Getting Started with Plugin Development

The use of plugins allows you to extend the TeamCity functionality. See the list of existing TeamCity plugins created by JetBrains developers and community.

This document provides information on how to develop and publish a server-side plugin for TeamCity using Maven. The plugin will return the "Hello World" jsp page when using a specific URL to the TeamCity Web UI.

On this page:

Introduction

A plugin in TeamCity is a `zip` archive containing a number of classes packed into a JAR file and plugin descriptor file.
The TeamCity Open API can be found in the JetBrains Maven repository. The Javadoc reference for the API is available online a nd locally in <TeamCity Home Directory>/devPackage/javadoc/openApi-help.jar, after you install TeamCity.

Step 1. Set up the environment

To get started writing a plugin for TeamCity, set up the plugin development environment.

1. Download and install Oracle Java. Set the Java_Home environment variable on your system. The 32-bit Java 1.7 and sin ce Teamcity 9.1 Java 1.8 is recommended, the 64-bit version can be used.
2. Download and install TeamCity on your development machine. Since you are going to use this machine to test your plugin, it is recommended that this TeamCity server is of the same version as your production server. We are using TeamCity 9.0.2 installed on Windows in our setup.
3. Download and install a Java IDE; we are using Intellij IDEA 14.0.3 Community Edition, which has a built-in Maven integration.
4. Download and install Apache Maven. Maven 3.2.x is recommended. Set the M2_HOME environment variable. Run `mvn -version` to verify your setup. We are using Maven 3.2.5. in our setup.

Step 2. Generate a Maven project

We'll generate a Maven project from an archetype residing in JetBrains Maven repository. Executing the following command will produce a project for a server-side-only plugin.

```
mvn archetype:generate -DarchetypeRepository=http://download.jetbrains.com/teamcity-repository
-DarchetypeArtifactId=teamcity-server-plugin -DarchetypeGroupId=org.jetbrains.teamcity.archetypes
-DarchetypeVersion=RELEASE
```

You will be asked to enter the Maven `groudId`, `artifactId`, `version` and `package name` for your plugin.

We used the following values:

`groudId`	`com.demoDomain.teamcity.demoPlugin`
`artifactId`	`demoPlugin`
`version`	leave the default `1.0-SNAPSHOT`

packaging	leave the default package name

demoPlugin will be used as the internal name of our plugin.

When the build finishes, you'll see that the demoPlugin directory was created in the directory where Maven was called.

View the project structure

The root of the demoPlugin directory contains the following:

- the readme.txt file with minimal instructions to develop a server-side plugin
- the pom.xml file which is your Project Object Model
- the teamcity-plugin.xml file which is your plugin descriptor containing meta information about the plugin.
- the demoPlugin-server directory contains the plugin sources:
  - \src\main\java\zip contains the AppServer.java file
  - src\main\resources includes resources controlling the plugin look and feel.
  - src\main\resources\META-INF folder contains build-server-plugin-demo-plugin.xml, the bean definition file for our plugin. TeamCity plugins are initialized in their own Spring containers and every plugin needs a Spring bean definition file describing the main services of the plugin.
- the build directory contains the xml files which define how the project output is aggregated into a single distributable archive.

Step 3. Edit the plugin descriptor

Open the teamcity-plugin.xml file in the project root folder with Intellij IDEA and add details, such as the plugin display name, description, vendor, and etc. by modifying the corresponding attributes in the file.

Step 4. Create the plugin sources

Open the pom.xml from the project root folder with Intellij IDEA.

We are going to make a controller class which will return Hello.jsp via a specific TeamCity URL.

A. Create the plugin web-resources

The plugin web resources (files that are accessed via hyperlinks and JSP pages) are to be placed into the buildServerResources subfolder of the plugin's resources.

1. First we'll create the directory for our jsp: go to the demoPlugin-server\src\main\resources directory in IDEA and create the buildServerResources directory.
2. In the newly created demoPlugin-server\src\main\resources\buildServerResources directory, create the Hello.jsp file, e.g.

```
<html>
<body>
Hello world
</body>
</html>
```

B. Create the controller and obtain the path to the JSP

Go to \demoPlugin\demoPlugin-server\src\main\java\com\demoDomain\teamcity\demoPlugin and open the AppServer.java file to create a custom controller:

1. We'll create a simple controller which extends the TeamCity jetbrains.buildServer.controllers.BaseController class and implements the BaseController.doHandle(HttpServletRequest, HttpServletResponse) method.
2. The TeamCity open API provides the jetbrains.buildServer.web.openapi.WebControllerManager which allows registering custom controllers using the path to them: the path is a part of URL starting with a slash / appended to the URL of the server root.
3. Next we need to construct the path to our JSP file. When a plugin is unpacked on the TeamCity server, the paths to its

resources change. To obtain valid paths to the files after the plugin is installed, use the jetbrains.buildServer.web.openapi.PluginDescriptor class which implements the `getPluginResourcesPath` method; otherwise TeamCity might have difficulties finding the plugin resources.

```
package com.demoDomain.teamcity.demoPlugin;

import jetbrains.buildServer.controllers.BaseController;
import jetbrains.buildServer.web.openapi.PluginDescriptor;
import jetbrains.buildServer.web.openapi.WebControllerManager;
import org.jetbrains.annotations.Nullable;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class AppServer extends BaseController {
    private PluginDescriptor myDescriptor;

    public AppServer (WebControllerManager manager, PluginDescriptor descriptor) {
        manager.registerController("/demoPlugin.html",this);
        myDescriptor=descriptor;
    }

    @Nullable
    @Override
    protected ModelAndView doHandle(HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse) throws Exception {
        return new ModelAndView(myDescriptor.getPluginResourcesPath("Hello.jsp"));
    }
}
```

C. Update the Spring bean definition

Go to the `demoPlugin-server\src\main\resources\META-INF` directory and update `build-server-plugin-demo-plugin.xml` to include our AppServer class.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans default-autowire="constructor">
    <bean class="com.demoDomain.teamcity.demoPlugin.AppServer"></bean>
</beans>
```

Step 5. Build your project with Maven

Go to the root directory of your project and run

```
mvn package
```

The `target` directory of the project root will contain the `<demoPlugin>.zip` file. It is our plugin package, ready to be installed.

Step 6. Install the plugin to TeamCity

1. Copy the plugin zip to <TeamCity Data Directory>/plugins directory.
2. Restart the server and locate the TeamCity Demo Plugin in the Administration|Plugins List to verify the plugin was installed correctly.

The Hello World page is available via `<TeamCity server URL>/demoPlugin.html`.

Next Steps

Read more if you want to extend the TeamCity pages with custom elements.
The detailed information on TeamCity plugin development is available here.
You may also use the plugin allowing you to control a TeamCity instance from the command line and to install a new/updated plugin created from a Maven archetype.