Predefined Build Parameters

TeamCity provides a number of build parameters which are ready to be used in the settings of a build configuration or in build scripts.

On this page:

The predefined build parameters can originate from several scopes:

- #Server Build Properties - the parameters generated by TeamCity on the server-side in the scope of a particular build. An example of such property is a build number.
- #Agent Properties - the parameters provided by an agent on connection to the server. The parameters are not specific to any build and characterize the agent environment (for example, the path to .Net framework). These are mainly used in agent requirements.
- #Agent Build Properties - the parameters provided on the agent side in the scope of a particular build right before the build start. For example, a path to a file with a list of changed files.

All these parameters are finally passed to the build.

There is also a special kind of server-side build parameters that can be used in references while defining other parameters, but which are not passed into the build. See Configuration Parameters below for the list of such properties.

The most up-to-date list of parameters can be obtained in the TeamCity web UI while defining a text value supporting parameters: either click on  icon to the right of the text field, or enter "%" in the text field.

Server Build Properties

System properties can be referenced using `%system.propertyName%`.

System Property Name	Environment Variable Name	Description
teamcity.version	TEAMCITY_VERSION	The version of TeamCity server. This property can be used to determine the build is run within TeamCity.
teamcity.projectName	TEAMCITY_PROJECT_NAME	The name of the project the current build belongs to.
teamcity.buildConfName	TEAMCITY_BUILDCONF_NAME	The name of the Build Configuration the current build belongs to.
teamcity.buildType.id	none	The unique id used by TeamCity to reference the Build Configuration the current build belongs to
teamcity.configuration.properties.file	none	Full name (including path) of the file containing all the build properties in alphabetical order.
build.is.personal	BUILD_IS_PERSONAL	Is set to `true` if the build is a personal one. Is not defined otherwise.
build.number	BUILD_NUMBER	The build number assigned to the build by TeamCity using the build number format. The property is assigned based on the build number format.

teamcity.build.id	none	The internal unique id used by TeamCity to reference builds.
teamcity.auth.userId	none	A generated username that can be used to download artifacts of other build configurations. Valid only during the build. Details
teamcity.auth.password	none	A generated password that can be used to download artifacts of other build configurations. Valid only during the build. Details
build.vcs.number.<VCS root ID>	BUILD_VCS_NUMBER_<VCS root ID>	The latest VCS revision included in the build for the root identified. See Configuring VCS Roots for the <VCS root ID> description. If there is only a single root in the configuration, the `build.vcs.number` property (without the VCS root ID) is also provided. ⚠ Please note that this value is a VCS-specific (for example, for SVN the value is a revision number while for CVS it is a timestamp)

Configuration Parameters

These are the parameters that other properties can reference (only if defined on the Parameters page), but that are not passed to the build themselves.

You can get the full set of such server properties by adding the `system.teamcity.debug.dump.parameters` property to a build configuration and examining the "Available server properties" section in the build log.

Among these properties are the following:

Dependencies Properties

These are properties provided by the builds the current build depends on (via a snapshot or an artifact dependency).

In the dependent build, dependencies properties have the following format:

```
dep.<btID>.<property name>
```

- `<btID>` — is the ID of the build configuration to get the property from. Only the configurations the current one has snapshot or artifact dependencies on are supported. Indirect dependencies configurations are also available (e.g. A depends on B and B depends on C - A will have C's properties available).
- `<property name>` — the name of the build parameter of the build configuration with the given ID.

⚠ When using build parameters of type "Password", referencing them from a dependency such as `%dep.<btID>.password_parameter%` will not retrieve the actual value. This is done for security reasons to prevent dependencies from accessing the value, thus restricting the possibility of unauthorized access to it.

Overriding Dependencies Properties

It is possible to redefine the build parameters in the snapshot-dependency builds when the current build starts. For example, build configuration A depends on B and B depends on C; when triggering A, there is the ability to change parameters in any of its dependencies using the following format:

```
reverse.dep.<btID>.<property name>
```

⚠ Note that if a parameter is redefined in B, but only A is triggered, no parameters change occurs.

It is also possible to change parameter in all dependencies at once using the syntax:

```
reverse.dep.*.<property name>
```

The `reverse.dep.` parameters are processed on queuing of the build where the parameters are defined. As the parameter's values should be known at that stage, they can only be defined either as build configuration parameters or in the custom build dialog. Setting the parameter during the build has no effect.

Pushing a new parameter into the build will supersede the "Do not run new build if there is a suitable one" snapshot dependency option and may trigger a new build if the parameter is set to a non-default value.

Note that the values of the `reverse.dep.` parameters are pushed to the dependency builds "as is", without reference resolution. %-references, if any, will be resolved in the context of the build where the parameters are pushed to.
<property name> is the name of the property to set in the noted build configuration. To set system property, <property name> should contain "system." prefix.

VCS Properties

These are the settings of VCS roots attached to the build configuration.

VCS properties have the following format:

```
vcsroot.<VCS root ID>.<VCS root property name>
```

- `<VCS root ID>` — is the VCS root ID as described on the Configuring VCS Roots page.
- `<VCS root property name>` — the name of the VCS root property. This is VCS-specific and depends on the VCS support. You can get the available list of properties as described above.

If there is only one VCS root in a build configuration, the `<VCS root ID>.` part can be omitted.

Properties marked by the VCS support as `secure` (for example, passwords) are not available as reference properties.

Branch-Related Parameters

When TeamCity starts a build in a build configuration where Branch specification is configured, it adds a branch label to each build. This logical branch name is also available as a configuration parameter:

```
teamcity.build.branch
```

To distinguish builds started on a default and a non-default branch, there is an additional boolean configuration parameter available since 7.1.5 which allows differentiating these cases:

```
teamcity.build.branch.is_default=true|false
```

For Git & Mercurial, TeamCity provides additional parameters with the names of VCS branches known at the moment of the build start. Note that these may differ from the logical branch name as per branch specification configured.
This VCS branch is available from a configuration parameter with the following name:

```
teamcity.build.vcs.branch.<VCS root ID>
```

Where <VCS root ID> is the VCS root ID as described on the Configuring VCS Roots page.

Other Parameters

Parameter Name	Description
teamcity.build.triggeredBy	a human-friendly description of how the build was triggered
teamcity.build.triggeredBy.username	if the build was triggered by a user, the username of this user is reported. When a build is triggered not by a user, this property is not reported.

Agent Properties

Agent-specific properties are defined on each build agent and vary depending on its environment. Aside from standard properties (for example, teamcity.agent.jvm.os.name or teamcity.agent.jvm.os.arch, etc. — these are provided by the JVM running on agent) agents also have properties based on installed applications. TeamCity automatically detects a number of applications including the presence of .NET Framework, Visual Studio and adds the corresponding system properties and environment variables. A complete list of predefined agent-specific properties is provided in the table below.

If additional applications/libraries are available in the environment, the administrator can manually define the property in the <agent home>/conf/buildAgent.properties file. These properties can be used for setting various build configuration options, for defining build configuration requirements (for example, existence or absence of some property) and inside build scripts. For more information on how to reference these properties, see the Defining and Using Build Parameters in Build Configuration page.

In the TeamCity Web UI, the actual properties defined on the agent can be reviewed by going to the Agents tab at the top navigation bar | <Agent> | <Agent> page | the Agent Parameters tab:

Predefined Property	Description
teamcity.agent.name	The name of the agent as specified in the buildAgent.properties agent configuration file. Can be used to set a requirement of build configuration to run (or not run) on particular build agent.
teamcity.agent.work.dir	The path of Agent Work Directory.
teamcity.agent.work.dir.freeSpaceMb	Free space available in the Agent Work Directory.
teamcity.agent.home.dir	The path of Agent Home Directory.
teamcity.agent.tools.dir	The path to the Tools directory on the Agent
teamcity.agent.jvm.os.version	The corresponding JVM property (see JDK help for properties description)
teamcity.agent.jvm.user.country	The corresponding JVM property (see JDK help for properties description)
teamcity.agent.jvm.user.home	The corresponding JVM property (see JDK help for properties description)
teamcity.agent.jvm.user.timezone	The corresponding JVM property (see JDK help for properties description)
teamcity.agent.jvm.user.name	The corresponding JVM property (see JDK help for properties description)
teamcity.agent.jvm.user.language	The corresponding JVM property (see JDK help for properties description)
teamcity.agent.jvm.user.variant	TThe corresponding JVM property (see JDK help for properties description)
teamcity.agent.jvm.file.encoding	The corresponding JVM property (see JDK help for properties description)
teamcity.agent.jvm.file.separator	The corresponding JVM property (see JDK help for properties description)

teamcity.agent.jvm.path.separator	The corresponding JVM property (see JDK help for properties description)
teamcity.agent.jvm.specification	The corresponding JVM property (see JDK help for properties description)
teamcity.agent.jvm.version	The corresponding JVM property (see JDK help for properties description)
teamcity.agent.jvm.java.home	See the section below for details
teamcity.agent.os.arch.bits	Since TeamCity 10.0 The agent's OS bitness
DotNetFramework<version>[_x86|_x64]	This property is defined if the corresponding version(s) of .NET Framework runtime is installed. (Supported versions are 1.1, 2.0, 3.5, 4.0 - 4.6.1)
DotNetFramework<version>[_x86|_x64]_Path	This property value is set to the corresponding framework runtime version(s) path(s)
DotNetFrameworkSDK<version>[_x86|_x64]	This property is defined if the corresponding version(s) of .NET Framework SDK is installed. (Supported versions are 1.1, 2.0)
DotNetFrameworkSDK<version>[_x86|_x64]_Path	This property value is the path of the corresponding framework SDK version.
DotNetFrameworkTargetingPack<version>_Path	This property value is the path to the corresponding Reference assemblies (AKA Targeting Pack) location. (Supported versions are 2.0 - 4.6.1)
WindowsSDK<version>	This property is defined if the corresponding version of Windows SDK is installed. (Supported versions are 6.0, 6.0A, 7.0, 7.0A, 7.1 , 8.0, 8.0A, 8.1, 8.1A, 10)
WindowsSDK<version>_Path	This property value is the path of the corresponding version of Windows SDK.
VS[2003|2005|2008|2010|2012|2013|2015|2017]	This property is defined if the corresponding version(s) of Visual Studio is installed
VS[2003|2005|2008|2010|2012|2013|2015|2017]_Path	This property value is the path to the directory that contains deve `nv.exe`
teamcity.dotnet.nunitlauncher<version>	This property value is the path to the directory that contains the standalone NUnit test launcher, `NUnitLauncher.exe`. The version number refers to the version of .NET Framework under which the test will run. The version equals the version of .NET Framework and can have a value of 1.1, 2.0, or 2.0vsts.
teamcity.dotnet.nunitlauncher.msbuild.task	The property value is the path to the directory that contains the MSBuild task dll providing the NUnit task for MSBuild, Visual Studio (sln).
teamcity.dotnet.msbuild.extensions2.0	The property value is the path to the directory that contains MSBuild 2.0 listener and tasks assemblies.
teamcity.dotnet.msbuild.extensions4.0	The property value is the path to the directory that contains MSBuild 4.0 listener and tasks assemblies.
teamcity.agent.ownPort	The agent port used by the TeamCity server to connect to the agent
teamcity.agent.protocol	The protocol used for data transfers between the agent and the server
teamcity.agent.cpuBenchmark	CPU benchmarking result for the agent
teamcity.agent.hardware.cpuCount	The number of processors in the build agent system
teamcity.agent.hostname	The name of the build agent host

- Make sure to replace "." with "_" when using properties in MSBuild scripts; e.g. use `teamcity_dotnet_nunitla`

uncher_msbuild_task instead of teamcity.dotnet.nunitlauncher.msbuild.task
- _x86 and _x64 property suffixes are used to designate the specific version of the framework.
- teamcity.dotnet.nunitlauncher properties cannot be hidden or disabled.

Agent Environment Variables

An agent can define some environment variables. These variables can be used in build scripts as usual environment variables.

Java Home Directories

When a build agent starts, first the installed JDK and JRE are detected; when they are found, the Java-related environment variables are defined as described in the section below.

⚠ The environment variables are defined only if they are not already present in the environment: if a started agent already has the Java-related environment variables set, they are not redefined.

Detecting Java on Agent

The installed Java is searched for in the ALL locations listed below. Then, every discovered Java is launched to verify that it is a valid Java installation, and the Java version and bitness are determined based on the output.

The following locations are searched (a number of locations is common for all operating systems; some of them are OS-specific):

For All OS

- If defined, a custom directory on the agent is searched for Java installations. Defining a custom directory to search for Java is described below.
- The agent tools directory, <Agent Home Directory>/tools, is checked for containing a jre or jdk. By default, the subdirectories of /tools are not scanned. To search the subdirectories, define teamcity.agent.java.search.path=%agent.tools.NAME%/INNER_PATH in the buildAgent.properties file.
  ⚠ For Unix and Mac OS, remember to set the executable bit on the files for TeamCity to be able to launch the discovered Java.
- It is checked whether the JAVA_HOME, JDK_HOME, JRE_HOME variables are defined
- The OS-specific locations, listen in the next section, are checked
- The PATH environment variables are searched and the discovered directories are checked for containing Java

OS-specific locations

Windows

- The Windows Registry is searched for the Java installed with the Java installer
- C:\Program Files and C:\Program Files (x86) directories are searched for Java and JavaSoft subdirectories
- the C:\Java directory is searched

Unix

The following directories are searched for Java subdirectories:

- /usr/local/java
- /usr/local
- /usr/java
- /usr/lib/jvm
- /usr

Mac OS
The following directories are searched:

- /System/Library/Frameworks/JavaVM.framework/Versions/<Java Version>/Home
- /Library/Java/JavaVirtualMachines/Versions/<Java Version>/Home
- /Library/Java/JavaVirtualMachines/<Java Version>/Contents/Home

Defining Custom directory to Search for Java

You can define a custom directory on an agent to search for Java installations in by adding the teamcity.agent.java.search.path property to the buildAgent.properties file.

You can define a list of directories separated by an OS-dependent character.

Defining Java-related Environment Variables

For each major version V of java, the following variables can be defined:

- `JDK_1V`
- `JDK_1V_x64`
- `JRE_1V`
- `JRE_1V_x64`

The JDK variables are defined when the JDK is found, the JRE variables are defined when the JRE is found but the JDK is not. The _x64 variables point to 64-bit java only; the variables without the _x64 suffix may point to both 32-bit or 64-bit installations but 32-bit ones are preferred.
If several installations with the same major version and the same bitness but different minor version/update are found, the latest one is selected.

In addition, the following variables are defined:

- `JAVA_HOME` - for the latest JDK installation (but 32-bit one is preferred)
- `JDK_HOME` - the same as `JAVA_HOME`
- `JRE_HOME` - for the latest JRE or JDK installation (but 32-bit one is preferred), defined even if JDK is found.

The `JRE_HOME` and `JDK_HOME` variables may points to different installations; for example, if JRE 1.7 and JDK 1.6 but no JDK 1.7 installed - `JRE_HOME` will point to JRE 1.7 and `JDK_HOME` will point to JDK 1.6.

All variables point to the java home directories, not to binary files. For example, if you want to execute javac version 1.6, you can use the following path:

In a TeamCity build configuration:

```
%env.JDK_16%/bin/javac
```

In a Windows bat/cmd file:

```
%JDK_16%\bin\javac
```

In a unix shell script:

```
$JDK_16/bin/javac
```

Agent Build Properties

These properties are unique for each build: they are calculated on the agent right before build start and are then passed to the build.

System Property Name	Environment Variable Name	Description
teamcity.build.checkoutDir	none	Checkout directory used for the build.
teamcity.build.workingDir	none	Working directory where the build is started. This is a path where TeamCity build runner is supposed to start a process. This is a runner-specific property, thus it has different value for each new step.
teamcity.build.tempDir	none	Full path of the build temp directory automatically generated by TeamCity. The directory will be cleaned after the build.

teamcity.build.properties.file	TEAMCITY_BUILD_PROPERTIES_FILE	Full name (including path) of the file containing all the `system.*` properties passed to the build. "system." prefix stripped off. The file uses Java properties file format (for example, special symbols are backslash-escaped).
teamcity.build.changedFiles.file	none	Full path to a file with information about changed files included in the build. This property is useful if you want to support running of new and modified tests in your tests runner. This file is only available if there were changes in the build; it is not available for history builds.

See also system properties related to Risk Tests Reordering in Custom Test Runner