

STM32CubeCLI documentation

Release latest

Oct 01, 2026

Table Of Contents

1	Overview	3
1.1	Why it exists	3
1.2	What is STM32CubeCLI?	3
1.3	What the command surface looks like	3
1.4	Where to go next	4
1.4.1	Licenses	4
1.4.2	Why is STM32CubeCLI released as a beta?	4
2	Getting Started	5
2.1	Installing STM32CubeCLI	5
2.2	Your first project	6
2.2.1	Before you start	6
2.2.2	1. Getting the example project	6
2.2.3	2. Installing the pinned toolchain	7
2.2.4	3. Building the project	8
2.2.5	Where to go next	8
2.3	Installing and using bundles	8
2.3.1	Before you start	8
2.3.2	Example: install CMake	8
2.3.3	Updating bundles	9
2.3.4	Project-scoped installation	10
2.4	Working with a project	10
2.4.1	External bundle locations	10
3	Command reference	12
3.1	Syntax	12
3.2	Global options	12
3.3	Dynamic commands and runtimes	14
3.4	Bundle management	14
3.4.1	Inspect bundles	14
3.4.2	Install and maintain bundles	15
3.4.3	Projects and reproducibility	16
3.4.4	Create bundles	16
3.5	Environment variables	17

1 Overview

Welcome to the documentation portal for **STM32CubeCLI**.
STM32CubeCLI is a platform for discovering, installing, and running tools in the STM32Cube ecosystem.

1.1 Why it exists

STM32CubeCLI is the next generation of command line tools after STM32CubeCLT, evolved from one monolithic package into independently installable bundles. This evolution brings:

	Packaging		One monolithic packa
	Versions		
	Result	Varies: depends on the user's permissions/priv	

1.2 What is STM32CubeCLI?

STM32CubeCLI is made of three parts working together:
The promise behind all three:

“Your tools travel with your code.”

- **Package-based:** install only what a project needs, from a public registry, no login required
- **Low-risk updates:** individually updateable bundles (a bundle is updated together with its dependencies, other bundles with their dependencies are kept untouched, several versions of bundles coexist in the same environment when needed)
- **Day-one ready:** a new colleague is productive from the start — the environment *is* the checkout
- **Guaranteed reproducibility** across:
 - **host OS and architecture:** one lock file resolves the right binary on Windows, Linux, macOS, x86_64 or Arm
 - **projects:** each project pins its own tool versions and can coexist with other projects
 - **every surface:** IDE, CLI, CI and agentic AI all resolve the exact same tools

1.3 What the command surface looks like

The list of commands available on your machine depends on which bundles you have installed. See the [Command reference](#) for the full list of the wrapper commands.

To get a complete list of installed commands, run

```
cube --list
```

To get a complete list of available bundles online, please run

```
cube bundle list-online
```

1.4 Where to go next

- [Getting started](#) — install STM32CubeCLI and build your first reproducible project
- [Command reference](#) — the list of tools and commands available through `cube`
- [Licenses](#) — license terms for STM32CubeCLI and the bundles it installs
- Have a question, a request, or found an issue? Ask on the [ST community forum](#)

1.4.1 Licenses

The STM32CubeCLI software package is provided AS IS. By downloading, installing, copying, accessing, or otherwise using the package, you agree to be bound by the terms of the software license agreement.

On this page

- [SLA0048 software license agreement](#)
- [Third-party licenses](#)

SLA0048 software license agreement

- [Software license agreement](#)

Third-party licenses

1.4.2 Why is STM32CubeCLI released as a beta?

STM32CubeCLI has run inside STM32CubeIDE for Visual Studio Code since **April 2025** and inside STM32CubeMX2 since **March 2026**, on top of five years of internal use before either of those. The wrapper, the bundle manager, and the bundle format are neither new nor experimental.

What's still settling is the **command surface** — the exact names and shape of the commands you type. Today, `cube` commands are named after the underlying tool (`cube cmake`, `cube programmer`, `cube mx`), not after the task. Whether the CLI grows task-level commands instead (for example a single `cube build` or `cube flash`) is still an open decision.

This **documentation portal** is built for continuous deployment — new chapters and fixes ship as real feedback comes in, rather than waiting for everything to be finished up front.

What this means for you

- **The tools you rely on are stable.** Reliability isn't the open question.
- **Exact command names may change before general availability.** A script or CI pipeline that calls a specific command today may need a small update later if that command is renamed or reorganized.
- **Your project's pinned tool versions aren't affected.** The manifest and lock file mechanism — what makes builds reproducible — isn't part of what's changing.
- **Missing something, or found an issue?** [Tell us](#).

See the [Command reference](#) for the current command surface, and the release notes for anything that changes between releases.

Have a question, a request, or found an issue? Ask on the [ST community forum](#).

2 Getting Started

From installing STM32CubeCLI to a reproducible, pinned toolchain in a real project — start here.

- [Installation guide](#)
- [Your first project](#) — pin a tool, install it reproducibly, and build your code with it
- [Installing and using bundles](#) — a deeper look at browsing the catalog, installing, and updating bundles
- [Working with a project](#) — how project lock files make a toolchain reproducible

2.1 Installing STM32CubeCLI

If you have **STM32CubeMX2** or **STM32CubeIDE for VSCode** installed, STM32CubeCLI may already be available in your PATH. Check with `cube --version` before installing it.

Windows (PowerShell)

Windows (Bash)

macOS

Linux

Check whether STM32CubeCLI is installed:

```
cube --version
```

The STM32CubeCLI software package is provided AS IS. By downloading it, you agree to be bound by the terms of the [Software license agreement](#). The detailed content licenses are available [here](#).

If it is not available, install it:

```
irm https://developer.st.com/bundles/install/install.ps1 | iex
```

Open a new PowerShell window after installation, then verify:

```
cube --version
cube --help
```

In Git Bash, WSL, or MSYS2, check whether STM32CubeCLI is installed:

```
cube --version
```

The STM32CubeCLI software package is provided AS IS. By downloading it, you agree to be bound by the terms of the [Software license agreement](#). The detailed content licenses are available [here](#).

If it is not available, install it:

```
curl -fsSL https://developer.st.com/bundles/install/install.sh | bash -
```

Open a new shell after installation, then verify:

```
cube --version
cube --help
```

Check whether STM32CubeCLI is installed:

```
cube --version
```

The STM32CubeCLI software package is provided AS IS. By downloading it, you agree to be bound by the terms of the - [Software license agreement](#). The detailed content licenses are available [here](#).

If it is not available, install it with Bash or zsh:

```
curl -fsSL https://developer.st.com/bundles/install/install.sh | bash -
```

Open a new shell after installation, then verify:

```
cube --version
cube --help
```

Check whether STM32CubeCLI is installed:

```
cube --version
```

The STM32CubeCLI software package is provided AS IS. By downloading it, you agree to be bound by the terms of the - [Software license agreement](#). The detailed content licenses are available [here](#).

If it is not available, install it with Bash:

```
curl -fsSL https://developer.st.com/bundles/install/install.sh | bash -
```

Open a new shell after installation, then verify:

```
cube --version
cube --help
```

2.2 Your first project

This walkthrough clones a ready-made example project, installs its pinned toolchain, and builds it, with no board required. By the end, you'll have used **cube** to reproduce someone else's exact toolchain on your machine and build real STM32 firmware from it, without installing or configuring a compiler yourself.

2.2.1 Before you start

Install STM32CubeCLI by following the [installation guide](#), then check that the **cube** command is available:

```
cube --version
```

Note

Already using STM32CubeIDE for VS Code with a project that has `.settings/bundles.store.json` committed? Run `cube bundle install --project` from the project's root — that's the whole tutorial for you. The rest of this page walks through cloning and building an example project from scratch.

2.2.2 1. Getting the example project

Clone the tutorial repository:

```
git clone https://github.com/stm32-hotspot/STM32CubeCLI-basic-tutorial.git
```

Then change into the project directory:

```
cd STM32CubeCLI-basic-tutorial
```

The project targets an STM32F407VGTx as a custom target, generated by STM32CubeMX with CMake presets. Its top-level layout looks like this:

```
STM32CubeCLI-basic-tutorial/  
  .settings/  
    bundles.store.json  
    bundles-lock.store.json  
  Core/  
  Drivers/  
  cmake/  
  CMakeLists.txt  
  CMakePresets.json  
  startup_stm32f407xx.s  
  STM32F407xx_FLASH.ld  
  ...
```

The cloned project already pins its toolchain — `bundles.store.json` and `bundles-lock.store.json` are committed, so the exact same versions resolve on any machine. See [Working with a project](#) for the explanation of what these files contain.

2.2.3 2. Installing the pinned toolchain

Run the following command to install the pinned toolchain:

```
cube bundle install --project
```

This reads the project's lock data and installs the exact bundle versions it requires — CMake, Ninja, the GNU Tools for STM32, and clangd support for editor tooling — without you installing or configuring any of them yourself. The project dependencies can be found in `.settings/bundles.store.json`:

```
{  
  "bundles": [  
    {  
      "name": "cmake",  
      "version": "4.3.1+st.1"  
    },  
    {  
      "name": "ninja",  
      "version": "1.13.2+st.1"  
    },  
    {  
      "name": "gnu-tools-for-stm32",  
      "version": "14.3.1+st.2"  
    },  
    {  
      "name": "st-arm-clangd",  
      "version": "21.1.0+st.2"  
    }  
  ]  
}
```

To view all the bundles used by the project, run:

```
cube bundle list --project
```

2.2.4 3. Building the project

Configure the Debug preset:

```
cube cmake --preset Debug
```

Then build it:

```
cube cmake --build --preset Debug
```

A successful build produces a firmware image in `build/Debug/`:

```
STM32CubeCLI-basic-tutorial/  
  .settings/  
  build/  
    Debug/  
      STM32CubeCLI-basic-tutorial.elf  
    ...  
  Core/  
  Drivers/  
  cmake/  
  CMakeLists.txt  
  CMakePresets.json  
  ...
```

The commands above used `cube` to install the required toolchain and build tools automatically and then use them to build the project. Anyone who clones this project runs the same commands and gets the same toolchain, producing the same firmware image, on any machine.

From here, two natural next steps: running this same install-and-build sequence inside a CI pipeline instead of on your machine, or packaging your own tools as a bundle so others can pin them the same way. Documentation for both is coming; in the meantime, the pages below cover what's already available.

2.2.5 Where to go next

- [Installing and using bundles](#) — a deeper look at browsing the catalog, installing, and updating bundles
- [Working with a project](#) — what the project's lock files contain and why they make builds reproducible
- [Command reference](#) — the full `cube` and `cube bundle` syntax

2.3 Installing and using bundles

STM32CubeCLI delivers tools and resources as bundles. Use the `cube` command line tool to browse the online bundle catalog, install a bundle, and access its commands.

2.3.1 Before you start

Install STM32CubeCLI by following the [installation guide](#), then check that the `cube` command is available:

```
cube --version
```

2.3.2 Example: install CMake

The `cmake` bundle provides the CMake command-line interface. We are going to view available online versions, and then install one of them.

1. List the bundles available in the online catalog:

```
cube bundle list-online
```

Sample output:

Name	: Platform	Version
cmake	: x86_64-windows	["4.3.1+st.1", "4.2.3+st.1", "4.0.1+st.3"]

The output includes the bundle name, supported platform, and available versions.

2. Install the `cmake` bundle:

```
cube bundle install cmake
```

The command resolves the bundle, downloads it, and installs it. By default, it installs the latest available version:

```
Need to download the following bundles:
  cmake : x86_64-windows ["4.3.1+st.1"]
Total download size is 50.39 MiB, total approximate install size is 132.72 MiB
Installing cmake@4.3.1+st.1...
Installed cmake@4.3.1+st.1 successfully, 132.72 MiB used
Installation has been completed.
```

3. Verify that the bundle is installed:

```
cube bundle list
```

The list of installed bundles includes information about bundle version and installation location:

```
Installed bundles:
...
cmake@4.3.1+st.1 <bundle-installation-path>
cmake@4.0.1+st.3 <bundle-installation-path>
```

4. Display the CMake command-line help:

```
cube cmake --help
```

The help displays the CMake command-line interface:

```
Usage
cmake [options] <path-to-source>
cmake [options] <path-to-existing-build>
cmake [options] -S <path-to-source> -B <path-to-build>
```

2.3.3 Updating bundles

Update a specific installed bundle by providing its name:

```
cube bundle update cmake
```

If no newer version is available, the command reports:

```
Everything is up to date.
```

To update all installed bundles, omit the bundle name:

```
cube bundle update
```

For a project-scoped bundle, run the install from the project directory with the `--project` option:

```
cube bundle install --project cmake
```

2.3.4 Project-scoped installation

To install a bundle for the current project, run the command from the project directory with the `--project` option:

```
cube bundle install --project cmake
```

After installation, list the bundles in the current project context:

```
cube bundle list --project
```

The default install and list commands use the global context. Use the project-scoped alternatives when the bundle should be available only to one project. Use `cube bundle list-online` to inspect the catalog regardless of the installation context.

For non-interactive scripts, add `--yes` to automatically accept the install confirmation:

```
cube bundle --yes install cmake
```

2.4 Working with a project

A STM32CubeCLI project records its required tool bundles so developers and automation use the same toolchain. STM32CubeCLI searches the current directory and its parents for `.settings`. Use `--cwd <directory>` to select a project elsewhere.

The project contains two files:

- `.settings/bundles.store.json` records direct requirements and version ranges.
- `.settings/bundles-lock.store.json` records exact bundle and dependency versions for each platform.

Commit both files. Compatible lock data selects command versions and missing bundles to install, making the toolchain reproducible while those versions remain available.

```
cube bundle init --project cmake@~3.30.0
cube bundle show --project
```

After cloning a project, run the following command to **install its locked toolchain**:

```
cube bundle install --project
```

See [Your first project](#) for a hands-on walkthrough of this flow, and the [Command reference](#) for the full `cube bundle` syntax.

2.4.1 External bundle locations

External bundle locations let `cube` discover bundles that are managed outside the standard `CUBE_BUNDLE_PATH` repository. Register a directory to use bundles in place, without copying or installing them into the standard repository. While developing a bundle, this lets you test changes directly from its working directory without repeatedly packaging and installing it. External locations can also contain bundles maintained by another tool or in a shared workspace.

external add <directory>...

Register directories containing externally managed bundles. A directory is recorded even if it does not currently exist.

Linux / macOS

Windows

```
cube bundle external add ~/cube-bundles/custom-tool
```

```
cube bundle external add C:\CubeBundles\custom-tool
```

external remove <directory>...

Stop loading bundles from the specified external directories. Bundle files are not deleted.

Linux / macOS

Windows

```
cube bundle external remove ~/cube-bundles/custom-tool
```

```
cube bundle external remove C:\CubeBundles\custom-tool
```

external list

List registered external directories.

Example: `cube bundle external list`

3 Command reference

The `cube` command runs STM32CubeCLI tools and manages the bundles that provide them. Its command surface changes when bundles are installed, removed, or selected by a project.

3.1 Syntax

```
cube [GLOBAL_OPTION]... [COMMAND] [ARGUMENT]...  
cube bundle [BUNDLE_OPTION]... <OPERATION> [OPERATION_OPTION]... [ARGUMENT]...
```

Run `cube --help` to see the commands available in the current environment. Run `cube <command> --help` or `cube bundle <operation> --help` for command-specific options.

3.2 Global options

--help

Show help, including the commands provided by loaded bundles.

Example: `cube --help`

--help-license

Show the license agreement for `cube`.

Example: `cube --help-license`

--version

Show the `cube` version.

Example: `cube --version`

--cwd <directory>

Run as if started in `directory`. This also controls project discovery.

Linux / macOS

Windows

```
cube --cwd ~/projects/blinkly bundle show --project
```

```
cube --cwd C:\Projects\blinkly bundle show --project
```

--json

Use JSON output where the selected operation supports it.

Example: `cube --json bundle list`

The inspection operations `list`, `show`, `license`, `list-deps`, `list-usages`, `list-online`, `list-all-online`, and `external list` support JSON output. Place `--json` before `bundle`.

--verbose

Show additional wrapper diagnostics.

Example: `cube --verbose bundle list`

--list

List loaded bundles.

Example: `cube --list`

--resolve <command> [arguments]

Show the executable, providing bundle, and exported environment without running the command. Combine with `--json` for machine-readable output.

Examples: `cube --resolve cmake --version` and `cube --json --resolve cmake --version`

--with <version-range>

Select a bundle version matching the range for the following command or runtime.

Example: `cube --with ^3.30.0 cmake --version`

--detach

Start the selected command as a detached process.

Example: `cube --detach http-server`

--no-detach

Keep the selected command attached, even if its bundle requests detachment.

Example: `cube --no-detach http-server`

--show-settings

Show all `cube` configuration values.

Example: `cube --show-settings`

--get-current-value <key>

Show the effective value of a configuration key.

Example: `cube --get-current-value cmsis_pack_root`

--set <key> <value>

Save a value in `CubeUserConfig.json` under the platform's user configuration directory. The default location is `$XDG_CONFIG_HOME/stm32cube/CubeUserConfig.json` on Linux when `XDG_CONFIG_HOME` is set, or `~/.config/stm32cube/CubeUserConfig.json` otherwise; `%APPDATA%\stm32cube\CubeUserConfig.json` on Windows; and `~/Library/Application Support/stm32cube/CubeUserConfig.json` on macOS. Run `cube --verbose --show-settings` to see the file used by the current environment.

The wrapper recognizes these keys:

cmsis_pack_root

CMSIS Pack storage directory. Example: `cube --set cmsis_pack_root /path/to/packs`

cube_bundle_path

Installed bundle repository directory. Example: `cube --set cube_bundle_path /path/to/bundles`

cube_bundle_path_extra

Additional directory searched recursively for bundles. Example: `cube --set cube_bundle_path_extra /path/to/extra-bundles`

cube_bundle_registry

Online bundle registry URL. Example: `cube --set cube_bundle_registry https://example.com/bundles`

cube_cache_path

Cache directory. Example: `cube --set cube_cache_path /path/to/cache`

cube_log_path

Log directory. Example: `cube --set cube_log_path /path/to/logs`

cube_network_max_redirect

Maximum number of network redirects. Example: `cube --set cube_network_max_redirect 10`

cube_network_retry

Number of network retry attempts. Example: `cube --set cube_network_retry 5`

cube_network_timeout

Network timeout in milliseconds. Example: `cube --set cube_network_timeout 7200000`

--unset <key>

Remove a configuration value.

Example: `cube --unset cube_network_retry`

3.3 Dynamic commands and runtimes

Bundles declare commands and runtimes in `CubeBundle.json`. `cube` loads installed bundles and bundles under `CUBE_BUNDLE_PATH_EXTRA` each time it starts. Bundle commands appear under `COMMANDS` in `cube --help` and run as `cube <command> [arguments]`.

Runtimes are executables used internally by bundle commands. They do not appear as commands in `cube --help` and cannot be invoked directly. When a command uses a runtime, `cube` resolves the runtime from that command's bundle dependencies.

The available commands and selected versions depend on:

- installed bundle versions;
- bundles under `CUBE_BUNDLE_PATH_EXTRA`;
- versions selected by the current project's lock data;
- `--with <version-range>`, when specified.

Use these commands to inspect the active command surface and its resolution:

```
cube --help
cube --list
cube --resolve cmake
cube --json --resolve cmake
```

For example, `cube cmake --version` runs the `cmake` command from the selected bundle. If that command uses a runtime, `cube` resolves and runs the runtime automatically.

Installing or removing a bundle updates the available commands on the next `cube` invocation. If several installed versions provide the same command, project lock data and `--with` take precedence. Otherwise, `cube` selects the latest matching version.

3.4 Bundle management

A bundle identifier can be a name, an exact version such as `cmake@4.4.0+st.1`, or a version range such as `cmake@^3.30.0`. Use `cube bundle <operation> --help` for the complete syntax accepted by the installed version.

Examples: `cube bundle show cmake`, `cube bundle show cmake@4.4.0+st.1`, and `cube bundle show cmake@^3.30.0`

3.4.1 Inspect bundles

list

List installed bundles. With `--project`, list the current project's direct requirements and resolved bundles instead of every bundle installed in the local repository.

Examples: `cube bundle list` and `cube bundle list --project`

show <bundle>...

Show details for installed bundles. With `--project`, show the current project's direct requirements and exact resolved bundles instead of accepting bundle identifiers.

Examples: `cube bundle show cmake` and `cube bundle show --project`

license <bundle>...

Show bundle license information. With `--project`, show license information for the bundles resolved by the

current project instead of accepting bundle identifiers. With `--online`, show license information from the registry instead of installed bundles.

Examples: `cube bundle license cmake`, `cube bundle license --project`, and `cube bundle license --online cmake`

list-deps <bundle>...

List the dependencies selected for bundle identifiers.

Example: `cube bundle list-deps cmake@^3.30.0`

list-usages <bundle>...

List installed bundles that use the specified bundles.

Example: `cube bundle list-usages cmake`

list-online

List bundles available for this host from the configured registry.

Example: `cube bundle list-online`

list-all-online

List all bundles in the configured registry, including bundles for other platforms or OSs.

Example: `cube bundle list-all-online`

Use `--tags <tag,...>` with online listing operations to filter by bundle tags.

Example: `cube bundle --tags build list-online`

3.4.2 Install and maintain bundles

The install, download, update, remove, and autoremove commands may ask for confirmation. Place `--yes` before the operation to accept automatically. Place `--no` after the operation to decline automatically and inspect the proposed action without applying it.

Examples: `cube bundle --yes install cmake` and `cube bundle install --no cmake`

Install or download

install <bundle-or-path>...

Download and install bundles from identifiers or install local `.bundle` files. Required dependencies are installed automatically.

Example: `cube bundle install cmake@^3.30.0`

With `--project` and bundle identifiers, add those bundles as direct project requirements, resolve and update the lock data, and install anything missing. With `--project` and no identifiers, install missing bundles from the existing project requirements and compatible lock data without adding requirements. `--strict` stops the installation if it requires a bundle that was not explicitly requested. These two options cannot be combined.

Examples: `cube bundle install --project`, `cube bundle install --project cmake`, and `cube bundle install --strict cmake`

download <bundle>...

Download bundle archives and all their dependencies to the current directory. Installed bundles are downloaded again.

Example: `cube bundle download cmake`

`--platform` selects an operating system or a complete architecture and operating-system target. Without it, the current platform is used. With `--project`, also download the current project's locked bundles, or resolve its requirements if no lock data exists. Any supplied bundle identifiers are included in the download but are not added to the project.

Examples: `cube bundle download --platform x86_64-windows cmake` and `cube bundle download --project`

Update or repair

update [bundle]...

Update the named bundles when newer matching versions are available. Without identifiers, check every locally installed bundle.

Examples: `cube bundle update cmake` and `cube bundle update`

fix

Install dependencies missing from any installed bundle.

Example: `cube bundle fix`

Remove

remove <bundle>...

Remove installed bundles matching each identifier.

Example: `cube bundle remove cmake@4.4.0+st.1`

If an identifier matches several versions, specify a narrower version or place `--all` before `remove` to remove every match. With `--project`, remove the named direct requirements from the current project and re-resolve its lock data without deleting installed bundle files.

Examples: `cube bundle --all remove cmake` and `cube bundle remove --project cmake`

autoremove [bundle]...

Remove unused versions of the named bundles. Without identifiers, check every installed bundle. The latest installed version and versions required by other bundles are always kept.

Examples: `cube bundle autoremove cmake` and `cube bundle autoremove`

3.4.3 Projects and reproducibility

See [Working with a project](#) for how project lock data makes the toolchain reproducible, and how to register external bundle locations.

Place `--force` before an operation to continue installation when dependencies are unavailable, or to overwrite conflicting installation and removal work data. Place `--all` before `remove` to remove every installed version matching an ambiguous identifier.

Examples: `cube bundle --force install cmake` and `cube bundle --all remove cmake`

3.4.4 Create bundles

init

Create a `CubeBundle.json` template in the current directory. With `--project`, create `.settings/bundles.store.json` and initialize project lock data instead of creating a bundle template. Bundle identifiers supplied after `--project` become direct requirements and are installed as part of initialization.

Examples: `cube bundle init` and `cube bundle init --project cmake`

validate

Validate `CubeBundle.json` in the current directory.

Example: `cube bundle validate`

package

Package the bundle described by `CubeBundle.json` in the current directory.

Example: `cube bundle package`

show-bundle-schema

Print the JSON schema for `CubeBundle.json`.

Example: `cube bundle show-bundle-schema`

3.5 Environment variables

Environment variables apply to the current process and take precedence over values saved with `cube --set`. Use the `cube --set` command to save a value in the user configuration instead.

CUBE_BUNDLE_PATH

Override the installed bundle repository location.

CUBE_BUNDLE_PATH_EXTRA

Add a recursively searched directory of bundles without installing them.

CUBE_BUNDLE_REGISTRY

Override the online bundle registry URL.

CMSIS_PACK_ROOT

Override the CMSIS Pack storage directory.

CUBE_CACHE_PATH

Override the cache directory.

CUBE_LOG_PATH

Override the log directory.

CUBE_NETWORK_MAX_REDIRECT

Set the maximum number of network redirects.

CUBE_NETWORK_RETRY

Set the number of network retry attempts.

CUBE_NETWORK_TIMEOUT

Set the network timeout in milliseconds.

Examples:

Linux / macOS (Bash)

Windows (PowerShell)

Windows (Command Prompt)

User configuration

```
export CUBE_BUNDLE_PATH="$HOME/.local/share/stm32cube/bundles"
export CUBE_BUNDLE_PATH_EXTRA="$HOME/cube-bundles"
export CUBE_BUNDLE_REGISTRY="https://example.com/bundles"
export CMSIS_PACK_ROOT="$HOME/.local/share/stm32cube/packs"
export CUBE_CACHE_PATH="$HOME/.cache/stm32cube"
export CUBE_LOG_PATH="$HOME/.config/stm32cube/logs"
export CUBE_NETWORK_MAX_REDIRECT=10
export CUBE_NETWORK_RETRY=5
export CUBE_NETWORK_TIMEOUT=7200000
```

```
$env:CUBE_BUNDLE_PATH = "$HOME\AppData\Local\stm32cube\bundles"
$env:CUBE_BUNDLE_PATH_EXTRA = "C:\CubeBundles"
$env:CUBE_BUNDLE_REGISTRY = "https://example.com/bundles"
$env:CMSIS_PACK_ROOT = "$HOME\AppData\Local\stm32cube\packs"
$env:CUBE_CACHE_PATH = "$HOME\AppData\Local\stm32cube\cache"
$env:CUBE_LOG_PATH = "$HOME\AppData\Local\stm32cube\logs"
$env:CUBE_NETWORK_MAX_REDIRECT = "10"
$env:CUBE_NETWORK_RETRY = "5"
$env:CUBE_NETWORK_TIMEOUT = "7200000"
```

```
set "CUBE_BUNDLE_PATH=%LOCALAPPDATA%\stm32cube\bundles"
set "CUBE_BUNDLE_PATH_EXTRA=C:\CubeBundles"
```

(continues on next page)

(continued from previous page)

```
set "CUBE_BUNDLE_REGISTRY=https://example.com/bundles"
set "CMSIS_PACK_ROOT=%LOCALAPPDATA%\stm32cube\packs"
set "CUBE_CACHE_PATH=%LOCALAPPDATA%\stm32cube\cache"
set "CUBE_LOG_PATH=%LOCALAPPDATA%\stm32cube\logs"
set "CUBE_NETWORK_MAX_REDIRECT=10"
set "CUBE_NETWORK_RETRY=5"
set "CUBE_NETWORK_TIMEOUT=7200000"
```

Save persistent values with `cube --set` instead of setting environment variables:

```
cube --set cube_bundle_path /path/to/bundles
cube --set cube_bundle_path_extra /path/to/extra-bundles
cube --set cube_bundle_registry https://example.com/bundles
cube --set cmsis_pack_root /path/to/packs
cube --set cube_cache_path /path/to/cache
cube --set cube_log_path /path/to/logs
cube --set cube_network_max_redirect 10
cube --set cube_network_retry 5
cube --set cube_network_timeout 7200000
```

The effective values and defaults are shown by `cube --help` and `cube --show-settings`.