Server-side Object Model

Project model

The main entry point for project model is jetbrains.buildServer.serverSide.ProjectManager. With help of this class you can obtain projects and build configurations, create new projects or remove them.

On the server side projects are represented by jetbrains.buildServer.serverSide.SProject interface. Project has unique id (projectId). Any change in the project will not be persisted automatically. If you need to persist project configuration on disk use SProject.persist() method.

Build configurations are represented by jetbrains.buildServer.serverSide.SBuildType. As with projects any change in the build configuration settings is not saved on disk automatically. Since build configurations are stored in the projects, you should persist corresponding project after the build configuration modification.

Note: interfaces available on the server side only have prefix S in their names, like SProject, SBuildType and so on.

Build lifecycle

When build is triggered it is added to the build queue (jetbrains.buildServer.serverSide.BuildQueue). While staying in the queue and waiting for a free agent it is represented by jetbrains.buildServer.serverSide.SQueuedBuild interface. Builds in the queue can be reordered or removed. To add new build in the queue use addToQueue() method of the jetbrains.buildServer.serverSide.SBuildType.

A separate thread periodically tries to start builds added to the queue on a free agent. A started build is removed from the queue and appears in the model as jetbrains.buildServer.serverSide.SRunningBuild. After the build finishes it becomes jetbrains.buildServer.serverSide.SFinishedBuild and is added to the build history. Both SRunningBuild and SFinishedBuild extend common interface: jetbrains.buildServer.serverSide.SBuild.

There is another entity jetbrains.buildServer.serverSide.BuildPromotion which is associated with build during the whole build lifecycle. This entity contains all information necessary to reproduce this build, i.e. build parameters (properties and environment variables), VCS root revisions, VCS root settings with checkout rules and dependencies. BuildPromotion can be obtained on the any stage: when build is in the queue, running or finished, and it always be the same object.

Accessing builds

A started build (running or finished) can be found by its' id (buildId). For this you should use jetbrains.buildServer.serverSide.SBuildServer#findBuildInstanceById(long) method.

It is also possible to find build in the build history, or to retrieve all builds from the history. Take a look at SBuildType#getHistory() method and at jetbrains.buildServer.serverSide.BuildHistory service.

Note: if not mentioned specifically the returned collections of builds are always sorted by start time in reverse order, i.e. most recent build comes first.

Listening for server events

A lot of events are generated by the server during its lifecycle, these are events like buildStarted, buildFinished, changeAdded and so on. Most of these events are defined in the jetbrains.buildServer.serverSide.BuildServerListener interface. There is corresponding adapter class jetbrains.buildServer.serverSide.BuildServerAdapter which you can extend.

To register your listener you should obtain reference to EventDispatcher<BuildServerListener>. Since this dispatcher is defined as a Spring bean, you can obtain reference with help of Spring autowiring feature.

User model events

You can also watch for events from TeamCity user model. For example, you can track new user accounts registration, removing of the users or changing of the user settings. You should use jetbrains.buildServer.serverSide.UserModelListener interface and register your listeners in the jetbrains.buildServer.users.UserModel.

VCS changes

TeamCity server constantly polls version control systems to determine whether a new change occurred. Polling is done per VCS root (jetbrains.buildServer.vcs.SVcsRoot). Each VCS root has unique id, VCS specific properties, scope (shared or project local) and version. Every change in VCS root creates a new version of the root, but VCS root id remains the same. VCS roots can be obtained from SBuildType or found by id with help of jetbrains.buildServer.vcs.VcsManager.

A change is represented by jetbrains.buildServer.vcs.SVcsModification class. Each detected change has unique id and is associated with concrete version of the VCS root. A change also belongs to one or more build configurations (these are build configurations where VCS root was attached when change was detected), see getRelatedConfigurations() method.

There are several methods allowing to obtain VCS changes:

1. SBuildType#getPendingChanges() - use this method to find pending changes of the some build configuration (i.e. changes which are not yet associated with a build)
2. SBuild#getContainingChanges() - use this method to obtain changes associated with a build, i.e. changes since previous build
3. jetbrains.buildServer.vcs.VcsModificationHistory - use this service to obtain arbitrary changes stored in the changes history, find change by id and so on.

Note: if not mentioned specifically the returned collections of changes are always sorted in reverse order, with the most recent change coming first.

Agents

Agent is represented by jetbrains.buildServer.serverSide.SBuildAgent interface. Agents have unique id and name, and can be found by name or by id with help of jetbrains.buildServer.serverSide.BuildAgentManager. Agent can have various states:

1. registered / unregistered: agent is registered if it is connected to the server.
2. authorized / unauthorized: authorized agent can run builds, unauthorized can't. It is impossible to run build on unauthorized agent even manually. A number of authorized agents depends on entered license keys.
3. enabled / disabled: builds won't run automatically on disabled agents, but it is possible to start build manually on such agent if user has required permission.
4. outdated / up to date: agent is outdated if its' version does not match server version or if some of its' plugins should be updated. New builds will not start on an outdated agent until it upgrades, but already running builds will continue to run as usual.