Artifact Dependencies

This page details configuration of the TeamCity Artifact Dependencies.

The Build Configuration Settings | Dependencies page, Artifact Dependencies section allows configuring the dependencies. Since TeamCity 10.0, it is possible to disable a configured dependency temporarily or permanently using the corresponding option in the last column of the Artifact Dependencies list.

Configuring Artifact Dependencies Using Web UI

To add an artifact dependency to a build configuration:

1. When creating/editing a build configuration, open the Dependencies page.
2. Click the Add new artifact dependency link and specify the following settings:

Option	Description
Depend on	Specify the build configuration for the current build configuration to depend on. A dependency can be configured on a previous build of the same build configuration
Get artifacts from	Specify the type of build whose artifacts are to be taken: last successful build, last pinned build, last finished build, build from the same chain (this option is useful when you have a snapshot dependency and want to obtain artifacts from a build with the same sources), build with a specific build number or the last finished build with a specified tag. ⚠ • When selecting the build configuration, take your clean-up policy settings into account. Builds are cleaned and deleted on a regular basis, thus the build configuration could become dependent on a non-existent build. When artifacts are taken from a build with a specific number, then the specific build will not be deleted during clean-up. • If both dependency by sources and dependency by artifacts on the last finished build are configured for a build configuration, then artifacts will be taken from the build with the same sources.
Build number	This field appears if you have selected build with specific build number in the Get artifacts from list. Specify here the exact build number of the artifact.
Build tag	This field appears if you have selected last finished build with specified tag in the Get artifacts from list. Specify here the tag of the build whose artifacts are to be used. When resolving the dependency, TeamCity will look for the last successful build with the given tag and use its artifacts.
Build branch	This field appears if the dependency has a branch specified in the VCS Root settings. Allows setting a branch to limit source builds only to those with the branch. If not specified, the default branch is used. The logic branch name (shown in the UI for the build) is to be used. Patterns are not supported.
Artifacts Rules	A newline-delimited set of rules. Each rule must have the following syntax: `[+:|-:]SourcePath[!ArchivePath][=>DestinationPath]` Each rule specifies the files to be downloaded from the "source" build. The SourcePath should be relative to the artifacts directory of the "source" build. The path can either identify a specific file, directory, or use wildcards to match multiple files. Ant-like wildcards are supported. Downloaded artifacts will keep the "source" directory structure starting with the first * or ?. DestinationPath specifies the destination directory on the agent where downloaded artifacts are to be placed. If the path is relative (which is recommended), it will be resolved against the build checkout directory. If needed, the destination directories can be cleaned before downloading artifacts. If the destination path is empty, artifacts will be downloaded directly to the checkout root.

Basic examples:

- Use *a/b/**=>lib* to download all files from *a/b* directory of the source build to the *lib* directory. If there is a *a/b/c/file.txt* file in the source build artifacts, it will be downloaded into the file *lib/c/file.txt*.
- At the same time, artifact dependency ***/*.txt=>lib* will preserve the directories structure: the *a/b/c/file.txt* file from source build artifacts will be downloaded to *lib/a/b/c/file.txt*.

ArchivePath is used to extract downloaded compressed artifacts. Zip, 7-zip, jar, tar and tar.gz are supported. ArchivePath follows general rules for SourcePath: ant-like wildcards are allowed, the files matched inside the archive will be placed in the directory corresponding to the first wildcard match (relative to destination path)
For example: *release.zip!*.dll* command will extract all .dll files residing in the root of the *release.zip* artifact.

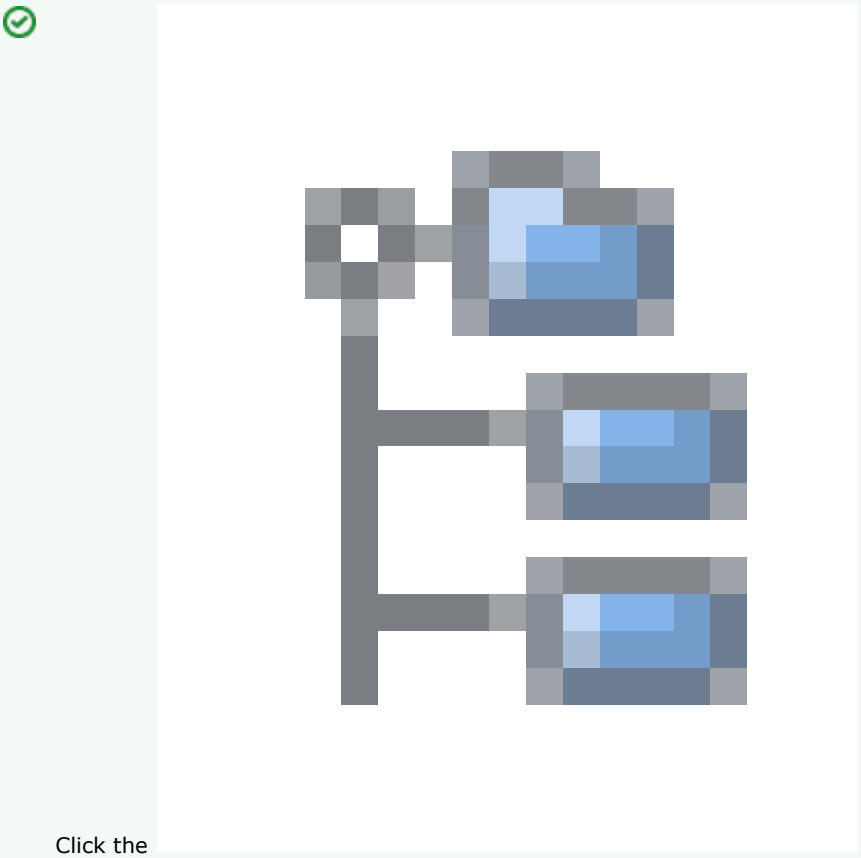
Archive processing examples:

- *release-*.zip!*.dll=>dlls* will extract *.dll from all archives matching the *release-*.zip* pattern to the *dlls* directory.
- *a.zip!**=>destination* will unpack the entire archive saving the path information.
- *a.zip!a/b/c/**/*.dll=>dlls* will extract all .dll files from *a/b/c* and its subdirectories into the *dlls* directory, without the *a/b/c* prefix.

+: and **-:** can be used to include or exclude specific files from download or unpacking. As **+:** prefix can be omitted: rules are inclusive by default, and at least one inclusive rule is required. The order of rules is unimportant. For each artifact the most specific rule (the one with the longest prefix before the first wildcard symbol) is applied. When excluding a file, DestinationPath is ignored: the file won't be downloaded at all. Files can also be excluded from archive unpacking. The set of rules applied to the archive content is determined by the set of rules matched by the archive itself.

Exclusive patterns examples:

- ***/*.txt=>texts*
  -:bad/exclude.txt
  Will download all *.txt files from all directories, excluding `exclude.txt` from the `bad` directory
- *+:release-*.zip!**/*.dll=>dlls*
  -:release-0.0.1.zip!Bad.dll
  Will download and unpack all dlls from `release-*.zip` files to the `dlls` directory. The `Bad.dll` file from `release-0.0.1.zip` will be skipped
- ***/*.*=>target*
  *-:excl/**/*.**
  +:excl/must_have.txt=>target
  Will download all artifacts to the `target` directory. Will not download anything from the `excl` directory, but the file called `must_have.txt`

	 Click the icon to invoke the Artifact Browser. TeamCity will try to locate artifacts according to the specified settings and show them in a tree. Select the required artifacts from the tree and TeamCity will place the paths to them into the input field. The artifacts placed under the .teamcity directory are considered hidden . These artifacts are ignored by wildcards by default. If you want to include files from the .teamcity directory for any purpose, be sure to add the artifact path starting with `.teamcity` explicitly. Example of accessing hidden artifacts: • `.teamcity/properties/*.properties` • `.teamcity/*.*`
Clean destination paths before downloading artifacts	Check this option to delete the content of the destination directories before copying artifacts. It will be applied to all inclusive rules

At any point you can launch a build with custom artifact dependencies.

Configuring Artifact Dependencies Using Ant Build Script

This section describes how to download TeamCity build artifacts inside the build script. These instructions can also be used to download artifacts from outside of TeamCity.

To handle artifact dependencies between builds, this solution is more complicated then configuring dependencies in the TeamCity UI but allows for greater flexibility. For example, managing dependencies this way will allow you to start a personal build and verify that your build is still compatible with dependencies.

To configure dependencies via Ant build script:
1. Download Ivy.

TeamCity itself acts as an Ivy repository. You can read more about the Ivy dependency manager here: http://ant.apache.org/ivy/.

2. Add Ivy to the classpath of your build.
3. Create the `ivyconf.xml` file that contains some meta information about TeamCity repository. This file is to have the following content:

```
<ivysettings>
<property name='ivy.checksums' value=''/>
<caches defaultCache="${teamcity.build.tempDir}/.ivy/cache"/>
<statuses>
  <status name='integration' integration='true'/>
</statuses>
<resolvers>
 <url name='teamcity-rep' alwaysCheckExactRevision='yes' checkmodified='true'>
   <ivy
pattern='http://YOUR_TEAMCITY_HOST_NAME/httpAuth/repository/download/[module]/[revision]/teamcity-ivy.xml' />
   <artifact
pattern='http://YOUR_TEAMCITY_HOST_NAME/httpAuth/repository/download/[module]/[revision]/[artifact](.[ext])' />
 </url>
</resolvers>
<modules>
 <module organisation='.*' name='.*' matcher='regexp' resolver='teamcity-rep' />
</modules>
</ivysettings>
```

4. Replace `YOUR_TEAMCITY_HOST_NAME` with the host name of your TeamCity server.
5. Place `ivyconf.xml` in the directory where your `build.xml` will be running.
6. In the same directory create the `ivy.xml` file defining which artifacts to download and where to put them, for example:

```
<ivy-module version="1.3">
  <info organisation="YOUR_ORGANIZATION" module="YOUR_MODULE"/>
  <dependencies>
    <dependency org="org" name="BUILD_TYPE_EXT_ID" rev="BUILD_REVISION">
      <include name="ARTIFACT_FILE_NAME_WITHOUT_EXTENSION"
ext="ARTIFACT_FILE_NAME_EXTENSION" matcher="exactOrRegexp"/>
    </dependency>
  </dependencies>
</ivy-module>
```

Where:

- `YOUR_ORGANIZATION` replace with the name of your organization.
- `YOUR_MODULE` replace with the name of your project or module where artifacts will be used.
- `BUILD_TYPE_EXT_ID` replace with the external ID of the build configuration whose artifacts are downloaded.
- `BUILD_REVISION` can be either a build number or one of the following strings:
  - `latest.lastFinished`
  - `latest.lastSuccessful`
  - `latest.lastPinned`
  - `TAG_NAME.tcbuildtag` - last build tagged with the TAG_NAME tag
- `ARTIFACT_FILE_NAME_WITHOUT_EXTENSION` file name or regular expression of the artifact without the extension part.
- `ARTIFACT_FILE_NAME_EXTENSION` the extension part of the artifact file name.

7. Modify your `build.xml` file and add tasks for downloading artifacts, for example (applicable for Ant 1.6 and later):

```
<target name="fetchArtifacts" description="Retrieves artifacts for TeamCity"
xmlns:ivy="antlib:org.apache.ivy.ant">
    <taskdef uri="antlib:org.apache.ivy.ant"  resource="org/apache/ivy/ant/antlib.xml"/>
      <classpath>
        <pathelement location="${basedir}/lib/ivy-2.0.jar"/>
        <pathelement location="${basedir}/lib/commons-httpclient-3.0.1.jar"/>
        <pathelement location="${basedir}/lib/commons-logging.jar"/>
        <pathelement location="${basedir}/lib/commons-codec-1.3.jar"/>
      </classpath>
    </taskdef>
    <ivy:configure file="${basedir}/ivyconf.xml" />
    <!--<ivy:cleancache />-->
    <ivy:retrieve pattern="${basedir}/[artifact].[ext]"/>
  </target>
```


- `commons-httpclient`, `commons-logging` and `commons-codec` are to be in the `classpath` of Ivy tasks.
- To clean the Ivy cache directory before retrieving dependencies, uncomment the `<ivy:cleancache />` element in the example above.

Artifacts repository is protected by a basic authentication. To access the artifacts, you need to provide credentials to the <ivy:configure/> task. For example:

```
<ivy:configure file="${basedir}/ivyconf.xml"
               host="TEAMCITY_HOST"
               realm="TeamCity"
               username="USER_ID"
               passwd="PASSWORD"/>
```

where `TEAMCITY_HOST` is hostname or IP address of your TeamCity server (without port and servlet context).
As `USER_ID/PASSWORD` you can use either username/password of a regular TeamCity user (the user should have corresponding permissions to access artifacts of the source build configuration) or system properties `teamcity.auth.userId/teamcity.auth.password`.

Build-level authentication

The system properties teamcity.auth.userId and teamcity.auth.password store automatically generated build-unique values which can be used to authenticate on TeamCity server. The values are valid only during the time the build is running. This generated user has limited permissions which allow build-related operations. The primary intent for the user is to use the authentication to download artifacts from other TeamCity builds within the build script.

Using the properties is preferable to using real user credentials since it allows the server to track the artifacts downloaded by your build. If the artifacts were downloaded by the build configuration artifact dependencies or using the supplied properties, the specific artifacts used by the build will be displayed at the Dependencies tab on the build results page. In addition, the builds which were used to get the artifacts from, can be configured to have different clean-up logic.

See also:

Concepts: Dependent Build