Build Runner Plugin

A build runner plugin consists of two parts: agent-side and server-side. The server side part of the plugin provides meta information about the build runner, the web UI for the build runner settings and the build runner properties validator. The agent-side part launches builds.

On this page:

A build runner can have various settings which must be edited by the user in the web UI and passed to the agent. These settings are called runner parameters (or runner properties) and provided as a Map<String, String> to the agent part of the runner.

Hint: some build runners whose source code can be used as a reference:

- Rake Runner
- FxCop runner sources
  Other build runner plugins.

Server-side part of the runner

The main entry point for the runner on the server side is jetbrains.buildServer.serverSide.RunType. A build runner plugin must provide its' own RunType and register it in the jetbrains.buildServer.serverSide.RunTypeRegistry.

RunType has a type which must be unique among all build runners and correspond to the type returned by the agent-side part of the runner (see jetbrains.buildServer.agent.AgentBuildRunnerInfo).

The getEditRunnerParamsJspFilePath and getViewRunnerParamsJspFilePath methods return paths to JSP files for editing and viewing runner settings. These JSP files must be bundled with plugin in buildServerResources subfolder, read more. The paths should be relative to the buildServerResources folder.

Since TeamCity 5.1, the path to the build runner resources files should be a full path without context. This path could be either a path to a .jsp file or a path that is handled by a controller. The plugin class may use PluginDescriptor#getPluginResourcesPath() method to create a path to a .jsp file from the buildServerResources folder of the plugin.

TeamCity 5.0.x and earlier uses the following rule to compute a full path to the runner's jsp:

```
<context path>/plugins/<runType>/<returned jsp path>
```

Hint: before writing your own JSP for a custom build runner, take a look at the JSP files of the existing runners bundled with TeamCity.

When a user fills in your runner settings and submits the form, jetbrains.buildServer.serverSide.PropertiesProcessor returned by the getRunnerPropertiesProcessor method will be called. This processor will be able to the verify user settings and indicate which of them are invalid.

Usually a JSP page is simple and does not provide much controls except for fields, checkboxes and so on. But if you need more control on how the page is processed on the server side, then you should register your own extension to the runner editing controller: jetbrains.buildServer.controllers.admin.projects.EditRunTypeControllerExtension.

And finally if you need to prefill some settings with default values, you can do this with the help of the getDefaultRunnerProperties method.

Agent-side part of the runner

The main interface for agent-side runners is jetbrains.buildServer.agent.AgentBuildRunner. However, if your custom runner runs an external process, it is simpler to use the following classes:

1. jetbrains.buildServer.agent.runner.CommandLineBuildServiceFactory
2. jetbrains.buildServer.agent.runner.CommandLineBuildService
3. jetbrains.buildServer.agent.runner.BuildServiceAdapter

You should implement the CommandLineBuildServiceFactory factory interface and make your class a Spring bean. The factory also provides some meta information about the runner via jetbrains.buildServer.agent.AgentBuildRunnerInfo.

CommandLineBuildService is an abstract class which simplifies external processes launching and allows listening for process events (output, finish and so on). Your runner should extend this class. Snce TeamCity 6.0, we introduced the jetbrains.buildServer.agent.runner.BuildServiceAdapter class that extends CommandLineBuildService and provides utility methods to access build and runner context parameters.

AgentBuildRunnerInfo has two methods: getType which must return the same type as the one returned by the server-side part of the plugin, and canRun which is called to determine whether the custom runner can run on the agent (in the agent environment).

If the command line build service is not suitable for your needs, you can still implement the AgentBuildRunner interface and define it in the Spring context. Then it will be loaded automatically.

Extending the Ant runner

The TeamCity Ant runner, while being a plugin itself, can also be extended with the help of jetbrains.buildServer.agent.ant.AntTaskExtension. This extension works in the same JVM where Ant is running. Using this extension, you can watch for Ant tasks, modify/patch them and log various messages to the build log.

Your class implementing AntTaskExtension interface must be defined in the Spring bean and it will be picked up by the Ant runner automatically. You need to add a dependency to <teamcity>/webapps/ROOT/WEB-INF/plugins/ant/agent/antPlugin.zip!antPlugin/ant-runtime.jar jar.

Your Build Runner Results in TeamCity

Build log

Usually a build runner starts an external process, and logging is performed from that process. The simplest way to log messages in this case is to use service messages, read more. In brief, a service message is a specially formatted text with attributes; when such text is logged to the process output, it is parsed and the associated processing is performed. With the help of these messages you can create a TeamCity hierarchical build log, report tests, errors and so on.

If an external process launched by your runner is Java and you can't use service messages, it is possible to obtain jetbrains.buildServer.agent.BuildProgressLogger in the class running in this JVM. For this, the following jar files must be added in the classpath of the external Java process: runtime-util.jar, server-logging.jar. Then you should use the jetbrains.buildServer.agent.LoggerFactory method to construct the logger: LoggerFactory.createBuildProgressLogger(parentClassloader). Since this way is more involved, it is recommended to use service messages instead.

If logging of the messages is done in the agent JVM (not from within the external process started by your runner), you can obtain jetbrains.buildServer.agent.BuildProgressLogger from the jetbrains.buildServer.agent.AgentRunningBuild#getBuildLogger method.

Artifacts

You can instruct your build runner to publish the resulting artifacts to TeamCity using service messages. Note that artifacts are uploaded to the TeamCity server in the background, so to verify that your artifacts are uploaded, you'll have to wait until your build is finished.

Reports

XML Report processing

If your runner reports build results in a format supported by TeamCity, they can be displayed in the TeamCity web UI on the Build Results page. There are two ways to approach this:
- using the XML Report Processing build feature
- via service messages

HTML Report processing

If your build runner produces some static HTML content, it can be displayed in the TeamCity web UI. Configure a custom report tab to show the results on a project or build level.