REST API

On this page:

General information

REST API is an open-source plugin bundled since TeamCity 5.0.

To use the REST API, an application makes an HTTP request to the TeamCity server and parses the response.

The TeamCity REST API can be used for integrating applications with TeamCity and for those who want to script interactions with the TeamCity server. TeamCity's REST API allows accessing resources (entities) via URL paths.

> The URL examples on this page assume that your TeamCity server web UI is accessible via the `http://teamcity:8111` URL.

General Usage Principles

This documentation is not meant to be comprehensive, but just provide some initial knowledge useful for using the API.

You can start by opening `http://teamcity:8111/app/rest` URL in your browser: this page will give you several pointers to explore the API.

Use http://teamcity:8111/app/rest/application.wadl to get the full list of supported requests and names of parameters. This is the primary source of discovery for the supported requests and their parameters. The same data is also exposed in Swagger format via .../app/rest/swagger.json endpoint You can start with `http://teamcity:8111/app/rest/server` request and then drill down following "`href`" attributes of the entities listed.
Please make sure you read through this "General information" section before using the API.

For the list of supported locator dimensions, use "$help" locator.

Experiment and read the error messages returned: for the most part they should guide you to the right requests.

Example on how to explore the API

Suppose you want to know more on the agents and see (in "`/app/rest/server`" response) that there is a "`/app/rest/agents`" URL.

- try the "`/app/rest/agents/`" request - see the authorized agent list, get the "`default`" way of linking to an agent from the agent's element `href` attribute.
- get individual agent details via `/app/rest/agents/id:10` URL (obtained from "`href`" for one of the elements of the previous request).
- if you send a request to "`/app/rest/agents/$help, or "/app/rest/agents/aaa:bbb`" (supplying unsupported locator dimension), you will get the list of the supported dimensions to find an agent via the agent's locator
- most of the attributes of the returned agent data (name, connected, authorized) can be used as "`<field name>`" in the "`app/rest/agents/<agentLocator>/<field name>`" request. Moreover, if you issue a request to the "`app/rest/agents/id:10/test`" URL, you will get a list of the supported fields in the error message

REST Authentication

You can authenticate yourself for the REST API in the following ways:

- Using basic HTTP authentication (it can be slow with certain authentications, see below). Provide a valid TeamCity username and password with the request. You can force basic auth by including "`httpAuth`" before the "`/app/rest`" part: e.g. `http://teamcity:8111/httpAuth/app/rest/builds`
- Using access to the server as a guest user (if enabled) include "`guestAuth`" before the "`/app/rest`" part: e.g.: `http://teamcity:8111/guestAuth/app/rest/builds`
- if you are checking REST `GET` requests from within a browser and you are logged in to TeamCity in the browser, you can just use "`/app/rest`" URL: e.g. `http://teamcity:8111/app/rest/builds`

Authentication can be slow when not built-in authentication module is used, consider applying the session reuse approach for reusing authentication between sequential requests.

If you perform a request from within a TeamCity build, for a limited set of build-related operations (like downloading artifacts) you can use values of `teamcity.auth.userId/teamcity.auth.password` system properties as credentials (within TeamCity settings you can reference them as `%system.teamcity.auth.userId%` and `%system.teamcity.auth.password%`).

Within a build, a request for current build details can look like:
curl -u "%system.teamcity.auth.userId%:%system.teamcity.auth.password%"
"%teamcity.serverUrl%/httpAuth/app/rest/builds/id:%teamcity.build.id%"

Superuser access

You can use the super user account with REST API: just provide no user name and the generated password logged into the server log.

REST API Versions

As REST API evolves from one TeamCity version to another, there can be incompatible changes in the protocol.

Under the `http://teamcity:8111/app/rest/` or `http://teamcity:8111/app/rest/latest` URL the latest version is available.
Under the `http://teamcity:8111/app/rest/<version>` URL, the current version is available and earlier versions CAN be available. Our general policy is to supply TeamCity with at least one previous version.
e.g. in TeamCity 10.x for <version> you can use "10.0" for the current and "9.1", "9.0", "8.1", "6.0" to get earlier versions of the protocol. Protocol version corresponds to the TeamCity version where it was first introduced.

Breaking changes in the API are described in the related Upgrade Notes section.
Please note that additions to the objects returned (such as new XML attributes or elements) are not considered major changes and do not cause the protocol version to increment.
Also, the endpoints marked with "`Experimental`" comment in `application.wadl` may change without a special notice in future versions.

Note: The examples on this page use the "`/app/rest`" relative URL, replace it with the one containing the version if necessary.

URL Structure

The general structure of the URL in the TeamCity API is `teamcityserver:port/<authType>/app/rest/<apiVersion>/<restApiPath>?<parameters>`, where

- `teamcityserver` and `port` define the server name and the port used by TeamCity. This page uses "`http://teamcity:8111/`" as example URL
- `<authType>` (optional) is the authentication type to be used, this is generic TeamCity functionality
- `app/rest` is the root path of TeamCity REST API
- `<apiVersion>` (optional) is a reference to specific version of REST API
- `<restApiPath>?<parameters>` is the REST API part of the URL

When `<restApiPath>` represents a collection of items `.../app/rest/<items>` (e.g. `.../app/rest/builds`), then the URL regularly accepts the "locator" parameter which can filter the items returned. Individual items can regularly be addressed by a URL in the form of `.../app/rest/<itmes>/<item_locators>`. Both multiple and single items requests regularly support the fields parameter.

Locator

In a number of places, you can specify a filter string which defines what entities to filter/affect in the request. This string representation is referred to as "locator" in the scope of REST API.

The locators formats can be:

- single value: a string without the following symbols: `,:-( )`
- dimension, allowing to filter entities using multiple criteria: `<dimension1>:<value1>,<dimension2>:<value2>,<dimension3>:(<dimension3.1>:<value3.1>,<dimension3.2>:<value3.2>)`

Refer to each entity description below for the most popular locator descriptions.
If in doubt what a specific locator supports, send a request with "$help" as the locator value. In response you will get a textual description of what the locator supports. If a request with invalid locators is sent, the error messages often hint at the error and list the supported locator dimensions as well.

Note: If the value contains the "," symbol, it should be enclosed into parentheses: "(<value>)". The value of a dimension can also be encoded as Base64url and sent as "<dimension>:($base64:<base64-encoded-value >)" instead of " <dimension>: <value>".

Examples:

`http://teamcity:8111/app/rest/projects` gets you the list of projects
`http://teamcity:8111/app/rest/projects/<projectsLocator>`- `http://teamcity:8111/app/rest/projects/id:RESTAPIPlugin` (the example id is used) gets you the full data for the REST API Plugin project.
`http://teamcity:8111/app/rest/buildTypes/id:bt284/builds?locator=<buildLocator>` - `http://teamcity:8111/app/rest/buildTypes/id:bt284/builds?locator=status:SUCCESS,tag:EAP` - (example ids are used) to get builds
`http://teamcity:8111/app/rest/builds/?locator=<buildLocator>` - to get builds by build locator.

Supported HTTP Methods

- GET: retrieves the requested data. e.g. usually `.../app/rest/entities` retrieves a list of entities, `.../app/rest/entities/<entity locator>` retrieves a single entity
- POST: creates the entity in the request adding it to the existing collection. When posting XML, be sure to specify the "`Content-Type: application/xml`" HTTP header. e.g. to create a new entity, one regularly needs to post a single entity data to the `.../app/rest/entities` URL
- PUT: based on the existence of the entity, creates or updates the entity in the request. e.g. supported for some entities, for URLS like `.../app/rest/entities/<entity locator>`
- DELETE: removes the requested data e.g. for the `.../app/rest/entities/<entity locator>` URL

Response Formats

The TeamCity REST APIs returns HTTP responses in the following formats according to the HTTP "Accept" header:

Format	Response Type	HTTP "Accept" header value
plain text	single-value responses	text/plain
XML	complex value responses	application/xml
JSON	complex value responses	application/json

Full and Partial Responses

By default, when a list of entities is requested, only basic fields are included into the response. When a single entry is requested, all the fields are returned. The complex field values can be returned in full or basic form, depending on a specific entity.

It is possible to change the set of fields returned for XML and JSON responses for the majority of requests.
This is done by supplying the fields request parameter describing the fields of the top-level entity and sub-entities to return in the response. An example syntax of the parameter is: `field,field2(field2_subfield1,field2_subfield1)`. This basically means "include field and field2 of the top-level entity and for field2 include field2_subfield1 and field2_subfield1 fields". The order of the fields specification plays no role.Examples:
`http://teamcity.jetbrains.com/app/rest/buildTypes?locator=affectedProject:(id:TeamCityPluginsByJetBrains)&fields=buildType(id,name,project)`
`http://teamcity.jetbrains.com/app/rest/builds?locator=buildType:(id:bt345),count:10&fields=count,build(number,status,statusText,agent,lastChange,tags,pinned)`

At this time, the response can sometimes include the fields/elements not specifically requested. This can change in the future versions, so it is recommended to specify all the fields/elements used by the client.

Logging

You can get details on errors and REST request processing in `logs\teamcity-rest.log` server log.

If you get an error in response to your request and want to investigate the reason, look into rest-related server logs.

To get details about each processed request, turn on debug logging (e.g. set Logging Preset to "debug-rest" on the Administration/Diagnostics page or modify the Log4J "`jetbrains.buildServer.server.rest`" category) .

CORS Support

TeamCity REST can be configured to allow cross-origin requests using the `rest.cors.origins` internal property.

To allow requests from a page loaded from a specific domain:

- Add the page address (including the protocol and port , do not use wildcards) to the comma-separated internal property `rest.cors.origins`, e.g.

  rest.cors.origins=http://myinternalwebpage.org.com:8080,https://myinternalwebpage.org.com

To enable support for a preflight OPTIONS request:

1. Add the `rest.cors.optionsRequest.allowUnauthorized=true` internal property.
2. Restart the TeamCity server.
3. Use the '**`/app/rest/latest`**' URL for the requests ⚠ Do not use '`/app/rest`', do not use the '`httpAuth`' prefix.

If that does not help, enable debug logging and look for related messages. If there are none, capture the browser traffic and messages to investigate the case.

API Client Recommendations

When developing a client using REST API, consider the following recommendations:

- Make root REST API URL configurable (e.g. allow to specify an alternative for "`app/rest/<version>`" part of the URL). This will allow to direct the client to another version of the API if necessary.
- Ignore (do not error out) item's attributes and sub-items which are unknown to the client. New sub-items are sometimes added to the API without version change and this will ensure the client is not affected by the change.
- Set large (and make them configurable) request timeouts. Some API calls can take minutes, especially on a large server.
- Use HTTP sessions to make consecutive requests (use TCSESSIONID cookie returned from the first authenticated response instead of supplying raw credentials all the time). This saves time on authentication which can be significant for external authentication providers.
- Beware of partial answers when requesting list of items: some requests are paged by default. Value of the "count" attribute in the response indicate the number of the items on the current page and there can be more pages available. If you need to process more (e.g. all) items, read and process "nextHref" attribute of the response entity for items collections. If the attribute is present it means there might be more items when queried by the URL provided. Related locator dimensions are "count" (page limit) and "lookupLimit" (depth of search). Even when the returned "count" is 0, it does not mean there are no more items if there is "nextHref" attribute present.
- Do not increase the "`lookupLimit`" value in the locators without a second thought. Doing so has the direct effect of loading the server more and may require increased amounts of CPU and memory. It is assumed that those increasing the default limit understand the negative consequences for the server performance.
- Do not abuse the ability to execute automated requests for TeamCity API: do not query the API too frequently and restrict the data requested to only that necessary (using due locators and specifying necessary fields). Check the server behavior under load from your requests. Make sure not to repeat the request frequently if it takes time to process the request.

TeamCity Data Entities Requests

Projects and Build Configuration/Templates Lists

List of projects: `GET http://teamcity:8111/app/rest/projects`
Project details: `GET http://teamcity:8111/app/rest/projects/<projectLocator>` where `<projectLocator>` can be `id:<internal_project_id>` or `name:<project%20name>`

List of Build Configurations: `GET http://teamcity:8111/app/rest/buildTypes`
List of Build Configurations of a project: `GET http://teamcity:8111/app/rest/projects/<projectLocator>buildTypes`

Get projects with sub-projects/ Build Configurations data and their order as configured by the specified user on the Overview page: `GET http://teamcity:8111/app/rest/projects?locator=selectedByUser:current&fields=count,project(id,parentProjectId,projects(count,project(id),$locator(selectedByUser:current)),buildTypes(count,buildType(id),$locator(selectedByUser:current)))`

List of templates for a particular project: `GET http://teamcity:8111/app/rest/projects/<projectLocator>/templates`
List of all the templates on the server: `GET http://teamcity:8111/app/rest/buildTypes?locator=templateFlag:true`

Project Settings

Get project details: `GET http://teamcity:8111/app/rest/projects/<projectLocator>/`
Delete a project: `DELETE http://teamcity:8111/app/rest/projects/<projectLocator>/`
Create a new empty project: `POST` plain text (name) to `http://teamcity:8111/app/rest/projects/`
Create (or copy) a project: `POST XML <newProjectDescription name='New Project Name' id='newProjectId' copyAllAssociatedSettings='true'><parentProject locator='id:project1'/><sourceProject locator='id:project2'/></newProjectDescription>` to `http://teamcity:8111/app/rest/projects`. Also see an example.
Edit project parameters: `GET/DELETE/PUT http://teamcity:8111/app/rest/projects/<projectLocator>/parameters/<parameter_name>` (produces XML, JSON and plain text depending on the "Accept" header, accepts plain text and XML and JSON)
Also supported are requests `.../parameters/<parameter_name>/name` and `.../parameters/<parameter_name>/value`.

Project name/description/archived status: `GET/PUT http://teamcity:8111/app/rest/projects/<projectLocator>/<field_name>` (accepts/produces text/plain) where `<field_name>` is one of "`name`", "`description`", "`archived`".

Project's parent project: GET/PUT XML http://teamcity:8111/app/rest/projects/<projectLocator>/parentProject

Project Features

Project features (e.g. issue trackers, versioned settings, custom charts, shared resources and third-party report tabs) are exposed as entries under the "project" node and via dedicated requests.

List of project features: http://teamcity:8111/app/rest/projects/<projectLocator>/projectFeatures To filter features, add "?locator=<projectFeaturesLocator>" to the URL e.g. to find all issue tracker features of GitHub type, use the locator "type:IssueTracker,property(name:type,value:GithubIssues)"

Create feature: POST to http://teamcity:8111/app/rest/projects/<projectLocator>/projectFeatures

Edit features: GET/DELETE/PUT http://teamcity:8111/app/rest/projects/<projectLocator>/projectFeatures/<featureId>

VCS Roots

List all VCS roots: GET http://teamcity:8111/app/rest/vcs-roots, add locator=<vcsRootLocator> parameter to list only the VCS roots matched
Get details of a VCS root/delete a VCS root: GET/DELETE http://teamcity:8111/app/rest/vcs-roots/<vcsRootLocator> , where "<vcsRootLocator>" can be "id:<internal VCS root id>" or other VCS root locator
Create a new VCS root: POST VCS root XML (similar to the one retrieved by a GET request for VCS root details) to http://teamcity:8111/app/rest/vcs-roots

Also supported:

GET/PUT http://teamcity:8111/app/rest/vcs-roots/<vcsRootLocator>/properties/<property_name>
GET/PUT http://teamcity:8111/app/rest/vcs-roots/<vcsRootLocator>/<field_name>, where <field_name> is "id", "name", "project" (post project locator to "project" to associate a VCS root with a specific project).

List VCS root instances: GET http://teamcity:8111/app/rest/vcs-root-instances?locator=<vcsRootInstancesLocator>

A 'VCS root' is the setting configured in the TeamCity UI, a "VCS root instance" is an internal TeamCity entity which is derived from the "VCS root" to perform the actual VCS operation.
If a VCS root has no %-references to parameters, a single VCS root corresponds to a single "VCS root instance".
If a VCS root has %-reference to a parameter and the reference resolves to a different value when the VCS root is attached to different configurations or when custom builds are run, a single "VCS root" can generate several "VCS root instances".

Since TeamCity 10.0:

There are two endpoints dedicated to being used in commit hooks from the version control repositories:
POST http://teamcity:8111/app/rest/vcs-root-instances/checkingForChangesQueue?locator=<vcsRootInstancesLocator> - schedules checking for changes for the matched VCS root instances and returns the list of VCS root instances matched (just like GET http://teamcity:8111/app/rest/vcs-root-instances?locator=<vcsRootInstancesLocator>)
POST http://teamcity:8111/app/rest/vcs-root-instances/commitHookNotification?locator=<vcsRootInstancesLocator> - schedules checking for changes for the matched VCS root instances and returns plain-text human-readable message on the action performed, HTTP response 202 in case of successful operation
Both perform the same action (put the VCS root instances matched by the <locator>) to the queue for "checking for changes" process and differ only in responses they produce.
Note that since the matched VCS root instances are the same as for .../app/rest/vcs-root-instances?locator=<locator> request and that means that by default only the first 100 are matched and the rest are ignored. If this limit is reached, consider tweaking the <locator> to match fewer instances (recommended) or increase the limit, e.g. by adding ",count:1000" to the locator.

VCS root instance locator

Some of the supported "<vcsRootInstancesLocator>" from above:
type:<VCS root type> - VCS root instances of the specified version control (e.g. "jetbrains.git", "mercurial", "svn")
vcsRoot:(<vcsRootLocator>) - VCS root instances corresponding to the VCS root matched by "<vcsRootLocator>"
buildType:(<buildTypeLocator>) - VCS root instances attached to the matching build configuration
property:(name:<name>,value:<value>,matchType:<matching>) - VCS root instances with the property of name "<name>" and value matching condition "<matchType>" (e.g. equals, contains) by the value "<value>".

Build Configuration And Template Settings

Build Configuration/Template details: GET `http://teamcity:8111/app/rest/buildTypes/<buildConfigurationLocator>` (details on the Build Configuration locator).

Please note that there is no transaction, etc. support for settings editing in TeamCity, so all the settings modified via REST API are taken into account at once. This can result in half-configured builds triggered, etc. Please make sure you pause a build configuration before changing its settings if this aspect is important for your case.

To get aggregated status for several build configurations, see Build Status Icon section.

Get/set paused build configuration state: `GET/PUT http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/paused` (put "true" or "false" text as text/plain)

Build configuration settings: `GET/DELETE/PUT http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/settings/<setting_name>`
Build configuration parameters: `GET/DELETE/PUT http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/parameters/<parameter_name>`
(produces XML, JSON and plain text depending on the "Accept" header, accepts plain text and XML and JSON). The requests `.../parameters/<parameter_name>/name` and `.../parameters/<parameter_name>/value` are also supported.

Build configuration steps: `GET/DELETE http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/steps/<step_id>`

Create build configuration step: POST `http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/steps`. The XML/JSON posted is the same as retrieved by GET request to `.../steps/<step_id>` except for the secure settings like password: these are not included into responses and should be supplied before POSTing back

Features, triggers, agent requirements, artifact and snapshot dependencies follow the same pattern as steps with URLs like:
`http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/features/<id>`
`http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/triggers/<id>`
`http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/agent-requirements/<id>`
`http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/artifact-dependencies/<id>`
`http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/snapshot-dependencies/<id>`

Since TeamCity 10, it is possible to disable/enable artifact dependencies and agent requirements:
Disable/enable an artifact dependency PUT `http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/artifact-dependencies/<id>/disabled` (put " true " or "false" text as text/plain)
Disable/enable an agent requirement PUT `http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/agent-requirements/<id>/disabled` (put " true " or "false" text as text/plain)

Build configuration VCS roots: GET/DELETE `http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/vcs-root-entries/<id>`
Attach VCS root to a build configuration: POST `http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/vcs-root-entries`. The XML/JSON posted is the same as retrieved by GET request to `http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/vcs-root-entries/<id>` except for the secure settings like password: these are not included into responses and should be supplied before POSTing back.

Create a new build configuration with all settings: POST `http://teamcity:8111/app/rest/buildTypes`. The XML/JSON posted is the same as retrieved by GET request. (Note that `/app/rest/project/XXX/buildTypes` still uses the previous version notation and accepts another entity.)

Create a new empty build configuration: POST plain text (name) to `http://teamcity:8111/app/rest/projects/<projectLocator>/buildTypes`

Copy a build configuration: POST XML `<newBuildTypeDescription name='Conf Name' sourceBuildTypeLocator='id:XXX' copyAllAssociatedSettings='true' shareVCSRoots='false'/>` to `http://teamcity:8111/app/rest/projects/<projectLocator>/buildTypes`

Since TeamCity 2017.2: Read, detach and attach a build configuration from/to a template: `GET/DELETE/POST/PUT http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/templates`
Before 2017.2: Read, detach and attach a build configuration from/to a template: `GET/DELETE/PUT http://teamcity:8111/app/rest/buildTypes/<buildTypeLocator>/template` (`PUT` accepts template locator with the "text/plain" Content-Type)
Some examples: click to expand

Set build number counter:
curl -v --basic --user <username>:<password> --request PUT
http://<teamcity.url>/app/rest/buildTypes/<buildTypeLocator>/settings/buildNumbe
rCounter --data <new number> --header "Content-Type: text/plain"

Set build number format:
curl -v --basic --user <username>:<password> --request PUT
http://<teamcity.url>/app/rest/buildTypes/<buildTypeLocator>/settings/buildNumbe
rPattern --data <new format> --header "Content-Type: text/plain"

Build Configuration Locator

The most frequently used values for "<buildTypeLocator>" are id:<buildConfigurationOrTemplate_id> and name:<Build%2
0Configuration%20name>.

Since TeamCity 2017.2, the experimental type locator is supported with one of the values: regular, composite or deployment

Other supported dimensions are (these are in experimental state):
internalId - internal id of the build configuration
project - <projectLocator> to limit the build configurations to those belonging to a single project
affectedProject - <projectLocator> to limit the build configurations under a single project (recursively)
template - <buildTypeLocator> of a template to list only build configurations using the template
templateFlag - boolean value to get only templates or only non-templates
paused - boolean value to filter paused/not paused build configurations

Build Requests

List builds: GET http://teamcity:8111/app/rest/builds/?locator=<buildLocator>
Get details of a specific build: GET http://teamcity:8111/app/rest/builds/<buildLocator> (also supports DELETE to delete a build)

Get the list of build configurations in a project with the status of the last finished build in each build configuration:
GET http://teamcity:8111/app/rest/buildTypes?locator=affectedProject:(id:ProjectId)&fields=buildType(id,nam
e,builds($locator(running:false,canceled:false,count:1),build(number,status,statusText)))

Build Locator

Using a locator in build-related requests, you can filter the builds to be returned in the build-related requests. It is referred to as "build locator" in the scope of REST API.

For some requests, a default filtering is applied which returns only "normal" builds (finished builds which are not canceled, not failed-to-start, not personal, and on default branch (in branched build configurations)), unless those types of builds are specifically requested via the locator. To turn off this default filter and process all builds, add "defaultFilter:false" dimension to the build locator. Default filtering varies depending on the specified locator dimensions. e.g. when "agent" or "user" dimensions are present, personal, canceled and failed to start builds are included into the results.

Examples of supported build locators:

- id:<internal build id> - use internal build id when you need to refer to a specific build
- number:<build number> - to find build by build number, provided build configuration is already specified
- <dimension1>:<value1>,<dimension2>:<value2> - to find builds by multiple criteria

The list of supported build locator dimensions:

project:<project locator> - limit the list to the builds of the specified project (belonging to any build type directly under the project).
affectedProject:<project locator> - limit the list to the builds of the specified project (belonging to any build type directly or indirectly under the project)
buildType:(<buildTypeLocator>),defaultFilter:false - all the builds of the specified build configuration

tag:<tag> - since TeamCity 10 get tagged builds. If a list of tags is specified, e.g. tag:<tag1>, tag:<tag2>, only the builds

containing all the specified tags are returned. The legacy tags:<tags> locator is supported for compatibility

status:<SUCCESS/FAILURE/ERROR> - list builds with the specified status only
user:(<userLocator>) - limit builds to only those triggered by the user specified
personal:<true/false/any> - limit builds by the personal flag. By default, personal builds are not included.

canceled:<true/false/any> - limit builds by the canceled flag. By default, canceled builds are not included.
failedToStart:<true/false/any> - limit builds by the failed to start flag. By default, canceled builds are not included.

state: <queued/running/finished> - - limit builds by the specified state.
running:<true/false/any> - limit builds by the running flag. By default, running builds are not included.
state:running,hanging:true - fetch hanging builds (since TeamCity 10.0)

pinned:<true/false/any> - limit builds by the pinned flag.

branch:<branch locator> - limit the builds by branch. <branch locator> can be the branch name displayed in the UI, or "(name:<name>,default:<true/false/any>,unspecified:<true/false/any>,branched:<true/false/any>)". By default only builds from the default branch are returned. To retrieve all builds, add the following locator: branch:default:any. The whole path will look like this: /app/rest/builds/?locator=buildType:One_Git,branch:default:any

revision:<REVISION> - find builds by revision, e.g. all builds of the given build configuration with the revision: /app/rest/builds?locator=revision:(REVISION),buildType:(id:BUILD_TYPE_ID). See more information below.
agentName:<name> - agent name to return only builds ran on the agent with name specified

sinceBuild:(<buildLocator>) - limit the list of builds only to those after the one specified
sinceDate:<date> - limit the list of builds only to those started after the date specified. The date should be in the same format as dates returned by REST API (e.g. "20130305T170030+0400").
queuedDate/startDate/finishDate:(date:<time-date>,build:<build locator>,condition:<before/after>) - filter builds based on the time specified by the build locator, e.g. (finishDate:(date:20151123T203446+0100,condition:after) - (finished after November 23, 2015, 20:34:46)

count:<number> - serve only the specified number of builds
start:<number> - list the builds from the list starting from the position specified (zero-based)
lookupLimit:<number> - limit processing to the latest N builds only (the default is 5000). If none of the latest N builds match the other specified criteria of the build locator, 404 response is returned for single build request and empty collection for multiple builds request. See related note in the section abov

Queued Builds

GET http://teamcity:8111/app/rest/buildQueue

Supported locators:

- project:<locator>
- buildType:<locator>

Get details of a queued build:
GET http://teamcity:8111/app/rest/buildQueue/id:XXX

For queued builds with snapshot dependencies, the revisions are available in the revisions element of the queued build node if a revision is fixed (for regular builds without snapshot dependencies it is not).

Get compatible agents for queued builds (useful for builds having "No agents" to run on)
GET http://teamcity:8111/app/rest/buildQueue/id:XXX/compatibleAgents

Examples:
List queued builds per project:
GET http://teamcity:8111/app/rest/buildQueue?locator=project:<locator>

List queued builds per build configuration:
GET http://teamcity:8111/app/rest/buildQueue?locator=buildType:<locator>

Triggering a Build

To start a build, send a POST request to http://teamcity:8111/app/rest/buildQueue with the "build" node (see below) in content - the same node as details of a queued build or finished build. The queued build details will be returned.

When the build is started, the request to the queued build (/app/rest/buildQueue/XXX) will return running/finished build data. This way, you can monitor the build completeness by querying build details using the "href" attribute of the build details

returned on build triggering, until the build has the `state="finished"` attribute.

Build node example

Basic build for a build configuration:

```
<build>
   <buildType id="buildConfID"/>
</build>
```

Build for a branch marked as personal with a fixed agent, comment and a custom parameter:

```
<build personal="true" branchName="logicBuildBranch">
   <buildType id="buildConfID"/>
   <agent id="3"/>
   <comment><text>build triggering comment</text></comment>
   <properties>
      <property name="env.myEnv" value="bbb"/>
   </properties>
</build>
```

Queued build assignment to an agent pool:

```
<build>...
  <agent>
    <pool id="N"/>
  </agent>
...
</build>
```

Build on a specified change, forced rebuild of all dependencies and clean sources before the build, moved to the build queue top on triggering. (Note that the change is set via the change's internal modification id; see more below):

```
<build>
  <triggeringOptions cleanSources="true" rebuildAllDependencies="true" queueAtTop="true"/>
  <buildType id="buildConfID"/>
  <lastChanges>
    <change id="modificationId" personal="false"/>
  </lastChanges>
</build>
```

Example command line for the build triggering: click to expand

```
curl -v -u user:password http://teamcity.server.url:8111/app/rest/buildQueue
--request POST --header "Content-Type:application/xml" --data-binary @build.xml
```

Build Tags

Get tags: `GET http://teamcity:8111/app/rest/builds/<buildLocator>/tags/`
Replace tags: `PUT http://teamcity:8111/app/rest/builds/<buildLocator>/tags/` (put the same XML or JSON as returned by GET)

Add tags: `POST http://teamcity:8111/app/rest/builds/<buildLocator>/tags/` (post the same XML or JSON as returned by GET or just a plain-text tag name)
(`<buildLocator>` here should match a single build only)

Build Pinning

Get current pin status: GET `http://teamcity:8111/app/rest/builds/<buildLocator>/pin/` (returns "true" or "false" text)
Pin: `PUT http://teamcity:8111/app/rest/builds/<buildLocator>/pin/` (the text in the request data is added as a comment for the action)
Unpin: `DELETE http://teamcity:8111/app/rest/builds/<buildLocator>/pin/` (the text in the request data is added as a comment for the action)
(`<buildLocator>` here should match a single build only)

Build Canceling/Stopping

Cancel a running or a queued build: POST the `<buildCancelRequest comment='CommentText' readdIntoQueue='false' />` item to the URL of a running or a queued build:

Example of cancelling a queued build: click to expand

```
curl -v -u user:password --request POST "http://teamcity:8111/app/rest/buildQueue/<buildLocator >"
--data "<buildCancelRequest comment='' readdIntoQueue='false' />" --header "Content-Type:
application/xml"
```

Stop a running build and readd it to the queue: POST the `<buildCancelRequest comment='CommentText' readdIntoQueue='true' />` item to the URL of a running build:

Example of cancelling a running build: click to expand

```
curl -v -u user:password --request POST "http://teamcity:8111/app/rest/builds/<buildLocator >" --data
"<buildCancelRequest comment='' readdIntoQueue='true' />" --header "Content-Type: application/xml"
```

Expose cancelled build details:

See the `canceledInfo` element of the build item (available via `GET http://teamcity:8111/app/rest/builds/<buildLocator>`)

Build Artifacts

`GET http://teamcity:8111/app/rest/builds/<build_locator>/artifacts/content/<path>` (returns the content of a build artifact file for a build determined by <buid_locator>)

`<path>` above can be empty for the root of build's artifacts or be a path within the build's artifacts. The path can span into the archive content, e.g. `dir/path/archive.zip!/path_within_archive`

Media-Type: application/octet-stream or a more specific media type (determined from artifact file extension)
Possible error: 400 if the specified path references a directory

`GET http://teamcity:8111/app/rest/builds/<build_locator>/artifacts/metadata/<path>` (returns information about a build artifact)
Media-Type: application/xml or application/json

`GET http://teamcity:8111/app/rest/builds/<build_locator>/artifacts/children/<path>` (returns the list of artifact children for directories and archives)
Media-Type: application/xml or application/json
Possible error: 400 if the artifact is neither a directory nor an archive

`GET http://teamcity:8111/app/rest/builds/<build_locator>/artifacts/archived/<path>?locator=pattern:<wildcard>` (returns the archive containing the list of artifacts under the path specified. The optional locator parameter can have file <wildcard> to limit the files only to those matching the wildcard)
Media-Type: application/zip
Possible error: 400 if the artifact is neither a directory nor an archive`<artifact relative name>` supports referencing files under archives using "!/" delimiter after the archive name.

Examples:

`GET http://teamcity:8111/app/rest/builds/id:100/artifacts/children/my-great-tool-0.1.jar\!/META-INF`
`GET http://teamcity:8111/app/rest/builds/buildType:(id:Build_Intallers),status:SUCCESS/artifacts/metadata/my-great-tool-0.1.jar\!/META-INF/MANIFEST.MF`

GET http://teamcity:8111/app/rest/builds/buildType:(id:Build_Intallers),number:16.7.0.2/artifacts/metadata/my-great-tool-0.1.jar!/lib/commons-logging-1.1.1.jar!/META-INF/MANIFEST.MF
GET http://teamcity:8111/app/rest/builds/buildType:(id:Build_Intallers),tag:release/artifacts/content/my-great-tool-0.1.jar!/lib/commons-logging-1.1.1.jar!/META-INF/MANIFEST.MF

Authentication

If you download artifacts from within a TeamCity build, consider using `teamcity.auth.userId`/`teamcity.auth.password` system properties as credentials for the download artifacts request: this way TeamCity will have a way to record that one build used artifacts of another and will display it on the build's Dependencies tab.

Other Build Requests

Changes

<changes>

<changes> is meant to represent changes the same way as displayed in the build's Changes in TeamCity UI. In the most cases these are the commits between the current and previous build. The <changes> tag is not included into the build by default, it has the href attribute only. If you execute the request specified in the href, you'll get the required changes.

Get the list of all of the changes included into the build: GET http://teamcity:8111/app/rest/changes?locator=build:(id:<buildId>)
Get details of an individual change: GET http://teamcity:8111/app/rest/changes/id:changeId
Get information about a changed file action: the files node lists changed files. The information about the changed file action is reported via the changeType attribute for the files listed as one of the following: added, edited, removed, copied or unchanged.

Filter all changes by a locator: GET http://teamcity:8111/app/rest/changes?locator=<changeLocator>
Note that the change id is the change's internal id, not the revision. The id can be seen in the change node listed by the REST API or in the URL of the change details (as modId).

Get all changes for a project: GET http://teamcity:8111/app/rest/changes?locator=project:projectId
Get all the changes in a build configuration since a particular change identified by its id: http://teamcity:8111/app/rest/changes?locator=buildType:(id:buildConfigurationId),sinceChange:(id:changeId)
Get pending changes for a build configuration http://teamcity:8111/app/rest/changes?locator=buildType:(id:BUILD_CONF_ID),pending:true

<lastChanges>
The <lastChanges> tag contains information about the last commit included into the build and is only good for re-triggering the build: it contains the TeamCity internal id (the id attribute) associated with the commit, which is necessary for TeamCity to trigger a custom build on the same commit (see the example above).

Revisions

<revisions>

The <revisions> tag the same as revisions table on the build's Changes tab in TeamCity UI: it lists the revisions of all of the VCS repositories associated with this build that will be checked out by the build on the agent.
A revision might or might not correspond to a change known to TeamCity. e.g. for a newly created build configuration and a VCS root, a revision will have no corresponding change.

Get all builds with the specified revision: http://teamcity:8111/app/rest/builds?locator=revision(version:XXXX)

<versionedSettingsRevision>

Since TeamCity 10, <versionedSettingsRevision> is added to represent revision of the versioned settings of the build.

Snapshot dependencies

It is possible to retrieve the entire build chain (all snapshot-dependency-linked builds) for a particular build:

http://teamcity:8111/app/rest/builds?locator=snapshotDependency:(to:(id:XXXX),includeInitial:true),defaultFilter:false
This gets all the snapshot dependency builds recursively for the build with id XXXX

It possible to find all the snapshot-dependent builds for a particular build:
http://teamcity:8111/app/rest/builds?locator=snapshotDependency:(from:(id:XXXX),includeInitial:true),defaul

`tFilter:false`

Artifact dependencies

Since TeamCity 10.0.3, there is an experimental ability to:

- get all the builds which downloaded artifacts from the build with the given ID (Delivered artifacts in the TeamCity Web UI):
  `GET http://teamcity:8111/app/rest/builds?locator=artifactDependency:(from:(id:<build ID>),recursive:false)`
- get all the builds whose artifacts were downloaded by the build with the given ID (Downloaded artifacts in the TeamCity Web UI):
  `GET http://teamcity:8111/app/rest/builds?locator=artifactDependency:(to:(id:<build ID>),recursive:false)`

Build Parameters

Get the parameters of a build: `http://teamcity:8111/app/rest/builds/id:<build id>/resulting-properties`

Build fields

Get single build's field: GET `http://teamcity:8111/app/rest/builds/<buildLocator>/<field_name>` (accepts/produces text/plain) where `<field_name>` is one of "number", "status", "id", "branchName" and other build's bean attributes

Statistics

Get statistics for a single build: GET `http://teamcity:8111/app/rest/builds/<buildLocator>/statistics/` only standard/bundled statistic values are listed. See also Custom Charts
Get single build statistics value: GET `http://teamcity:8111/app/rest/builds/<buildLocator>/statistics/<value_name>`

Get statistics for a list of builds: GET `http://teamcity:8111/app/rest/builds?locator=BUILDS_LOCATOR&fields=build(id,number,status,buildType(id,name,projectName),statistics(property(name,value)))`

Build log

Downloading build logs via a REST request is not supported, but there is a way to download the log files described here.

Tests and Build problems

List build problems: GET `http://teamcity:8111/app/rest/problemOccurrences?locator=build:(BUILD_LOCATOR)`

List tests: GET `http://teamcity:8111/app/rest/testOccurrences?locator=<locator dimension>:<value>`

Supported locators:

- `build:(<build locator>)` - test run in the build
- `build:(<build locator>),muted:true` - failed tests which were muted in the build
- `currentlyFailing:true,affectedProject:<project locator>` - tests currently failing under the project specified (recursively)
- `currentlyMuted:true,affectedProject:<project locator>` - tests currently muted under the project specified (recursively) - See also project's Muted Problems tab

Examples:
List all build's tests: `GET http://teamcity:8111/app/rest/testOccurrences?locator=build:<buildLocator>`

Get individual test history:
`GET http://teamcity:8111/app/rest/testOccurrences?locator=test:<testLocator>`

List build's tests which were muted when the build ran:
`GET http://teamcity:8111/app/rest/testOccurrences?locator=build:(id:XXX),muted:true`

List currently muted tests (muted since the failure):

GET http://teamcity:8111/app/rest/testOccurrences?locator=build:(id:XXX),currentlyMuted:true

Supported test locators:

- "id:<internal test id>" available as a part of the URL on the test history page
- "name:<full test name>"

Since TeamCity 10 there is experimental support for exposing single test invocations / runs:

Get invocations of a test:
GET http://teamcity:8111/app/rest/testOccurrences?locator=build:(id:XXX),test:(id:XXX)&fields=$long,testOccurrence($short,invocations($long))

List all test runs with all the invocations flattened:
GET http://teamcity:8111/app/rest/testOccurrences?locator=build:(id:XXX),test:(id:XXX),expandInvocations:true

Muted tests and build problems

(only since TeamCity 2017.2)

List all muted tests and build problems GET http://teamcity:8111/app/rest/mutes
Unmute a test or build problems DELETE http://teamcity:81111/app/rest/mutes/id:XXXX
Mute a test or build problems POST to http://teamcity:8111/app/rest/mutes. Use the same XML or JSON as returned by GET

Investigations

List investigations in the Root project and its subprojects: http://teamcity:8111/app/rest/investigations

Supported locators:

- test: (id:TEST_NAME_ID)
- test: (name:FULL_TEST_NAME)
- assignee: (<user locator>)
- buildType:(id:XXXX)

Get investigations for a specific test:
http://teamcity:8111/app/rest/investigations?locator=test:(id:TEST_NAME_ID)
http://teamcity:8111/app/rest/investigations?locator=test:(name:FULL_TEST_NAME)

Get investigations assigned to a user: http://teamcity:8111/app/rest/investigations?locator=assignee:(<user locator>)

Get investigations for a build configuration: http://teamcity:8111/app/rest/investigations?locator=buildType:(id:XXXX)

Since TeamCity 2017.2 it is possible to assign/replace investigations:

POST/PUT to http://teamcity:8111/app/rest/investigations (accepts a single investigation) and experimental support for multiple investigations: POST/PUT to http://teamcity:8111/app/rest/investigations/multiple (accepts a list of investigations). Use the same XML or JSON as returned by GET.

Agents

List agents (only authorized agents are included by default): GET http://teamcity:8111/app/rest/agents
List all connected authorized agents: GET http://teamcity:8111/app/rest/agents?locator=connected:true,authorized:true
List all authorized agents: GET http://teamcity:8111/app/rest/agents?locator=authorized:true
List all enabled authorized agents: GET http://teamcity:8111/app/rest/agents?locator=enabled:true,authorized:true
List all agents (including unauthorized): GET http://teamcity:8111/app/rest/agents?locator=authorized:any

The request uses default filtering (depending on the specified locator dimensions, others can have default implied value). To disable this filtering, add ",defaultFilter:false" to the locator.

Enable/disable an agent: PUT http://teamcity:8111/app/rest/agents/<agentLocator>/enabled (put " true " or "false" text as text/plain). See an example.
Authorize/unauthorize an agent: PUT http://teamcity:8111/app/rest/agents/<agentLocator>/authorized (put " true " or "false" text as text/plain)

Add a comment when enabling/disabling and authorizing/unauthorising an agent (since TeamCity 10.0):
Agent enabled/authorized data is exposed in the enabledInfo and authorizedInfo nodes:

```
<agent id="1" name="agentName" typeId="1" connected="true" enabled="true" authorized="true"
uptodate="true" ip="....." href="/app/rest/agents/id:1">
 <enabledInfo status="true">
  <comment>
   <user username="userName" id="1" href="/app/rest/users/id:1"/>
   <timestamp>20160406T175040+0300</timestamp>
   <text>newcomment</text>
  </comment>
 </enabledInfo>
 <authorizedInfo status="true">
  <comment>
   <user username="userName" id="1" href="/app/rest/users/id:1"/>
   <timestamp>20160406T183033+0300</timestamp>
  </comment>
 </authorizedInfo>
....
</agent>
```

GET and PUT requests are supported to the following URLs:
http://teamcity:8111/app/rest/agents/<agentLocator>/enabledInfo
http://teamcity:8111/app/rest/agents/<agentLocator>/authorized

On PUT only status and comment/text sub-items are taken into account:

Example of disabling an agent with a comment: click to expand

```
curl -v -u user:password --request PUT "http://teamcity:8111/app/rest/agents/id:1/enabledInfo" --data
"<enabledInfo status='false'><comment><text>commentText</text></comment></enabledInfo>" --header
"Content-Type:application/xml"
```

Get/PUT agent's single field: GET/PUT http://teamcity:8111/app/rest/agents/<agentLocator>/<field name>
Delete a build agent: DELETE http://teamcity:8111/app/rest/agents/<agentLocator>

Agent Pools

Get/modify/remove agent pools:
GET/PUT/DELETE http://teamcity:8111/app/rest/projects/XXX/agentPools/ID

Add an agent pool:
POST the agentPool name='PoolName element to http://teamcity:8111/app/rest/projects/XXX/agentPools

Move an agent to the pool from the previous pool:
POST <agent id='YYY'/> to the pool's agents http://teamcity.url/app/rest/agentPools/id:XXX/agents

Example:

curl -v -u user:password http://teamcity.url/app/rest/agentPools/id:XXX/agents --request POST --header "Content-Type:application/xml" --data "<agent id='1'/>"

Assigning Projects to Agent Pools

Add a project to a pool:
POST the <project> node to http://teamcity.url/app/rest/agentPools/id:XXX/projects

Delete a project from a pool:
DELETE http://teamcity.url/app/rest/agentPools/id:XXX/projects/id:YYY

Users

List of users: `GET` `http://teamcity:8111/app/rest/users`
Get specific user details: `GET` `http://teamcity:8111/app/rest/users/<userLocator>`
Create a user: `POST` `http://teamcity:8111/app/rest/users`
Update/remove specific user: `PUT/DELETE` `http://teamcity:8111/app/rest/users/`
For the `POST` and `PUT` requests for a user, post data in the form retrieved by the corresponding GET request. Only the following attributes/elements are supported: name, username, email, password, roles, groups, properties.
Work with user roles: `http://teamcity:8111/app/rest/users/<userLocator>/roles`

<userLocator> can be of a form:

- `id:<internal user id>` - to reference the user by internal ID
- `username:<user's username>` - to reference the user by username/login name

User's single field: `GET/PUT` `http://teamcity:8111/app/rest/users/<userLocator>/<field name>`
User's single property: `GET/DELETE/PUT` `http://teamcity:8111/app/rest/users/<userLocator>/properties/<property name>`

User Groups

List of groups: `GET` `http://teamcity:8111/app/rest/userGroups`
List of users within a group: `GET` `http://teamcity:8111/app/rest/userGroups/key:Group_Key`

Create a group: `POST` `http://teamcity:8111/app/rest/userGroups`
Delete a group: `DELETE` `http://teamcity:8111/app/rest/userGroups/key:Group_Key`

Other

Data Backup

Start backup: `POST` `http://teamcity:8111/app/rest/server/backup?includeConfigs=true&includeDatabase=true&includeBuildLogs=true&fileName=`
where <fileName> is the prefix of the file to save backup to. The file will be created in the default backup directory (see more).
Get current backup status (idle/running): `GET` `http://teamcity:8111/app/rest/server/backup`

Typed Parameters Specification

List typed parameters:

- for a project: `http://teamcity:8111/app/rest/projects/<locator>/parameters`
- for a build configuration: `http://teamcity:8111/app/rest/buildTypes/<locator>/parameters`
  The information returned is: `parameters count`, `property name`, `value`, and `type`. The `rawValue` of the `type` element is the parameter specification as defined in the UI.

Get details of a specific parameter:
`GET` to `http://teamcity:8111/app/rest/buildTypes/<locator>/parameters/<name>` . Accepts/returns plain-text, XML, JSON. Supply the relevant `Content-Type` header to the request.

Create a new parameter:
`POST` the same XML or JSON or just plain-text as returned by `GET` to `http://teamcity:8111/app/rest/buildTypes/<locator>/parameters/`. Note that secure parameters, i.e. `type=password`, are listed, but the values not included into response, so the result should be amended before POSTing back.

Since TeamCity 9.1, partial updates of a parameter are possible (currently in an experimental state):

- name: PUT the same XML or JSON as returned by `GET` to `http://teamcity:8111/app/rest/buildTypes/<locator>/parameters/NAME`
- type: GET/PUT accepting XML and JSON as returned by `GET` to the URL `http://teamcity:8111/app/rest/buildTypes/<locator>/parameters/NAME/type`
- type's rawValue: GET/PUT accepting plain text `http://teamcity:8111/app/rest/buildTypes/<locator>/parameters/NAME/type/rawValue`

Build Status Icon

Icon that represents a build status:

An .svg icon (recommended): `GET http://teamcity:8111/app/rest/builds/<buildLocator>/statusIcon.svg`

A .png icon: `GET http://teamcity:8111/app/rest/builds/<buildLocator>/statusIcon`

Icon that represents build status for several builds (since TeamCity 10.0):
`GET` request and "`strob`" build locator dimension:

> Example requests:
>
> For project with id "PROJECT_ID":
> GET
> http://teamcity:8111/app/rest/builds/aggregated/strob:(buildType:(project:(id:PROJECT_ID)))/statusIcon.svg
>
> For all active branches in a build configuration with id "BUILD_CONF_ID":
> GET
> http://teamcity:8111/app/rest/builds/aggregated/strob:(branch:(buildType:(id:BUILD_CONF_ID),policy:active_history_and_active_vcs_branches),locator:(buildType:(id:BUILD_CONF_ID)))/statusIcon.svg

For request /app/rest/builds/aggregated/<build locator> the status is calculated by list of the builds:
app/rest/builds?locator=<build locator>

This allows embedding a build status icon into any HTML page with a simple `img` tag:

> For build configuration with internal id "btXXX":
> Status of the last build: <img
> src="http://teamcity:8111/app/rest/builds/buildType:(id:btXXX)/statusIcon"/>
> Status of the last build tagged with tag "myTag": <img
> src="http://teamcity:8111/app/rest/builds/buildType:(id:btXXX),tag:myTag/statusIcon"/>

All other `<buildLocator>` options are supported.

e.g. you can use the following markdown markup to add the build status for GitHub repository for the build configuration with id "TeamCityPluginsByJetBrains_TeamcityGoogleTagManagerPlugin_Build" and server https://teamcity.jetbrains.com with guest authentication enabled:

> [![Build
> status](https://teamcity.jetbrains.com/guestAuth/app/rest/builds/buildType:(id:TeamCityPluginsByJetBrains_TeamcityGoogleTagManagerPlugin_Build)/statusIcon.svg)](https://teamcity.jetbrains.com/viewType.html?buildTypeId=TeamCityPluginsByJetBrains_TeamcityGoogleTagManagerPlugin_Build)

If the returned image contains "no permission to get data" text (

), ensure that one of the following is true:

- the server has the guest user access enabled and the guest user has permissions to access the build configuration referenced, or
- the build configuration referenced has the "enable status widget" option ON
- you are logged in to the TeamCity server in the same browser and you have permissions to view the build configuration referenced. Note that this will not help for embedded images in GitHub pages as GitHub retrieves the images from the

server-side.

TeamCity Licensing Information Requests

Since TeamCity 10:
Licensing information: `GET http://teamcity:8111/app/rest/server/licensingData`

List of license keys: `GET http://teamcity:8111/app/rest/server/licensingData/licenseKeys`
License key details: `GET http://teamcity:8111/app/rest/server/licensingData/licenseKeys/<license_key>`
Add license key(s): `POST text/plain newline-delimited keys to http://teamcity:8111/app/rest/server/licensingData/licenseKeys`
Delete a license key: `DELETE http://teamcity:8111/app/rest/server/licensingData/licenseKeys/<license_key>`

CCTray

CCTray-compatible XML is available via `http://teamcity:8111/app/rest/cctray/projects.xml`.

Without authentication (only build configurations available for guest user): `http://teamcity:8111/guestAuth/app/rest/cctray/projects.xml` .

The CCTray-format XML does not include paused build configurations by default. The URL accepts "locator" parameter instead with standard build configuration locator.

Request Examples

Request Sending Tool

You can use curl command line tool to interact with the TeamCity REST API.
Example command:

```
curl -v --basic --user USERNAME:PASSWORD --request POST "http://teamcity:8111/app/rest/users/" --data
@data.xml --header "Content-Type: application/xml"
```

Where USERNAME, PASSWORD, "teamcity:8111" are to be substituted with real values and data.xml file contains the data to send to the server.

Creating a new project

Using curl tool

```
curl -v -u USER:PASSWORD http://teamcity:8111/app/rest/projects --header "Content-Type: application/xml"
--data-binary
"<newProjectDescription name='New Project Name' id='newProjectId'><parentProject
locator='id:project1'/></newProjectDescription>"
```

Making user a system administrator

1. Get super user token

3. Issue the request
Get curl command line tool and use a command line:

```
curl -v -u :SUPERUSER_TOKEN --request PUT http://teamcity:8111/app/rest/users/username:USERNAM
E/roles/SYSTEM_ADMIN/g/
```

where
"SUPERUSER_TOKEN" - the super user token unique for each server start
"teamcity:8111" - the TeamCity server URL
"USERNAME" - the username of the user to be made the system administrator

More examples (for TeamCity 8.0) are available in this external posting.