# Configuring Build Steps

When creating a build configuration, it is important to configure the sequence of build steps to be executed.

Build steps are configured on the Build Steps section of the Build Configuration Settings page: the steps can be auto-detected by TeamCity or added manually.

Each build step is represented by a build runner and provides integration with a specific build or test tool. You can add as many build steps to your build configuration as needed. For example, call a NAnt script before compiling VS solutions.

Build steps are invoked sequentially.

The decision whether to run the next build step may depend on the exit status of the previous build steps and the current build status.

The build step status is considered failed if the build process returned a non-zero exit code and the Fail build if build process exit code is not zero build failure condition is enabled (see Build Failure Conditions); otherwise build step is successful.

Note that the status of the build step and the build can be different. A build step can be successful, but the build can be failed because of another build failure condition, not based on the exit code (like failing a test or something else). On the other hand, if a build step has failed, the build will be failed too.

## Execution policy

You can specify the step execution policy via the Execute step option:

- Only if build status is successful: before starting the step, the build agent requests the build status from the server, and skips the step if the status is failed. This considers the failure conditions processed by the server, like failure on test failures or on metric change. Note that this still can be not exact as some failure conditions are processed on the server asynchronously (TW-17015)
- If all previous steps finished successfully (default): the build analyzes only the build step status on the build agent, and doesn't send a request to the server to check the build status and considers only important step failures, i.e. failures on test failures or on metric change are not considered.
- Even if some of previous steps failed: select to make TeamCity execute this step regardless of the status of previous steps and status of the build.
- Always, even if build stop command was issued: select to ensure this step is always executed, even if the build was canceled by a user. For example, if you have two steps with this option configured, stopping the build during the first step execution will interrupt this step, while the second step will still run. Issuing the stop command for the second time will result in ignoring the execution policy: the build will be terminated.

- You can copy a build step from one build configuration to another from the original build configuration settings page.
- You can reorder build steps as needed. Note, that if you have a build configuration inherited from a template, you cannot reorder inherited build steps. However, you can insert custom build steps (not inherited) at any place and in any order, even before or between inherited build steps. Inherited build steps can be reordered in the original template only.
- You can disable a build step temporarily or permanently, even if it is inherited from a build configuration template using the corresponding option in the last column of the Build Steps list.

## Bundled runners

For the details on configuring individual build steps, refer to:

- .NET CLI (dotnet)
- .NET Process Runner
- Ant
- Command Line
- Deployers
  - Container Deployer
  - FTP Upload
  - SMB Upload
  - SSH Exec
  - SSH Upload
- Docker Build
- Docker Compose
- Duplicates Finder (.NET)
- Duplicates Finder (Java)
- FxCop

- Gradle
- Inspections
- Inspections (.NET)
- IntelliJ IDEA Project
- Ipr (deprecated)
- Maven
- MSBuild
- MSpec
- MSTest
- NAnt
- NuGet
    - NuGet Installer
    - NuGet Pack
    - NuGet Publish
- NUnit
- PowerShell
- Rake
- Simple Build Tool (Scala)
- Visual Studio (sln)
- Visual Studio 2003
- Visual Studio Tests
- Working with Meta-Runner
- Xcode Project

See also:

Concepts: Build Runner