleshooting*.

3.8.5 Typical prompts

- `Program this firmware image to the connected STM32 device and verify it.`
- `Read the requested memory range from the connected STM32 device.`
- `Erase the selected STM32 device after confirming the target.`
- `Read the current option bytes, explain the requested changes, and program them after confirmation.`

3.9 STM32 Sidekick Knowledge Base Context

3.9.1 Purpose and scope

The STM32 Sidekick Knowledge Base Context provides one remote search service over approved STM32 content, the source corpus for Sidekick's retrieval-augmented generation (RAG) workflow. The service returns source provenance when available, identifying where content came from. Provenance does not necessarily include a URL. The host AI uses the retrieved content to generate a response. The context does not expose separate search services for individual source types.

Knowledge retrieval is provided by a remote Sidekick service. This context does not launch a local knowledge server.

3.9.2 Context scope

The knowledge base includes official documentation, wiki articles, STM32 Community articles, GitHub examples, and accepted solutions. Search results identify their sources when that information is available.

3.9.3 Assets and capabilities

The context is delivered in the `cube-agentic-ai-sidekick-kb` bundle. It searches a remote Sidekick service and returns source information with its results. The remote service must be reachable for searches to complete.

If a clear user need remains after retrieval, any handoff must use the project's routing guidance. The knowledge-search service does not select or recommend another connector by itself. Search results may be incomplete or out of date and do not prove that a result applies to a specific project.

3.9.4 Cube commands

The bundle command exposes context metadata:

Command	Use
`cube cube-agentic-ai-sidekick-kb --help`	Show the context command help.
`cube cube-agentic-ai-sidekick-kb --version`	Show the installed bundle version.
`cube cube-agentic-ai-sidekick-kb --list-ai-services`	List the knowledge-search service.

The AI coding agent invokes the remote service through the selected skill.

3.9.5 Typical prompts

- `Find the NUCLEO-C562RE user manual in the Sidekick knowledge base and summarize its power-supply options.`
- `Search the Sidekick knowledge base for published guidance on this pin's electrical limits.`
- `Find examples in the Sidekick knowledge base that use this STM32 peripheral pattern.`
- `Find accepted STM32 Community solutions for this error message in the Sidekick knowledge base.`

3.10 ST Official Documentation Context

3.10.1 Purpose and scope

The ST Official Documentation Context helps locate and use authoritative technical documents published by ST. It preserves document identity, version, source location, and citation information.

It is intended for document discovery and retrieval. It does not diagnose a project, recommend workarounds without a source, or replace the official documentation.

3.10.2 Context scope

- `st-docs.source-discovery` finds and cites official ST documents published on st.com, including reference manuals, datasheets, errata sheets, application notes, user manuals, programming manuals, technical notes, and release notes.
- `st-docs.document-topology` identifies authoritative document types and selects a minimal set of complementary documents after source discovery.
- `st-docs.raw-asset-retrieval` retrieves an available official document asset after its identity and source have been established.

3.10.3 Assets and capabilities

The context is delivered in the `cube-agentic-ai-st-docs` bundle, which provides skills, an ST documentation catalog, document-topology information, official-source configuration, and retrieval examples. It searches official ST documentation, not the Sidekick knowledge base.

Raw asset retrieval is performed only after source discovery and document identification. Available formats depend on the document catalog. The host AI system receives the result and its provenance and decides how to use it.

3.10.4 Cube commands

The bundle command exposes context metadata and service discovery:

Command	Use
`cube cube-agentic-ai-st-docs --help`	Show the context command help.
`cube cube-agentic-ai-st-docs --version`	Show the installed bundle version.
`cube cube-agentic-ai-st-docs --list-ai-services`	List the services declared by the context.

The AI coding agent calls the deployed skills when a request matches the context scope.

3.10.5 Typical prompts

- `Find the STM32C562 datasheet and cite the source.`
- `Which ST document describes the option bytes for this STM32 device?`
- `Retrieve the official PDF for this document and report its version.`
- `Select the ST documents needed to investigate this clock configuration issue.`

4 Toolkit status

The toolkit is in beta and does not have one shared bundle version. Contexts are versioned independently. The installer catalog is the source for the contexts available to a deployment.

Status topic	Details
Supported IDEs and AI coding agents	The six supported editor-and-agent combinations and their deployment layouts.
Prerequisites	Host operating systems, Python, VS Code, and context-specific tools. Related guidance links to the PowerShell Ctrl+U and STM32CubeMX2 Electron environment workarounds in the FAQ.
Supported contexts and versions	Current installer and context bundle versions, plus update guidance.
Known limitations	Current beta restrictions and workarounds.
Why the toolkit is in beta	Beta status, validation scope, and how the offer evolves.

The installer is the central point for catalog and context updates. To check the versions installed on a machine, run `cube agentic-ai machine-discovery`. To see contexts supported by the installed installer, run `cube agentic-ai show-catalog`.

4.1 Supported IDEs and AI coding agents

The installer deploys the Agentic Contexts to every supported AI coding agent. It adapts only the folder names, the file formats and the MCP configuration file to what each agent reads. The concepts, such as skills, agents, instructions and MCP servers, are the same everywhere. Two contexts depend on STM32CubeIDE for Visual Studio Code and are available only with `--ide vscode`, as described in *Contexts that need VS Code*.

You choose the target with two options of the `cube agentic-ai` commands:

- `--agentic-system` selects the AI coding agent: `github-copilot`, `claude-code` or `cursor`. It decides where skills, agents, instructions, the routing skill and the MCP configuration file are written, because the agent is the MCP client that reads them.
- `--ide` selects the editor that hosts the agent: `vscode`, `cursor` or `none` for a terminal session without an editor. It decides which editor extensions the installer proposes. It changes the MCP file only for GitHub Copilot in VS Code.

4.1.1 Supported combinations

Six combinations are supported. The installer rejects any other pair with the error `TARGET_COMBINATION_NOT_SUPPORTED`, for example `--ide cursor` with `--agentic-system github-copilot`. The option `--canonical` writes only the portable `.cube-ai/` layout and skips this check.

Here, **supported** means that the installer accepts the target and writes its documented configuration. It does not mean every agent and editor combination has been integration-tested. See *Known limitations* for the current validation coverage.

AI coding agent and editor	Options	MCP configuration file	Skills, agents and instructions	Project instruction file
GitHub Copilot in VS Code	`--ide vscode --agentic-system github-copilot` (default)	`.vscode/mcp.json`	`.github/`	`.github/copilot-instructions.md`
GitHub Copilot CLI, no editor	`--ide none --agentic-system github-copilot`	`.mcp.json`	`.github/`	`.github/copilot-instructions.md`
Claude Code CLI, no editor	`--ide none --agentic-system claude-code`	`.mcp.json`	`.claude/`	`CLAUDE.md`
Claude Code extension in VS Code	`--ide vscode --agentic-system claude-code`	`.mcp.json`	`.claude/`	`CLAUDE.md`
Cursor editor	`--ide cursor --agentic-system cursor`	`.cursor/mcp.json`	`.cursor/` (rules as `.cursor/rules/*.mdc`)	`AGENTS.md`
Cursor CLI, no editor	`--ide none --agentic-system cursor`	`.cursor/mcp.json`	`.cursor/` (rules as `.cursor/rules/*.mdc`)	`AGENTS.md`

The project instruction file is the file that the agent reads at the start of every session. The installer proposes the statement that routes STM32 requests to the routing skill, and you add it to that file.

For agent-specific VS Code extension prerequisites, see *Prerequisites*.

The installer deploys contexts into a project or workspace, not into the user home directory. Select contexts for each project independently. Only the context bundles are installed at machine scope. Project locks record the versions and services deployed in that project. See *Supported contexts and versions* for the current bundle versions.

The default target is GitHub Copilot in VS Code. To use another combination, pass both options on the `setup`, `project-add` and `project-deploy` commands. For example, for the Claude Code CLI:

```
cube agentic-ai project-add <bundle> <project> --ide none --agentic-system claude-code
```

4.1.2 Contexts that need VS Code

The *STM32Cube Build Context* and the *STM32 DAP Debug Context* need STM32CubeIDE for Visual Studio Code. The installer deploys them only when `--ide vscode` is used, whatever the AI coding agent. With `--ide none`, `--ide cursor` or `--canonical`, the installer does not install them. `setup` lists them as unavailable, and a `project-add` or `setup` command that names one of them fails with an error. All other contexts are available in the six combinations.

Combination	Build	DAP Debug	Other contexts
GitHub Copilot in VS Code	Yes	Yes	Yes
Claude Code extension in VS Code	Yes	Yes	Yes
GitHub Copilot CLI, Claude Code CLI	No	No	Yes
Cursor editor, Cursor CLI	No	No	Yes

4.1.3 How each agent reads the deployed files

GitHub Copilot

- Copilot in VS Code reads the MCP servers from `.vscode/mcp.json`, under the key `servers`. See the VS Code MCP servers documentation.

- Copilot CLI does not read `.vscode/mcp.json`. It reads `.mcp.json` from the working directory up to the repository root, under the key `mcpServers`. It also reads `.github/mcp.json`, but only inside a Git repository, so the installer writes `.mcp.json`. See the Copilot CLI MCP documentation.
- Skills are read from `.github/skills/`. See the GitHub documentation on agent skills.

Claude Code

- The Claude Code CLI and the Claude Code extension in VS Code share the same configuration. Both read `.mcp.json`, under the key `mcpServers`. See the Claude Code MCP documentation and the Claude Code VS Code documentation.
- Skills are read from `.claude/skills/`. See the Claude Code skills documentation.
- Claude Code reads `CLAUDE.md` as the project instruction file. It reads `AGENTS.md` only when no `CLAUDE.md` exists. See the Claude Code memory documentation.

Cursor

- The Cursor editor and the Cursor CLI share the same project files. Both read `.cursor/mcp.json`, under the key `mcpServers`. See the Cursor MCP documentation and the Cursor CLI MCP documentation.
- Skills are read from `.cursor/skills/<name>/SKILL.md` and subagents from `.cursor/agents/`. See the Cursor skills documentation and the Cursor subagents documentation.
- Cursor loads rules only from `.cursor/rules/*.mdc`. The installer converts each instruction to a rule: an instruction that applies to every file becomes an always-on rule, and an instruction with a file pattern becomes a rule that Cursor attaches when matching files are in the context. See the Cursor rules documentation.

4.1.4 Approve the MCP servers before first use

Each agent asks you to approve project MCP servers before it starts them. The installer cannot do this step for you.

Combination	What to do
GitHub Copilot in VS Code	Trust the workspace, then start the servers from `.vscode/mcp.json`.
GitHub Copilot CLI	Start `copilot` in the project folder and accept the folder trust prompt. In non-interactive mode (`copilot -p`), set the environment variable `GITHUB_COPILOT_PROMPT_MODE_WORKSPACE_MCP=true`. The command `copilot mcp list` does not list the project servers, so do not use it to check them.
Claude Code CLI, Claude Code extension in VS Code	Start `claude` in the project folder and approve the servers from `.mcp.json` when prompted. `claude mcp list` shows `Pending approval` until you do.
Cursor editor, Cursor CLI	Enable the servers in the Cursor settings, or run `agent mcp enable <server>`. For scripts, use the option `--approve-mcps`.

Note

If you start Copilot CLI from the terminal of a VS Code window that has the project folder open, Copilot CLI connects to that window and does not load the project `.mcp.json`. Run Copilot CLI from a terminal outside VS Code, or use Copilot Chat in VS Code.

4.1.5 One target per project

A project has one target. After you deploy a context, the installer rejects a command that asks for another `--ide`, `--agentic-system` or `--canonical` with the error `TARGET_CHANGED_IN_DEPLOYED_PROJECT`, and it changes nothing. To move a project to another target, remove each deployed context with `project-remove`, then deploy again with the new target.

4.1.6 Not supported

The installer does not provide native configuration for agents or editors outside the six combinations above. For another agentic system, `--canonical` writes the portable `.cube-ai/` layout. You must adapt it manually for that agent. This is not an additional native or integration-tested target. A pairing outside the table is rejected unless `--canonical` is used.

For the exact behavior of each option, see the installer user manual.

4.2 Prerequisites

Before installing, choose one of the *supported editor-and-agent setups*.

4.2.1 System requirements

Supported host systems are Windows 10 or 11 (64-bit), macOS 15 or 26, Ubuntu 22.04, 24.04, or 26.04, and Fedora 43 or 44. Keep at least 500 MB of free disk space for the contexts. Development tools and generated projects need additional space.

Use PowerShell, bash, or zsh. On Windows, PowerShell 7.3 or later is recommended. PowerShell 5.1 in the VS Code integrated terminal can display `^U` when Ctrl+U is pressed. See *What should I do when PowerShell shows ^U?* for a quick workaround.

Python 3.10 or later is required for the STM32Cube Embedded Software Context's HAL/LL guidance. Python and the VS Code application are prerequisites: the toolkit never installs either one. The installer can resolve declared Python packages. VS Code extensions are installed only when explicitly requested. Other contexts can declare their own dependencies. Check the context pages or run `cube agentic-ai doctor <project-folder>` after deployment.

GitHub Copilot in VS Code requires the GitHub Copilot Chat extension (`GitHub.copilot-chat`), and Claude Code in VS Code requires Anthropic's Claude Code extension. Install the appropriate agent extension yourself. The toolkit installer does not install or manage either one. Its optional VS Code extension installation applies to extensions declared by toolkit contexts. Install the agent extension from the GitHub Copilot Chat extension Marketplace page or the Claude Code for VS Code guide.

For interpreter selection, package provisioning, and repair, see the *Embedded Software Context*.

The installer and contexts need network access to the Cube bundle registry. Remote contexts also need access to their documented services. See the relevant context page for service-specific network requirements.

4.2.2 VS Code build and debug

Build and DAP Debug contexts are available only for the VS Code target. Their extension, workspace, and debugger requirements are described in the *Build* and *Debug* context pages. The installer installs their declared extensions only when explicitly requested.

4.2.3 CMake without the Build Context

The Build Context is optional. Without it, an agent can guide a standard CMake workflow, but it does not replace a compatible compiler toolchain or project configuration. Follow the CMake preset and build instructions provided by the project or its toolchain.

Optional Cube bundles include `cmake`, `ninja`, and `gnu-tools-for-stm32`. Installing tools, editing build files, and starting the build require the user's approval. The Build Context checks these prerequisites and obtains approval before it builds.

4.2.4 Related guidance

- *PowerShell Ctrl+U in VS Code*
- *STM32CubeMX2 startup and Electron environment*

4.3 Supported contexts and versions

The table lists reference bundle versions for this release, checked on November 2, 2026. The toolkit does not have one shared bundle version. Contexts are updated independently, and the installer catalog can change when new contexts or routing guidance are released. Use the commands below to check installed versions rather than assuming this table matches every machine.

Context	Scope	Bundle	Reference version
STM32Cube for Agentic AI installer	Installs context bundles on the machine and deploys selected context files into a project.	`cube-agentic-ai-i`	`1.0.0`
STM32Cube Build	Checks prerequisites and builds CMake-based STM32 projects.	`cube-agentic-ai-b`	`1.0.0`
STM32CubeCLI	Guides STM32CubeCLI commands, bundles, dependencies, licenses, and downloads.	`cube-agentic-ai-c`	`1.0.3`
STM32 DAP Debug	Prepares and launches STM32 debug sessions through DebugMCP.	`cube-agentic-ai-d`	`1.0.0`
STM32 Errata Hunter	Finds and analyzes published silicon errata.	`cube-agentic-ai-e`	`1.0.0`
STM32Cube Embedded Software	Discovers examples, provides HAL/LL guidance, and assists with application code.	`cube-agentic-ai-e`	`1.0.0`
STM32CubeMX2	Creates, configures, and generates STM32CubeMX2 projects.	`cube-agentic-ai-m`	`1.0.0`
STM32CubeProgrammer	Programs internal and external NVM, erases and reads memory, and programs option bytes on STM32 MCUs. STM32 MPUs are not covered.	`cube-agentic-ai-p`	`1.0.0`
STM32 Sidekick Knowledge Base	Searches supported STM32 knowledge-base sources with provenance.	`cube-agentic-ai-s`	`1.0.0`
ST Official Documentation	Discovers official ST documents and retrieves available assets.	`cube-agentic-ai-s`	`1.0.0`

The installer is separate from the technical contexts. Its release-reference version is `1.0.0`. Update it when the catalog gains contexts or its general project guidance changes.

Check installed versions and supported contexts with:

```
cube agentic-ai machine-discovery --ai-only --no-json
cube agentic-ai show-catalog
```

For each context's service identifiers and descriptions, use its discovery command, for example `cube cube-agentic-ai-esw --list-ai-services`. See *STM32Cube Embedded Software Context* for the eSW service mapping.

Update the installer first, then ask it to install or update eligible contexts from the configured registry:

```
cube bundle update cube-agentic-ai-installer
cube agentic-ai machine-bootstrap --upgrade-installed-in-range true
```

Updating a machine bundle does not change a deployed project. Before deploying a newer context version, remove the old context from the project, add the new version, then resolve its requirements. Run `doctor <project-folder>` and follow the target-preserving update commands in `data.nextSteps`:

```
cube agentic-ai doctor <project-folder> --fix
```

The registry can change after this page is built. Run `machine-discovery` for installed versions and `show-catalog` for the latest catalog carried by the installed installer.

4.4 Known limitations

These limitations apply to the beta. Verify generated projects, build output, device operations, and agent responses before relying on them. This list was reviewed on November 2, 2026.

Limitation	Affected component/version	Workaround or notes
STM32CubeMX2 GUI IP errors and warnings are not propagated by the connector.	STM32CubeMX2 Context.	Check STM32CubeMX2 GUI diagnostics before generation. A successful connector readiness result does not prove that the GUI has no blocking IP errors.
STM32CubeMX2 can display `EPIPE: broken pipe` because of local shell settings.	Local terminal environment used to launch STM32CubeMX2.	Check the shell environment before treating the popup as a project configuration failure. See *Why does MX2 fail in a terminal?* in the FAQ.
The toolkit is for evaluation and integration validation. No target is production-validated.	All contexts in this beta.	Verify agent output, project files, generated code, and device state. Keep backups and follow the tools' normal safety procedures.
Build and debug require a VS Code project setup.	Build and DAP Debug contexts. See *Supported contexts and versions* for current versions.	These contexts need the VS Code target. Follow the *Build* and *Debug* guides for setup and tool requirements.
HAL1-to-HAL2 project migration is not delivered.	STM32Cube Embedded Software Context.	Use the delivered example discovery, HAL/LL guidance, and application-code services. The toolkit does not provide a project-migration workflow in this release.
ST Docs does not yet support board user manuals.	ST Official Documentation Context. See *Supported contexts and versions* for current versions.	Search the Sidekick Knowledge Base for board manuals, including the NUCLEO-C562RE user manual. For other missing official documents, check st.com.
Accepted target combinations are not all integration-tested.	Installer target catalog. See *Supported IDEs and AI coding agents*.	VS Code with GitHub Copilot and Claude Code CLI are integration-tested pairings, not fully validated workflows. Cursor setups and other Claude Code setups are only partially validated. Other supported combinations are checked for file placement. Verify agent behavior in the selected host.
VS Code 1.142.x is excluded from this beta because of an upstream Copilot Local-mode change.	Beta host compatibility, not a connector version.	VS Code 1.141.x and 1.143.x are supported. Avoid 1.142.x. See microsoft/vscode#341937.

For the reason the toolkit is in beta, see *Why is STM32Cube for Agentic AI released as a beta?*. For changes to the current context inventory, see *Supported contexts and versions*.

4.5 Why is STM32Cube for Agentic AI released as a beta?

STM32Cube for Agentic AI is delivered as bundles by STM32CubeCLI, which has run inside STM32CubeIDE for Visual Studio Code since April 2025 and inside STM32CubeMX2 since March 2026. The tools that the contexts drive, such as STM32CubeMX2 and STM32CubeProgrammer, are released on their own.

What is still settling is the agentic layer: the list of contexts and services, the names of the skills and of the `cube agentic-ai` commands, and the guidance that the contexts give to the AI coding agent. New contexts and fixes ship as feedback comes in.

4.5.1 What this means for you

- The tools that the contexts drive keep their own release status.
- Context, service and command names can change before general availability. The release notes list each change.

- A project keeps the context versions it was deployed with. Updating the machine does not change a project until you deploy again.
- The AI coding agent and its model produce the answers. The toolkit does not itself perform AI inference or make autonomous decisions.
- The toolkit is for evaluation. No target is production-validated.
- Sidekick KB and Errata Hunter use proven data, but their MCP services remain beta.

4.5.2 Known limitations

For the current connector inventory and version-specific limitation table, see *Supported contexts and versions* and *Known limitations*.

Validation scope

Six editor-and-agent combinations are accepted, but integration testing covers only the pairings listed in *Known limitations*. The other combinations are tested for file placement. The current restrictions and workarounds are listed in *Known limitations* and *Supported IDEs and AI coding agents*.

Answer quality

The toolkit provides STM32Cube-specific context and tools, but response quality depends on the selected AI agent and its underlying model. Results can vary with the agent, model, and supplied project context. Review generated changes and verify technical results in the target environment.

5 FAQ and troubleshooting

For questions and feedback, visit the ST Community. Include the selected editor and agent, installed context versions, and the exact error. Remove credentials and private project information before posting.

5.1 How should I work with an AI coding assistant?

Treat the assistant as a development aid, not as the authority for project or device behavior. Results vary with the assistant, model, available context, and the details in the request.

For a focused request, identify the exact target and outcome. For example, name the NUCLEO-C562RE board, the STM32C562RET6 MCU, the peripheral, and the behavior expected. Include only the relevant source files, configuration, error messages, and constraints.

Before changes, ask the assistant to explain its proposed steps and which files or tools it expects to use. Review the changes and authorize software installation, builds, generated-file updates, and device operations before they run. Keep passwords, tokens, private keys, and other credentials out of prompts and project files.

Verify the result independently. Inspect project configuration and generated source, run the project's build and tests, and check behavior on the target board where applicable. A plausible explanation or successful generation does not prove that the project is correct. State any missing evidence and review *Known limitations* before relying on an uncertain answer.

5.2 Why use the toolkit?

A general-purpose model does not know the current project configuration or which STM32 sources apply. The toolkit gives the agent access to official documentation, selected software examples, and STM32Cube tool workflows. The installer resolves declared tool dependencies and records each project's context versions. Project files remain available for inspection and manual work in STM32Cube tools. Review the agent's sources and verify its results.

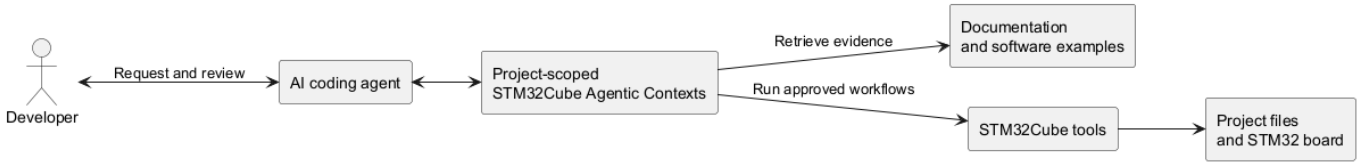


Fig. 1: STM32Cube contexts connect the agent to sources and tools

5.3 How does custom installation differ from setup?

Custom installation separates three levels of control:

- **Machine**: `cube agentic-ai machine-bootstrap` installs context bundles on the machine. It does not deploy contexts into a project.
- **Project**: `cube agentic-ai project-deploy-installed` selects installed contexts and services to deploy into a project.
- **Dependencies**: `cube agentic-ai doctor <project-folder> --fix` checks deployed services and resolves dependencies the installer can manage. Complete reported manual prerequisites separately.

`cube agentic-ai setup` combines these operations for a new project and abstracts their sequencing. Both paths report `data.readiness` and `data.nextSteps`. See *Choose your installation* for the installation choices and the installation guide in SEE ALSO for commands and options.

5.4 How can I use an existing venv?

For a new project, pass the existing venv's Python executable to `cube agentic-ai setup` with `--python "<absolute executable path>"`. Use the path ending in `Scripts/python.exe` on Windows or `bin/python` on macOS and Linux. `--python venv` instead selects the eSW bundle-local environment, not an existing user-created venv.

The custom installation path cannot explicitly select an existing venv through `doctor` in the current release. `doctor` has no `--python` option and uses `auto` for context checks. Activating a venv before running `doctor` is not a supported workaround for selecting where eSW packages are installed:

- With a saved eSW Python choice, `doctor` reuses that choice. A choice explicitly selected through `setup --python` is retained for later package checks and repair.
- Without a saved choice, eSW detects Python automatically. If package repair detects a virtual environment, it creates or reuses the bundle-local venv rather than installing into that detected environment.

Do not run `setup` on an already deployed project to change its Python choice. See *How can I diagnose eSW Python setup?* for detection and repair details.

A future release may add a `--python` option to `doctor` to allow explicit selection of an existing venv in the custom installation path. This option is not available in the current release.

5.5 What does `doctor` check?

`cube agentic-ai doctor <project-folder>` checks the requirements declared by deployed toolkit contexts. It reports whether required tools, Python packages, and declared VS Code extensions are present. Add `--fix` to resolve dependencies the installer can manage. Inspect `data.readiness` and `data.nextSteps` in the result.

`doctor` is not a general MCP or agent-host diagnostic tool. It does not validate arbitrary MCP JSON syntax or file placement, diagnose a server that fails to start, or check an agent's runtime, credentials, or network policy. For those issues, inspect the selected agent's MCP configuration and its own server status or startup logs. Do not put credentials in the chat or logs.

For example, to check and resolve the eSW Python packages in a deployed project, run:

```
cube agentic-ai doctor ./my-agentic-ai-project --bundles cube-agentic-ai-esw --fix
```

5.6 Why are dependencies missing after setup?

`data.status: ALL_DEPLOYED` confirms that the selected contexts were deployed. It does not guarantee that they are ready to use. Check `data.readiness`, `data.requirements`, and `data.nextSteps` for unresolved dependencies, manual prerequisites, or checks still to perform.

For example, omitting `--install-vscode-extensions` from setup leaves extension installation as a reported next step. Run `cube agentic-ai doctor <project-folder> --fix` to resolve supported requirements. Add `--vscode-profile "<exact existing profile name>"` to authorize installation of missing declared VS Code extensions in that profile. Without an explicit profile, doctor reports extension commands instead of installing them. Complete manual prerequisites, then rerun doctor. Do not rerun setup on the deployed project to repair dependencies.

5.7 How can I diagnose eSW Python setup?

The eSW context detects Python in this order: `py -3`, `python3`, then `python` on Windows. On macOS and Linux, it tries `python3`, then `python`. It uses the first candidate that runs as Python 3. That interpreter must be Python 3.10 or later. If the first candidate is older, the check reports it rather than trying a later command.

Run the detection commands in the same terminal that runs the installer. These commands show both the version and the executable path:

PowerShell

bash or zsh

```
py -3 --version
py -3 -c "import sys; print(sys.executable)"
python3 --version
python3 -c "import sys; print(sys.executable)"
python --version
python -c "import sys; print(sys.executable)"
```

```
python3 --version
python3 -c "import sys; print(sys.executable)"
python --version
python -c "import sys; print(sys.executable)"
```

If the first detected Python is older than 3.10, correct the launcher or `PATH` so it finds a supported version, or create a new project with `setup --python "<absolute path to Python 3.10 or later>"`. When detection finds a suitable Python, prefer `setup --python venv`. The bundle-local environment uses the detected Python as its base and is shared by projects using the same eSW bundle version.

Changing the system's default Python does not update an existing bundle venv. It remains based on the compatible Python detected when it was created. The user or organization is responsible for refreshing the shared venv when a Python upgrade is required, including for security updates. Remove the installed eSW bundle version's `pybin/.venv` directory, then run `cube agentic-ai doctor <project> --bundles cube-agentic-ai-esw --fix` for each affected deployed project configured to use `venv`. The command recreates the environment from the currently detected compatible Python and checks or installs the project's required packages. All projects using the same eSW bundle version then share the refreshed environment.

For a package-install failure in an existing project, run the eSW `doctor` command above and inspect its requirement and fix results. `doctor --fix` reuses the Python choice saved when that project was created. If pip cannot reach its package index, check the organization's proxy, certificate, mirror, and timeout settings, then retry. A failed pip install does not silently switch to another Python.

5.8 Where should contexts be installed?

Deploy contexts into the top-level folder of a project or workspace. Do not deploy them into the user home directory. This keeps context selection, configuration, and lock files project-scoped. See *Supported IDEs and AI coding agents* for the deployment layout and the installation guide for setup steps.

5.9 Why is a context unavailable?

The Build and DAP Debug contexts require the VS Code IDE target. They are not installed for Cursor or terminal-only targets. Check *Supported IDEs and AI coding agents* and the current list in *Supported contexts and versions*.

5.10 Why generate at the workspace root?

STM32CubeIDE for Visual Studio Code must discover the generated project before the Build and DAP Debug contexts can use it. Keep the STM32CubeMX2 project and generated build files in the top-level project folder opened in VS Code. Generating into a nested folder can prevent discovery. This is a requirement of that IDE workflow, not a general restriction on STM32CubeMX2 output paths.

For the project-folder requirements, see *Supported IDEs and AI coding agents* and the *Build* guide.

5.11 What do service IDs mean?

Interactive deployment lists services declared by each context bundle. Identifiers such as `S1` are local to that context, not toolkit-wide names. Inspect the installed context's service names and descriptions with:

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

For the embedded-software service mapping, see *STM32Cube Embedded Software Context*. For services already deployed into a project, run `cube agentic-ai project-lock-show <project-folder>`.

5.12 Do old bundle versions accumulate?

Yes. Installing or updating a bundle can leave earlier versions on the machine. Different projects can require different versions. Installing a new version does not require uninstalling the old one, and does not update existing project deployments. See *Supported contexts and versions* for project updates.

To inspect installed versions and remove a selected version, use:

```
cube bundle list
cube bundle remove <bundle-name>@<version>
```

Before removal, check project locks and dependent tools. Do not remove a version still needed by another project. Reinstalling a removed version can require network access. Removing a machine bundle does not remove its deployed project files. See the STM32CubeCLI documentation in SEE ALSO on *STM32CubeCLI and project housekeeping* for bundle lifecycle details.

5.13 Why did updating a bundle not update my project?

Machine installation and project deployment are separate. Updating a context on the machine does not change a project that already records an older version. Follow the project update steps in *Supported contexts and versions*.

5.14 Can the toolkit automatically build or debug with other IDEs?

No. The Build and DAP Debug contexts provide automated workflows only for STM32CubeIDE for Visual Studio Code. They do not automatically build or debug projects in STM32CubeIDE (the Eclipse-based IDE), IAR EWARM, or Keil MDK 5/6. STM32CubeIDE for Visual Studio Code is a separate product and workflow from STM32CubeIDE (Eclipse-based). Use each of those IDEs' own build and debug workflows. Other toolkit contexts remain available where their documented target support allows them.

5.15 Why can the build or debug check fail in VS Code?

Confirm that the project is the workspace root and that STM32CubeIDE for Visual Studio Code has discovered it. Check the setup and debugger requirements in the *Build* and *Debug* guides.

5.16 What should I do when PowerShell shows ^U?

PowerShell 5.1 in the VS Code integrated terminal can display `^U` when Ctrl+U is pressed. Use PowerShell 7.3 or later, or switch the integrated terminal to bash or zsh.

1. Install PowerShell 7.3 or later if it is not already installed.
2. **In VS Code, open the Command Palette and run Terminal: Select Default Profile.**
3. **Select PowerShell (the PowerShell 7 profile), close the existing** terminal, and create a new terminal.
4. Confirm the version with:

```
$PSVersionTable.PSVersion
```

If `^U` appears in the current command line, press Ctrl+C to cancel that input, then type the command again in the new terminal. Alternatively, select a bash or zsh profile from **Terminal: Select Default Profile**.

5.17 Why does MX2 fail in a terminal?

Local shell settings can affect STM32CubeMX2 startup and process output. An `EPIPE: broken pipe` JavaScript popup is a known symptom in this environment. Check the terminal environment before assuming that project configuration is invalid. The popup alone does not establish its cause.

If the first STM32CubeMX2 command reports `Cannot find any backend port.` in a VS Code terminal, check whether `ELECTRON_RUN_AS_NODE` is set. This variable can make the Electron-based application run as Node.js instead of starting its backend. Other startup failures can produce the same error, so inspect the command result and logs.

If the variable is set, remove it for the STM32CubeMX2 process and retry the failed command once. In bash or zsh, prefix that command with `env -u ELECTRON_RUN_AS_NODE`. In PowerShell, remove the variable from the current terminal with `Remove-Item Env:ELECTRON_RUN_AS_NODE` before retrying. Check whether the failed command changed project state before repeating an operation. If the retry fails, preserve the error and investigate backend startup.

5.18 Why did setup stop on an existing deployment?

`setup` is for a project that has not already been deployed by the installer. Use `project-add` or `project-remove` to change contexts in a deployed project, then run `doctor --fix`. The installer user manual describes the project update procedure.