Kotlin DSL

Besides storing settings in version control in XML format, TeamCity also allows storing the settings in the DSL (based on the Kotlin language).

Using the version control-stored DSL enables you to define settings programmatically. Since Kotlin is statically typed, you automatically receive the auto-completion feature in IDE which makes the discovery of available API options a much simpler task.

Check also our blog post series on Kotlin DSL.

On this page:

Getting Started with Kotlin DSL

The Kotlin tutorial helps you learn most Kotlin features in a few hours.

You can start working with Kotlin DSL by creating an empty sandbox project on your server and follow the steps below:

1. Enable versioned settings for your project.
2. Select Kotlin as the format. Make sure the Generate portable DSL scripts option is enabled.
3. Click Apply and TeamCity will commit the settings to your repository.

> ⚠ After enabling Kotlin for project settings, editing of the project in the web UI may not be available right after the switch. TeamCity needs to detect its own commit in the repository, and only after that editing will be enabled. Usually it takes a minute or two.

Project Settings Structure

After the commit to the repository, you will get the the .teamcity settings directory with the following files:

- pom.xml
- settings.kts

settings.kts is the main file containing all the project configuration, that's the file we will use to change project settings. pom.xml is only required to open the project in an IDE to get the auto-completion feature as well as ability to compile code, write unit tests for it, etc.

Opening Project in IntelliJ IDEA

To open the Kotlin DSL project in IntelliJ IDEA, open the .teamcity/pom.xml file as a project. All necessary dependencies will be resolved automatically right away. If all dependencies have been resolved, no errors in red will be visible in the settings.kts. If you already have an IntelliJ IDEA project and want to add Kotlin DSL module to it, see this section.

Editing Kotlin DSL

If you created an empty project, that's what you'll see in your IDE when you open settings.kts:

```
import jetbrains.buildServer.configs.kotlin.v2018_1.*
/* some comment text  */
version = "2018.1"

project {
}
```

`project { }` represents the current project whose settings we'll define in the DSL. This is the same project where we enabled versioned settings on the previous step. This project ID and name can be accessed via a special `DslContext` object but cannot be changed via the DSL.

You can create different entities in this project by calling `vcsRoot()`, `buildType()`, `template()`, or `subProject()` methods.

For instance, to add a build configuration with a command line script, we can do the following:

```
import jetbrains.buildServer.configs.kotlin.v2018_1.*
import jetbrains.buildServer.configs.kotlin.v2018_1.buildSteps.script

version = "2018.1"

project {
  buildType {
    id("HelloWorld")
    name = "Hello world"
    steps {
        script {
            scriptContent = "echo 'Hello world!'"
        }
    }
  }
}
```

After that you can submit this change to the repository, TeamCity will detect it and apply. If there are no errors during the script execution, then we should see a build configuration named "Hello world" in our project.

To get familiar with Kotlin API, see the online documentation on your local server, accessible via the link on the Versio ned Settings project tab in the UI or by running the `mvn -U dependency:sources` command in the IDE. See the documentation on the public TeamCity server as an example.

The documentation is generated in a separate Java process and might take several minutes to build after the server restart.

You can also use the Download settings in Kotlin format option from the project Actions menu. For instance, you can find a project that defines some settings that you want to use in your Kotlin DSL project and use this "download" action to see what the DSL generated by TeamCity looks like.

Patches

TeamCity allows editing of a project via the web interface, even though the project settings are stored in Kotlin DSL. For every change made in the project via the web interface, TeamCity will generate a patch in the Kotlin format, which will be checked in under the project patches directory with subdirectories for different TeamCity entities. For example, if you change a build configuration, TeamCity will submit the .teamcity/patches/buildTypes/<id>.kt script to the repository with necessary changes.

For instance, the following patch adds the Build files cleaner (Swabra) build feature to the build configuration with the ID

SampleProject_Build:

```
changeBuildType(RelativeId("SampleProject_Build")) { // this part finds the build configuration where the change has to be done
    features {
        add {
            // this is the part which should be copied to a corresponding place of the settings.kts file
            swabra {
                filesCleanup = Swabra.FilesCleanup.DISABLED
            }
        }
    }
}
```

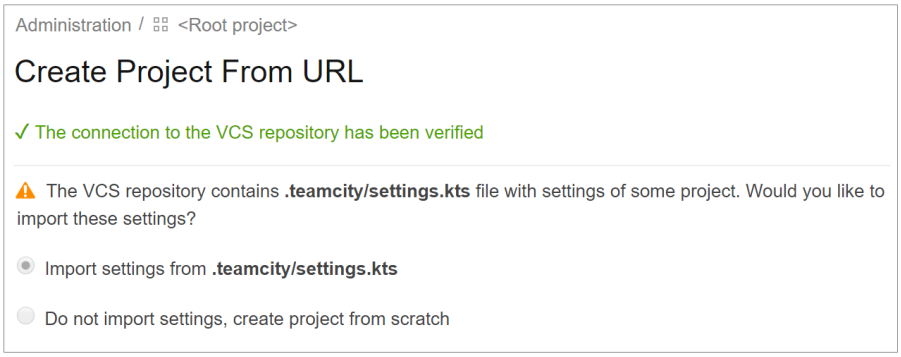
It is implied that you move the changes from the patch file to you settings.kts and delete the patch file. Patches generation allows smooth transition from editing settings via UI to Kotlin DSL.

Sharing Kotlin DSL Scripts

Besides simplicity, one of the advantages of the portable DSL script is that the script can be used by more than one project or more than one server (hence the name: portable).

If you have a repository with .teamcity containing settings in portable format, then you can easily create another project based on these settings. The Create Project From URL feature can be used for this.

Simply point TeamCity to your repository and it will detect the .teamcity directory and offer to import settings from there:



⚠ The settings.kts script can always access a `DslContext` object which contains the id and some other settings of the current project.

Based on the context, the DSL script can generate slightly different settings, for instance:

```
var deployTarget = if (DslContext.projectId == AbsoluteId("Trunk")) {
    "staging"
} else {
    "production"
}
```

Advanced Topics

Non-Portable DSL

Versions before 2018.1 used a different format for Kotlin DSL settings. This format can still be enabled by turning off the Gene rate portable DSL scripts checkbox on versioned settings page.

When TeamCity generates non-portable DSL, the project structure in the .teamcity directory looks as follows:

- pom.xml
- <project id>/settings.kts
- <project id>/Project.kt
- <project id>/buildTypes/<build conf id>.kt
- <project id>/vcsRoots/<vcs root id>.kt

where <project id> is the ID of the project where versioned settings are enabled. The Kotlin DSL files producing build configurations and VCS roots are placed under the corresponding subdirectories.

settings.kts

In the non-portable format each project has the following settings.kts file:

```
package MyProject
import jetbrains.buildServer.configs.kotlin.v2018_1.*
/* ... */
version = "2018.1"

project(MyProjectId.Project)
```

This is the entry point for project settings generation. Basically, it represents a Project instance which generates project settings.

Project.kt

The Project.kt file looks as follows:

```
package MyPackage
import jetbrains.buildServer.configs.kotlin.v2018_1.*
import jetbrains.buildServer.configs.kotlin.v2018_1.Project

object Project : Project({
   uuid = "05acd964-b90f-4493-aa09-c2229f8c76c0"
   id("MyProjectId")
   parentId("MyParent")
   name = "My project"
   ...
})
```

where:

- `id` is the absolute id of the project, the same id we'll see in browser address bar if we navigate to this project
- `parentId` is the absolute id of a parent project where this project is attached
- `uuid` is some unique sequence of characters.
  The `uuid` is a unique identifier which associates a project, build configuration or VCS root with its data. If the `uuid` is changed, then the data is lost. The only way to restore the data is to revert the `uuid` to the original value. On the other hand, the `id` of an entity can be changed freely, if the `uuid` remains the same. This is the main difference of the non-portable DSL format from portable. The portable format does not require specifying the `uuid`, but if it happened so that a build configuration lost its history, one has to reattach it again via the web interface.

  > ⚠ If the build history is important, it should be restored as soon as possible: after the deletion, there is a configurable timeout (5 days by default) before the builds of the deleted configuration are removed during the build history clean-up.

In case of non-portable DSL, patches are stored under the project patches directory of .teamcity:

- pom.xml
- <project id>/settings.kts
- <project id>/Project.kt

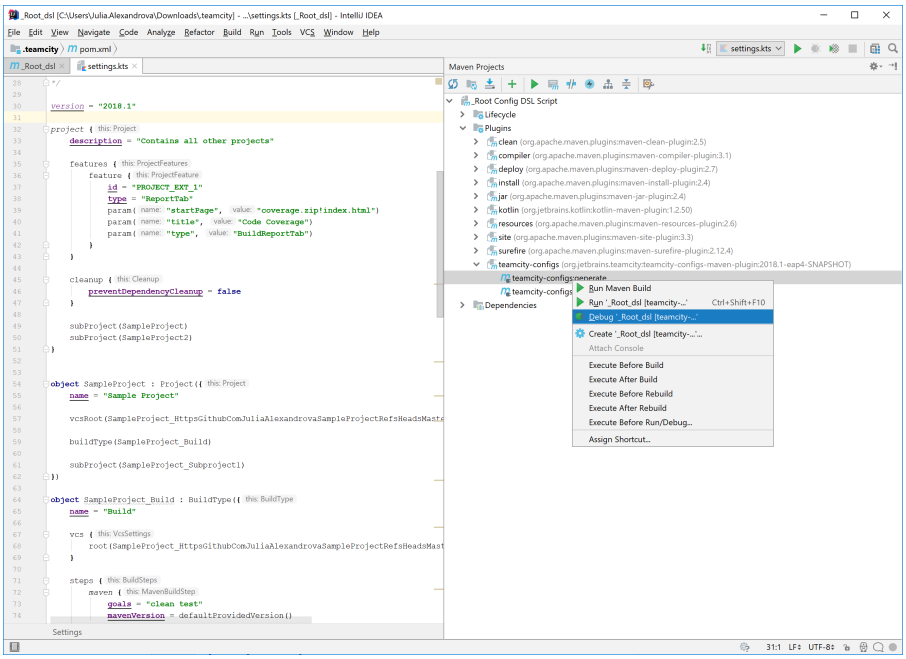
- <project id>/patches/<uuid>.kt

Working with patches is the same as in portable DSL: you need to move the actual settings from the patch to your script and remove the patch.

Debugging Maven 'generate' Task

The pom.xml file provided for a Kotlin project has the `generate` task, which can be used to generate TeamCity XML files locally from the Kotlin DSL files. This task supports debugging. If you're using IntelliJ IDEA, you can easily start debugging of a Maven task:

1. Navigate to View | Tool Windows | Maven Projects. The Maven Projects tool window is displayed.
2. Locate the task node: Plugins | teamcity-configs | teamcity-configs:generate, the Debug option is available in the context menu for the task:



Ability to Use External Libraries

You can use external libraries in your Kotlin DSL code, which allows sharing code between different Kotlin DSL-based projects.

To use an external library in your Kotlin DSL code, add a dependency on this library to the .teamcity/pom.xml file in the settings repository and commit this change so that TeamCity detects it. Then, before starting the generation process, the TeamCity server will fetch the necessary dependencies from the Maven repository, compile code with them, and then start the settings generator.

View DSL in UI

There is an option in the UI to help those using Kotlin-based DSL. When viewing your build configuration settings in the UI, you can click View DSL in the sidebar: the DSL representation of the current configuration will be displayed and the setting being viewed (e.g. a build step, a trigger, dependencies) will be highlighted. To go back, click Edit in UI.

FAQ and Common Problems

Why portable DSL requires the same prefix for all IDs?

In TeamCity projects, templates, build configurations and VCS roots all have unique IDs. These IDs usually look like:

<parent project id>_<entity id>

Since these IDs must be unique, there cannot be two different entities in the system with the same ID.

But one of the reasons why portable DSL is called portable is because the same settings.kts script can be used to generate settings for two different projects even on the same server.

In order to achieve this and overcome IDs uniqueness, TeamCity operates with relative IDs in portable DSL scripts. These relative IDs do not have parent project ID prefix in them. So when TeamCity generates portable Kotlin DSL scripts, it has to remove the parent project ID prefix from the IDs of all of the generated entities.

But this will not work if not all of the project entities have this prefix in their IDs. In this case the following error can be shown:

Build configuration id '<some id>' should have a prefix corresponding to its parent project id: '<parent project id>_'

To fix this problem, one should change the ID of the affected entity. If there are many such IDs, Bulk edit IDs project action can be used to change all of them at once.

How to Add .teamcity as a New Module to a Project?

Question: How to add the .teamcity settings directory as a new module to an existing project in IntelliJ IDEA?

Solution: In your existing project in IntelliJ IDEA:

- Go to File | Project Structure, or press Ctrl+Shift+Alt+S.
- Select Modules under the Project Settings section.
- Click the plus sign, select Import module and choose the folder containing your project settings. Click Ok and follow the wizard.
- Click Apply. The new module is added to your project structure.

New URL of Settings VCS Root (Non portable format)

Problem: I changed the URL of the VCS root where settings are stored in Kotlin and now TeamCity cannot find any settings in the repository at the new location.

Solution:

- Fix the URL in the Kotlin DSL in the version control and push the fix.
- Disable versioned settings to enable the UI.
- Fix the URL in the VCS root in the UI.
- Enable versioned settings with same VCS root and the Kotlin format again, TeamCity will detect that the repository contains the .teamcity directory and ask you if you want to import settings.
- Choose to import settings.

How to Read Files in Kotlin DSL

Problem: I want to generate a TeamCity build configuration based on the data in some file residing in the VCS inside the .team city directory

Solution: TeamCity executes DSL with the .teamcity as the current directory, so files can be read using the paths relative to the .teamcity directory e.g. File("data/setup.xml"). Files outside the .teamcity directory are not accessible to Kotlin DSL.

Kotlin DSL API documentation is not initialized yet

Problem:

- app/dsl-documentation/index.html on our Teamcity server displays "Kotlin DSL API documentation is not initialized yet"
- OutOfMemoryError during TeamCity startup with `org.jetbrains.dokka` in stack trace

Solution: set the internal property `teamcity.kotlinConfigsDsl.docsGenerationXmx=768m`

Passwords-Related Questions

Prior to TeamCity 2017.1

Problem: I do not want the passwords to be committed to the VCS, even in a scrambled form.

Solution: You can move the passwords to the parent project whose settings are not committed to a VCS.

Problem: I want to change passwords after the settings have been generated.

Solution: The passwords will have to be scrambled manually using the following command in the maven project with settings:

```
mvn -Dtext=<text to scramble> org.jetbrains.teamcity:teamcity-configs-maven-plugin:scramble
```

Since TeamCity 2017.1

Solution: Use tokens instead of passwords. Please refer to the related section.

See also:

- Administrator's Guide: Storing Project Settings in Version Control