TeamCity FAQ

THIS PAGE IS OBSOLETE. USE Documentation space INSTEAD

General information

When are you going to release next version?

We plan to release Agra (TeamCity 2.0) on March, 2007.

Are there real-life examples of TeamCity usage?

We do eat our dog food. Currently in JetBrains we have a production TeamCity installation which runs IntelliJ IDEA builds, Resharper builds, DotTrace builds, TeamCity 's builds (and some other ones). We utilize more than 20 build agent computers to run all those builds.

Where can I take a look?

The sample TeamCity installation is available at http://teamcity.jetbrains.com/?guest=1. You can see how the builds for several open source projects are processed by TeamCity.

How can I report a bug or request a feature?

Before filing a bug, please read How to Debug problems

See TeamCity Jira
or use our newsgroup (TeamCity forum)

Terms & Concepts

What is the 'project'?

The "project" represents a set of VCS configurations (VCS roots) that contain the source files. A project can contain several build configurations (see #What is the 'build configuration'?).

What is the 'build configuration'?

The build configuration is a set of actions and parameters specified for the build. These include build runner parameters, build schedule (specified time or according to changes in VCS), and more.

Examples of build configurations are integration builds, release builds, nightly builds, etc.

What is the 'build agent'?

Build agent is a part of TeamCity that actually runs the build. Usually, a build agent corresponds to a separate computer. When needed, TeamCity chooses a free agent and runs a build on it (see #How are the builds distributed among agents?). A build agent can run only one build at a time.

However, if you need to install TeamCity for testing or demonstration purposes, you can install the server and build agent on the same machine.

Installation and Upgrade

General configuration issues

How to configure build agents?

The build agent configuration file has the same format as Java properties file (name-value pairs). The file is located at:

```
<installation path>\conf\buildAgent.properties
```

There are three types of agent properties:

- General (like name, agent port, and the server URL).
- System properties - if the process that is executed by the build runner reads properties from system variables (for example, in case of Java process runner, java.lang.System.getProperty(<name>)), you need to use the following format: `system.<name>`
- Environment variables - if the process that is executed by the build runner reads properties from the OS environmnet, you need to use the following format: `env.<varname>`

For example, system and environment properties are used to:

- pass additional information to the build (`env.CATALINA_HOME=/home/builduser/tomcat`)
- match build configuraion requirements (see #How are the builds distributed among agents?)

How to create and configure TeamCity projects?

The projects are configured via TeamCity's web interface. After you have logged in, perform the following steps:

1. Click Administration link in the navigation bar.
2. On the Administration page, in the Projects and Build Configurations area, click Create new project link.
3. On the General project settings page that opens, type the project name and description, then click Create.

When the project is created, you can configure its VCS roots (see How to configure your VCS below) and create build configurations (see #How to set up build configurations?).

How to configure your VCS?

To configure VCS from which TeamCity obtains the source files for your project, go to Administration page and edit your project. You will see the list of supported VCS systems. By default these are CVS, Perforce, and Subversion (others can be added by plugins).

NOTE: By default the server will check for updates every 5 seconds. In order to change this interval, Tomcat should be started with the `modification.check.interval` property defined like this: `-Dmodification.check.interval=<interval in seconds>`.

For example:

```
SET JAVA_OPTS=%JAVA_OPTS% -Dmodification.check.interval=3600
```

The line above configures check interval to be one hour.

For more info on VCS settings, see #How to set up build configurations?.

How to set up build configurations?

Each project can contain one or more build configurations (integration build, release build, performance tests build etc). To add a new build configuration, go to Administration page, and click Create build configuration link for the particular project.

On the Build Configuration page, you can enter the build configuration name and specify the Run parameters, i.e. parameters that are specific for your build runner (Ant, Maven, MSBuild, NAnt, Inspections, Ipr). For instance, they specify where the `build.xml`file is located (in VCS or on the server), the file content, which target to run, where to download build artifacts, etc. When you specify the build runner in the Runner list, the set of parameters displayed on the build configuration page is changed according to the selected runner.

There are three important options that you need to configure for each build:

- Checkout sources automatically. If this option is switched on, the server will automatically take necessary source files or build patches from VCS and pass it to the build agents. If the option is switched off, it is supposed that your build script is able to check out these source files.
- Clean all files before build. This option is available only if the Checkout sources automatically option is switched on. If

the option is switched on, before starting the build all the source files for the certain project on the agent will be cleared. Actually, it is the same as doing the full project check out before starting new build.
- Run one build at a time. If this option is switched on, only one build of this configuration can be started at a time among all the agents. If one more build is started while the other one is in progress, the newly started build will get into the queue.

When you save the newly created build configuration or start editing some existing one, you can see the additional Build configuration steps in the right-upper corner of the configuration page. These steps allows you to configure the following:

- Build's properties and environment variables.
- Agent requirements that are requirements that a build agent should meet in order to be able to run this build configuration (see #How to configure build agents? and #How are the builds distributed among agents. They are also divided into system properties (`system.<varname>`) and environment variables (`env.<varname>`).
- Build triggering where you can customize due to which events the build should be started and specify build scheduling.

How are the builds distributed among agents?

Each build has a set of requirements that the agent should meet to run the build (see #How to set up build configurations?). At the same time, each agent has a set of parameters showing which requirements it can meet (see #How to configure build agents?)

When the build server chooses where to run the build, it looks for an agent which parameters match the build requirements.

For example, you want to specify that your build requires Websphere. In this case, you should go to the Administration page, select to edit the necessary build configuration, and click the Agent Requirements link. There you can add the following environment variable: `env.WEBSPHERE_HOME` and set "exists" condition. TeamCity will load the change without need to restart.

After that you can configure build agents to match this requirement by adding the following line into `<installation folder>/conf/buildAgent.properties`:

```
env.WEBSPHERE_HOME=/path/to/WebSphere
```

Build agent should be restarted after modifications.

You can view the list of compatible agents for each build configuration via TeamCity web interface. Click the Projectstab, and then click the required build configuration. On the right side bar, you will see the Settings bar, where the information about the number of compatible agents is displayed. Click the link in this section to open the Compatible Agents page. On this page there is a list of all the registered agents where they are marked as compatible or incompatible. For the incompatible agents, the information about conflicting variables is displayed.

How to use and configure notification service?

TeamCity sends notifications about successful and failed builds. The notification is sent as soon as the first problem occurs (even before the build process is completed). Now there are e-mail, Jabber, Windows tray, and IntelliJ IDEA plugin notifiers.

To configure your notification service, use TeamCity interface. Open the My Settings tab, and in the Notifications area, specify e-mail or Jabber account. In the right-upper corner of the My Settings page you can also see IntelliJ IDEA plugin and Windows tray notifier.

Note: TeamCity open API allows adding custom notifiers.

By default, if the failing (or failed) build contains your changes, you are notified about that (using notifiers specified in your settings). But you can customize notifcation service by creating your own notification rules. For example, you can:

- get notifications even if the build does not contain your changes
- get notifications on more events (e.g. build started or build status changed)
- define the list of notifiers for each rule

To create your own notification rules, on the My Settings page, in the Notifications area, click the Edit notification rules link.

Notification messages are created based on TeamCity notification templates that can be configured according to your needs. For more information, see Notificator templates customization.

How can I get build artifacts from my build?

Artifacts are files and folders that are produced by the build and needed to be uploaded to the server and be available for download (e.g. installer, WAR file, log files, etc).

You can mark certain files or folders as artifacts using TeamCity Web UI. Go to the Administration tab, and edit necessary build

configuration. On the Build configuration page, in the Artifact paths field, specify artifcats as a comma-separated string. For example: `dist/mydata.jar,installer.jar`

These comma-separated values represent paths relative to build working directory. After the build execution, TeamCity will search for these artifacts in the build working directory and upload them to the server where they will be available for download through Web UI.

What about build number support?

You can configure build numbering using TeamCity Web UI. Go to the Administration tab, and edit necessary build configuration. On the Build configuration page, edit Build number format and Build counter fields.

You can use different patterns for your build number, for example "5.0.{0}". The "{0}" substring will be replaced with the actual counter value.

For acquiring the build number at run time, during the build script execution, you can use the following predefined build properties:
${successful.build.number} - is increased for successful builds
${build.number} - is increased always
These numbers are also available as environment variables:
SUCCESSFUL_BUILD_NUMBER, BUILD_NUMBER

The information about build numbers is stored in
`.buildNumbers.properties`
file in $HOME/.BuildServer/config/<PROJECT_NAME>/ directory.

Can I access Perforce/SVN changeset number in my build?

Each project in TeamCity can have several associated VCS roots. So in fact, there are
several changeset numbers associated with each build.

For acquiring the changeset numbers at run time, you can use the following predefined build properties:
${build.vcs.number.1}, ${build.vcs.number.2} etc.

How can I control/detect build working directory?

See Build Process Working Directory page.

How to configure the location where TeamCity stores its data?

Use "teamcity.data.path" system property which must be set for Tomcat's VM.
By default it points to "${user.home}/.BuildServer" directory.

How to configure automatic cleanup for the builds history?

To avoid disk flooding, TeamCity should be configured to periodically remove old history entries and artifacts. Several cleanup policies are available:

1. Keep history entries for N days
2. Keep history entries for N successful builds
3. Keep history entries for N days but minimum M builds

When removing a history entry, all corresponding data is also removed: log messages, failed tests data, build artefacts (if any).
A cleanup policy can be specified either for all build configurations or per build configuration.

To specify the required cleanup policy, go to the Administration tab, and in the Build History Clean up area (upper-right corner of the Administration page), click the Configure Clean up policy link.

How to configure code coverage?

You can add code coverage statistics to build execution results. The code coverage engine used by TeamCity is EMMA.

This functionality is now available only for Ant build runner.

To switch on code coverage for a certain build configuration, go to the Administration tab, and edit necessary build configuration. On the Build configuration page, select the Enable code coverage check box.

If you enabled code coverage for a build configuration, the results will be shown on the results page of this build (you can view it by click the link of the correponding build on the Projects page).

In your build script, you can detect that code coverage is enabled when system property report.html.out.file is set.

Linking to last finished and last successful builds

It is possible to create links to last finished or last successful build of some build configuration. To do so you can copy link from browser address bar pointing to build results or build artifacts (or other build related page) and change parameter 'buildId'. To make a link to last finished build use 'lastFinished' value of buildId parameter; to link to last successful build use 'lastSuccessful' value of the parameter. For example:

http://<some host>/viewLog.html?tab=buildResultsDiv&buildTypeId=CodeHaus_XStream&buildId=lastFinished - points to the last finished build results page.
http://<some host>/viewLog.html?tab=artifacts&buildTypeId=CodeHaus_XStream&buildId=lastSuccessful - points to artifacts of the last successful build.

Linking to a build by build number

You can create link to a certain build by build number. For example:
http://<some host>/viewLog.html?tab=buildResultsDiv&buildTypeId=JMock_Tests&buildNumber=131 - points to the build results page of a build with build number 131.

Build runners configuration

What runners are supported?

The following build runners are supported:

- Ant
- Maven
- NAnt
- MS Build
- Visual Studio Solutions (2003 and 2005)
- Ipr - IntelliJ IDEA runner
- Inspections

By default, build agents are installed with all the runners that are supported by TeamCity. So, if you see that the agent is incompatible for the .Net build becuse of "incompatible runner", it means that there are some problems during the runner launching. You can find information about these problems in the agent's log file (`<installation path>\bin\team-server.log`)

To specify the build runner for a certain build configuration, go to the Administration tab, and edit necessary build configuration. On the Build configuration page, in the Runnerlist, select one of the supported runners. The set of parameters displayed on the page may change depending on the selected runner. To learn more about runner configurations, refer to the sections below.

How to add custom JVM parameters to build agent runner?

For JAVA-based runners it is possible to specify Java Virtual Machine parameters, such as maximum heap size or parameters enabling remote debugging. These settings will be passed by the JVM used to run your build.

This can be done using TeamCity Web UI. Go to the Administration tab, and edit necessary build configuration. On the Build configuration page, edit the JVM parameters field (it is available only if some Java-based runner is selected).

For example, you can specify the following values in this field:

```
-Xmx512m -Xms256m
```

How to configure build target JDK?

For JAVA-based runners, you can configure JDK in build configuration parameters. Go to the Administration tab, and edit necessary build configuration. On the Build configuration page, edit the JDK home path field (it is available only if some Java-based runner is selected).

If this field is not empty then path is read from JAVA_HOME environment variable or JDK home specified on the build agent machine. If these values are also not specified, the TeamCity will use JDK home on which build agent is started itself.

For all run parameters with values specified in terms of other properties (like `%system.JAVA_1_4%`), an implicit requirement is added ensuring the referenced property is either defined on the agent machine or specified in build parameters.

How to configure Maven 2 build?

No additional configuration on the agent side is needed to run Maven 2 builds. A snapshot version of Maven 2.1 is distributed as an agent plugin (just like bundled Ant 1.6.5)

Note that Maven 2 build runner has two required parameters:

- goals - a set of maven goals to be executed
- pomLocation- the directory where the pom.xml file is located. In the example the "%build.working.dir%" macro is used, that is substituted with the real directory path on the concrete agent machine.

Alternatevely, if there is a separate Maven 2.1.x installation on the agent machine, and either the environment variable "M2_HOME" or java system property "maven.home" is defined, the agent will try to start Maven from that location rather than using the bundled one. Note that because of more decent MavenEmbedder implementation, only Maven of version 2.1 can be run by the agent.

What are requirements for .Net runners?

All the .Net build agents must have .Net framework 1.1 or .Net framework 2.0 (or both) installed. The build agent also must have the corresponding properties - `system.DotNetFramework1.1` or `system.DotNetFramework2.0` - specified to indicate that the corresponding framework is installed.

How to configure NAnt build agent?

On the build agents that should run NAnt builds you'll need to set up the environment variable NAntHome. You can specify it in the agent's properties file (`buildAgent\conf\buildAgent.properties`) by adding env.NAntHome property. Note that property name is case-sensitive. This environment variable should provide path to NAnt distribution folder. Please, use NAnt build which is newer than 0.85 RC4.

In buildAgent.properties, you should also specify one or both system properties for .NET framework: system.DotNetFramework1.1 and/or system.DotNetFramework2.0

How to configure Microsoft Visual Studio .NET 2003 solution files?

Microsoft Visual Studio .NET 2003 solution build runner is build on the base of NAnt runner, described before. Solution task is used to run the build. Projects files are expected to be wellformed XML. You may configure this runner to run NUnit tests.

How to configure MSBuild build agent?

On the build agents that will run MSBuild builds, please set up Microsoft .NET Framework 2.0. For some builds you may need to have Microsoft .NET Framework SDK 2.0 installed.

How to build Microsoft Visual Studio 2005 solution files?

Microsoft Visual Studio 2005 solution runner is built on MSBuild runner, which is discussed upper. Differences are in TeamCity Administration Web UI only. You may configure this runner to run NUnit tests.

How to create a build configuration with Ipr build runner?

TeamCity allows to compile and run tests for IntelliJ IDEA projects using Ipr build runner.

When you specify the build configuration for your project, you can define the parameters of the Ipr build runner on Create/Edit Build Configuration page:

1. From the Runner drop-down list, select Ipr.
2. In the Path to .ipr file field, type the path to IntelliJ IDEA project file.
   Basing on the information on the IntelliJ IDEA project file you provide in this field, Ipr build runner will create an Ant script to run the build. Compared with Ant, the Ipr build runner creates the build script on-the-fly and does not require the build script rebuild.
3. To specify the Workspacesetting, click Add JDK, Add Library, or Add IDEA JDK links.
   In the window which opens, type the name of the JDK library you use and click Add.
   The corresponding content appears in the Workspace field (you can modify it if necessary).
   Note:
   The JDK and libraries are not kept in VCS, and it is necessary to define them on this step.
   Each project depends of one or several JDKs. For each JDK you have to define a set of jar files which is the library and JDK path used to compile the sources.
   Ipr runner generates Ant script, and the most transparent way to define the file set is to specify an Ant script fragment containing the corresponding path element. JDK home path should be defined using system variable with the corresponding name. If there is a JDK named, for example, `"IDEA-jdk"` there should be an entry like this:

```
<path id="jdk.IDEA-jdk.classpath">
  <fileset dir="%system.jdk.IDEA.home%/jre/lib">
 <include name="*.jar"/>
  </fileset>
</path>
```

`%system.jdk.IDEA.home%` will be replaced with the value from the corresponding build agent. A variable with this name will also be used to run the javac.
If some global libraries are used in the project, you must define a reference named `global.library.#LIBRARY-NAME#.path`.

4. In the Test patterns field, click Enter test patters to define a set of values for a JUnit task. You can select the classes of the specific modules which will be run as JUnit tests.
   By default, JUnit looks for all classes ending with `"Test"` in each project module but you can define it explicitly by typing `"*:**/*Test.java"`
   where
   * is the template of the module name
   `**/*Test.java` is the template of the test class name.
   You can also exclude some classes from testing. To do so, put a minus sign before the template. The sample below includes all test files from modules ending with `"-est"` and excludes all files from package containing `"exclude"` subpac kage:
   `-test:*/*Test.java`
   `-*:**.exclude.**/*.java`
5. To define the runtime classpath of some specific module, select Override classpath in tests option. By default, the test runtime classpath is combined from runtime classpaths of all used project modules.
6. The values you specify in JUnit fork mode (select Include Ant runtime option when JUnit fork mode is enabled), JVM parameters,Test working dir, and Timeoutfields become the corresponding attributes of the generated JUnit task.
7. If your IDEA project has predefined path variables, you should setup their values by specifying corresponding build properties. These properties should be named like path.macro.CATALINA_HOME, where CATALINA_HOME - name of your path variable. (path variables substitution works starting from TeamCity 1.2 release)

How to create a build configuration with inspections runner

Inspections project should define a number of additional parameters. They are:

1. *.ipr file - the project which would be analyzed (mandatory)
2. profile file - the profile which would be used. This parameter can be ommited. In such a case Code Inspection would proceed with "Project Editor Settings"
3. jdk.* - should be set to specify jdks used in project. This parameter can be ommited which means that JAVA_HOME System parameter would be used.
4. library.* - should be set to specify global libraries used in project (if any)
5. jvmArgs - arguments for Code Inspection jvm. Otherwise default jvm args would be used.

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE project SYSTEM "../project-config.dtd">
<project>
  <vcs-roots/>
  <build-type id="Inspection_inspection" name="Test" run-type="Inspection"
oneRunningInstanceOnly="false" checkout-on-server="false" clean-build="false">
    ...
    <run-parameters>
      <param name="ipr" value="c:/Test.ipr" />
      <param name="jvmArgs" value="-Xms32m -Xmx128m -XX:MaxPermSize=92m" />
    </run-parameters>
    ...     </build-type>
</project>
```

Advanced topics

How to configure MySQL database?

TeamCity supports MySQL version 4.1.18 and later or MySQL 5.0.X and later. InnoDB is required and should be configured as default engine. To configure TeamCity to use MySQL database:

1. Download mysql jdbc driver from http://dev.mysql.com/downloads/connector/j/5.0.html and put jar file to webapps/ROOT/WEB-INF/lib directory of TeamCity.
2. Shutdown TeamCity server.
3. Modify database configuration in general Spring configuration file. It is located in the `/WEB-INF/buildServerSpring.xml` file within the exploded TeamCity application. Just specify appropriate settings for mysqlDbSettings bean, and set reference to mysqlDbSettings in the buildServer bean definition.
4. Create database with the name specified in the connectionUrl property of mysqlDbSettings bean. You should choose one-byte charset for the database (latin1 and the like), multi-byte charsets (like UTF) are not supported by TeamCity.
5. Grant necessary access rigths for the user specified in the mysqlDbSettings bean.
6. Start TeamCity server.

How to configure NT domain authentication

NT domain authentication is supported if TeamCity is installed on a Windows platform.
To enable NT domain authentication, go to the Administration interface, "Server Configuration" link.
For "Authentication type", select "Use MS Windows NT domain authentication" and save settings.

How to enable guest login

TeamServer supports guest user login (login without password). To enable it go to the Administration interface, "Server Configuration" link.
Mark "Allow to login as a guest user" checkbox and save.

If guest login is enabled, you will see guest login link on the login page.

Guest user is not able to start or stop builds, or change build configurations in any way. My changes and profile settings pages are not available for guest user too.

How to disable user account registration from the login page

To disable user account registration from the login page go to the Administration interface, "Server Configuration" link.
If you use TeamCity authentication scheme, there is "Allow user registration from the login page" flag.

How to mark build stages with custom messages?

For Ant scripts there is a possibility to use custom ant tasks which allow user to mark certain build stages with custom text. This text will be visible in the web page describing the build being excuted as well as in the build log. By default the TeamServer will use the names of ant tasks used in the build file to describe build process.
There are 3 tasks located in the buildServerAntRunntime.jar from the AntPlugin distribution:

- started
- finished
- progress

If the script using these tasks is run only by TeamServer's agents, there is no need to explicitly define them in the script using taskdef directives: all the registration will be done by the agent.
In order to be able to run the script from the command line, there must be corresponding taskdefs (see ant manual about how to define the task). The following class names must be used when defining the tasks:

- jetbrains.buildServer.agent.ant.tasks.Started
- jetbrains.buildServer.agent.ant.tasks.Finished
- jetbrains.buildServer.agent.ant.tasks.Progress

Each task has the only attribute - message specifying the text to be displayed. The tasks started and finished work as opening and closing braces respectively, allowing you to group build activities in blocks. For example:

```
<started message="Building distribution...">
  <progress message="Searching for files">
  ...
  <progress message="Building archives">
  ...
  <started message="Deployment">
    ...
    <progress message="Copying files to $ {deployment.dir}">
    ...
    <progress message="Clearing temporary files...">
    ....
  <finished/>
```

```
  ...
  <progress message="Distribution jars were successfuly built.">
<finished/>
```

The same functionality can be achieved by using standard echo Ant task. In this case, you need to specify echo messages using the following formats:

- "##[message" - started
- "##]message" - finished
- "##message" - progress