SIEMENS

# Automated Test Execution with Polarion

# Contents

# Chapter 1: Run external automated tests from Polarion

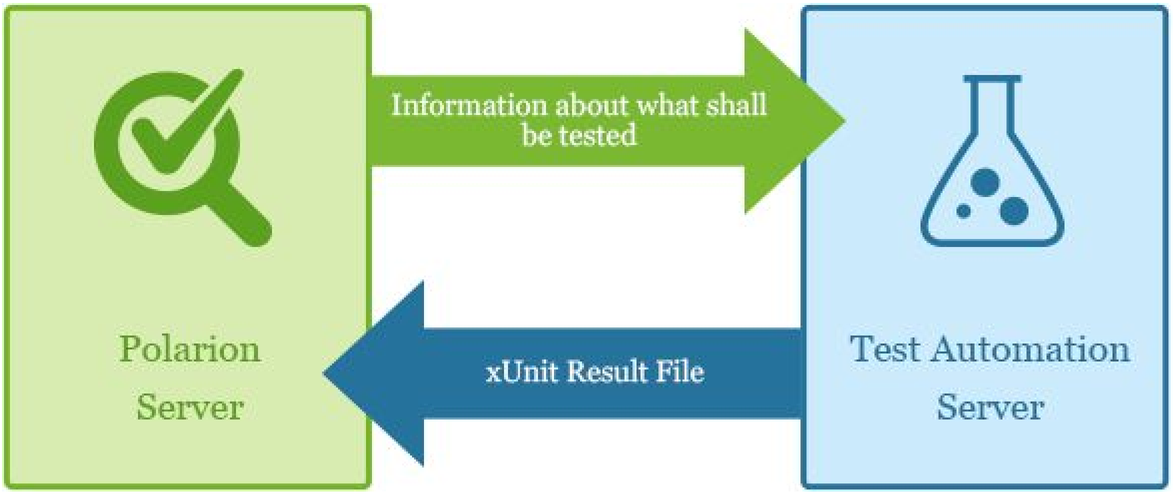
ALM|QA

This document outlines how to set up an environment to launch test automation scripts directly from Polarion, and automatically import all external test results for use by the relevant reports and metrics. You may opt to run test automation scripts explicitly, or schedule a job that runs them at a future time, and/or periodically.

When using Polarion for such a scenario, an external automated testing tool must be available and accessible to your network, and its role must be clearly defined. The role of the external automated testing tool is to receive a set of data representing what will be tested, and how it will be tested. Polarion typically holds this information and its role is to deliver it to the external tool in its environment.

## Basic setup concept

The first thing to consider is the basic configuration between Polarion and a Test Automation Server (TAS), including how to invoke a build tool to enable the exchange of test data with automated tests, using any of a variety of commercial and open-source test automation engines.



The above graphic shows the basic concept of data exchange between a Polarion instance and a TAS. In most cases, these servers are separate machines. It is important that the TAS contains some kind of Test Agent that receives incoming requests from the initiating tool — that is, Polarion. This is important because the running of tests often invokes complex third-party tools, which must be available in the testing environment. The Test Agent is what needs to be called from Polarion, and provided with all the information required to have the external testing tool perform tests.

## Invoke a build

To start the process of sharing test information between Polarion and a TAS, it is necessary to invoke a build process using one of the following tools:

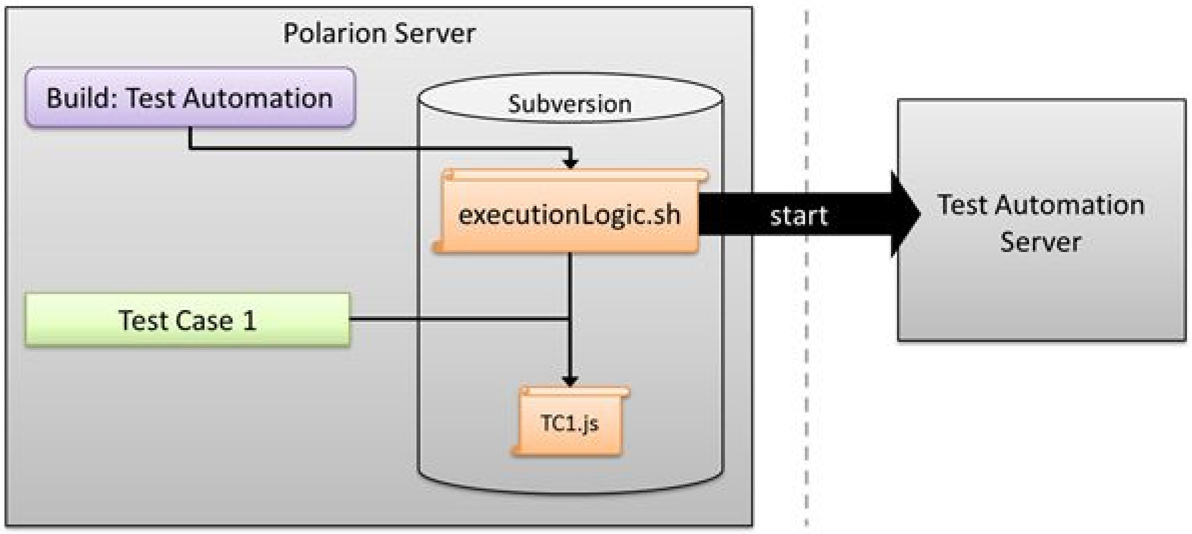
- Jenkins Integration Server
- Hudson
- ANT
- Maven
- Polarion builds

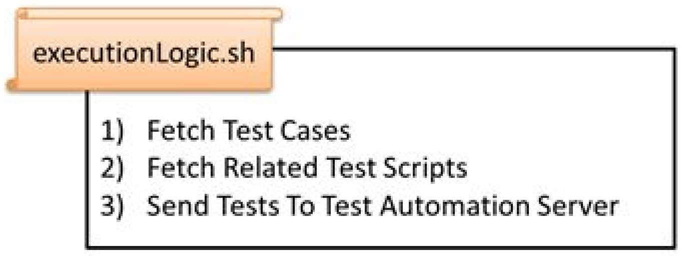
### Exchange test data

Once you have invoked the appropriate build tool, you will need to create and run the appropriate scripts for the following operations:

1. Fetch test cases from Polarion for what is to be tested.
2. Fetch related test scripts that must be run.
3. Send tests to the TAS.

### Trigger testing

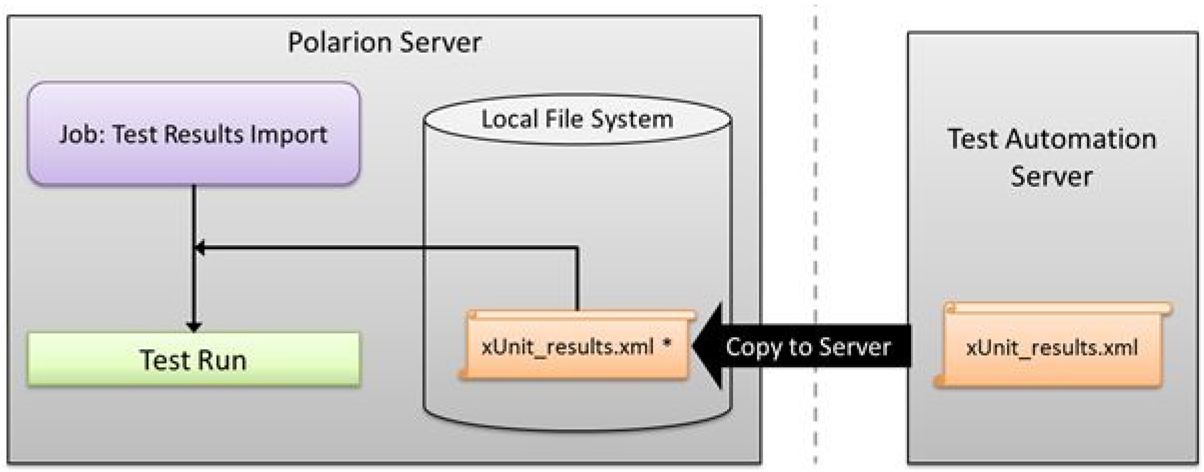



executionLogic.sh
1) Fetch Test Cases
2) Fetch Related Test Scripts
3) Send Tests To Test Automation Server

The running or performing of the actual tests is the responsibility of the TAS. After Polarion has supplied the test data and started the testing process, it simply waits for results to be returned. The results themselves are generated by the TAS, and must be copied to a local or network location accessible to Polarion once testing is complete.

## Receive test results

Once automated tests are completed, the results must somehow be transferred back to Polarion, and the relevant test case Work Items must be updated.



The diagram above shows the required architecture for the reimporting of test results. You can see that the file **xUnit_results.xml** is originally created on the TAS in a valid xUnit format. The TAS has finished submitting the results to a location Polarion can access. This may be a network share or just a folder on the Polarion server's file system. There is a separate job, which is commonly run every few minutes or hours. The job checks if new XML results are available in the specified location. The job creates a new Test Run for every valid XML file found in the specified location. Finally, the job deletes the XML test result files.

### Example using Jenkins

The information below outlines a process for integrating Polarion with Jenkins to fetch test cases, and automatically import test results generated by Jenkins.

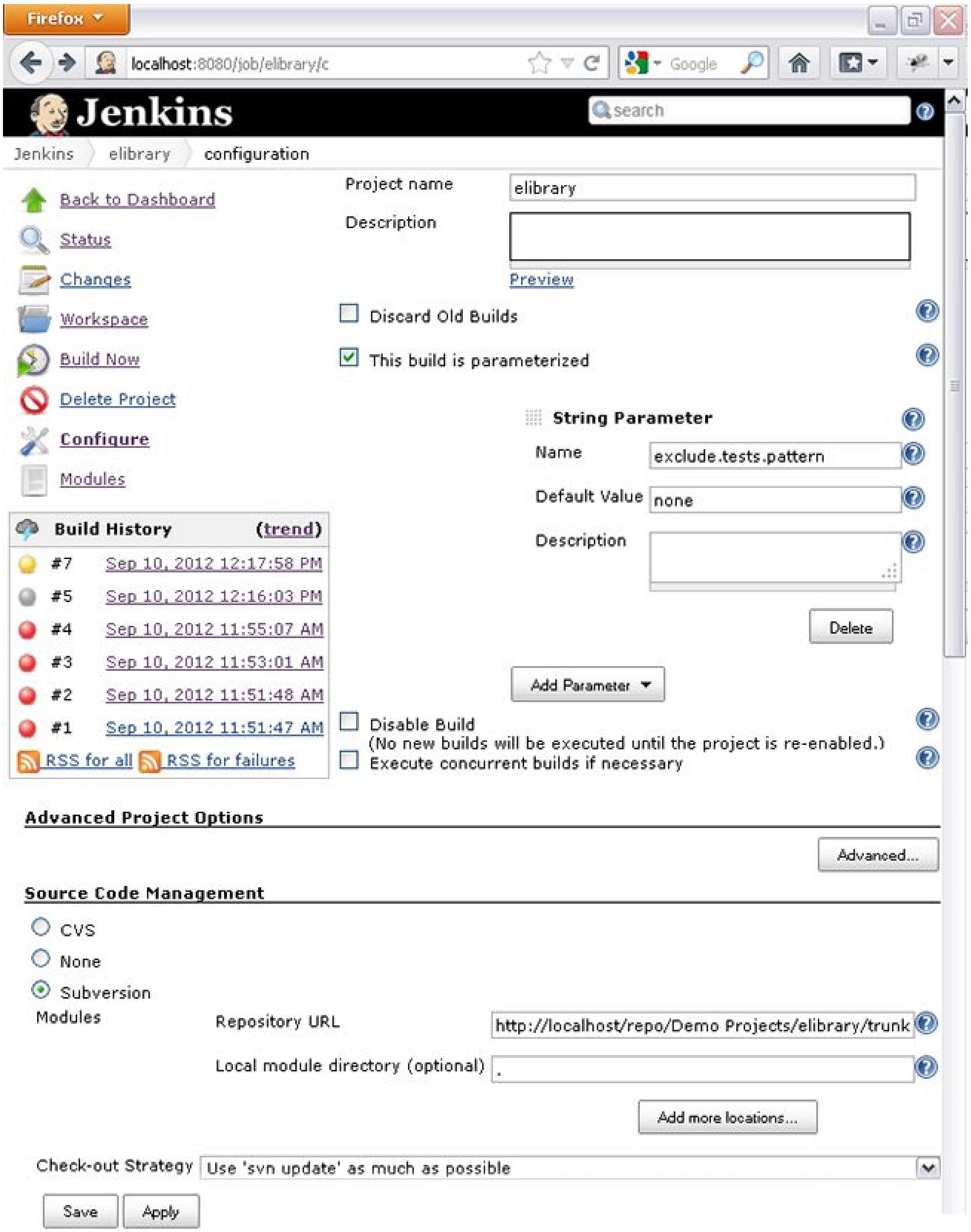
**Prerequisites:**

- Jenkins must be set up, running, and accessible online. Example URL: **http://myjenkins**
- Polarion must be set up, an ALM or QA license must be installed and available, and the server must be accessible online. Example URL: **http://mypolarion**
- A project must exist on the Polarion server that contains a set of Work Items representing test cases to be tested by the automated tests run on the Jenkins server.

This example is based on the **E-Library** demo project that ships with Polarion. If your administrator opted not to install demo projects on your instance, you might consider obtaining and installing an evaluation copy of Polarion. Demo projects are also accessible on the Polarion ALM Test Drive server, but you cannot modify anything there. This example will demonstrate how to use a Jenkins project to host the build automation procedure, build the **E-Library** project, and import the test results generated by Jenkins into Polarion.

The **E-Library** build process is parametrized by the `exclude-test-pattern` variable. Typically you will not have this situation, but in the case of E-Library you need to pass that variable.

The following figure shows the setup window in Jenkins. Note the Subversion repository URL pointing to the project URL in Polarion.



Project name: elibrary

Description:

☐ Discard Old Builds

☑ This build is parameterized

**String Parameter**

Name: exclude.tests.pattern

Default Value: none

Description:

☐ Disable Build
(No new builds will be executed until the project is re-enabled.)

☐ Execute concurrent builds if necessary

**Build History** (trend)

| | |
|---|---|
| #7 | Sep 10, 2012 12:17:58 PM |
| #5 | Sep 10, 2012 12:16:03 PM |
| #4 | Sep 10, 2012 11:55:07 AM |
| #3 | Sep 10, 2012 11:53:01 AM |
| #2 | Sep 10, 2012 11:51:48 AM |
| #1 | Sep 10, 2012 11:51:47 AM |

**Advanced Project Options**

**Source Code Management**

○ CVS
○ None
◉ Subversion

Modules

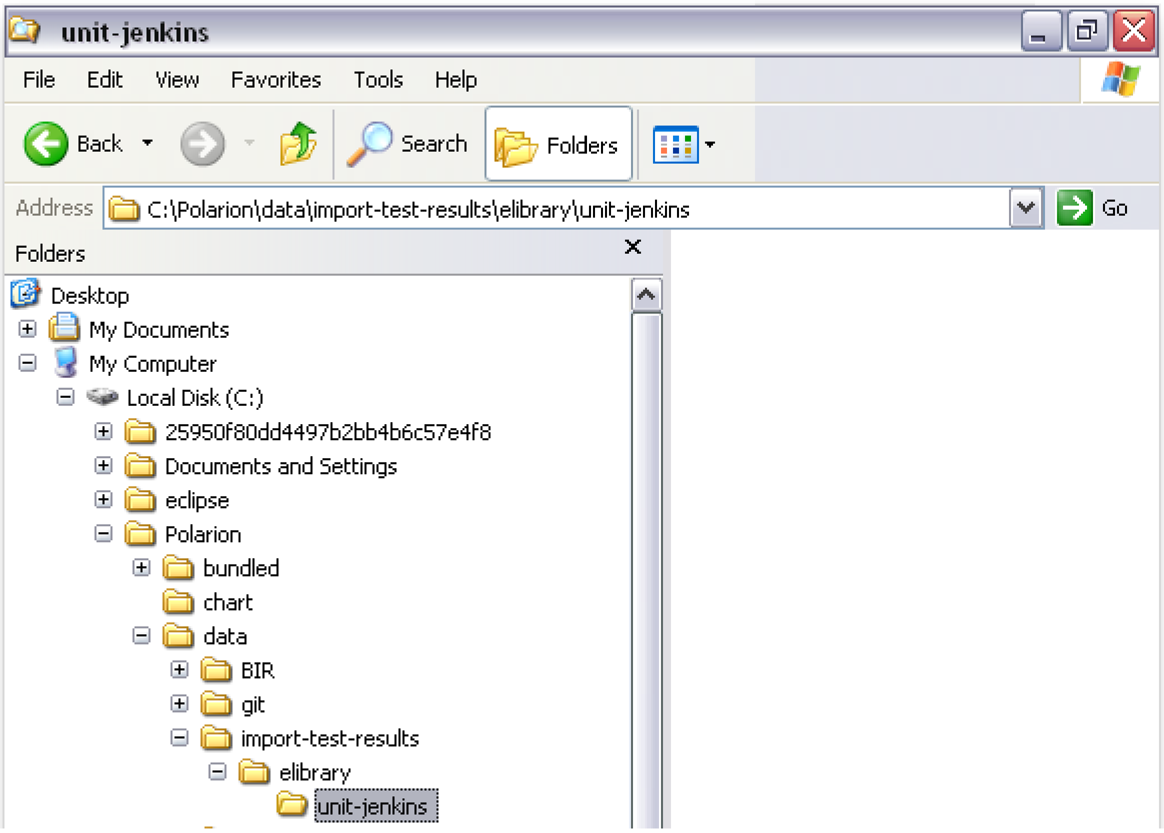
Repository URL: http://localhost/repo/Demo Projects/elibrary/trunk

Local module directory (optional): .

Check-out Strategy: Use 'svn update' as much as possible

Now it is necessary to set up the configuration for importing test results to Polarion,

1. Create a folder on the file system of the Polarion server to store imported test results. Example: **c:/polarion/data/import-test-results/elibrary/jenkins-unit**



   This folder must be shared so that it can be accessed from the Jenkins server.

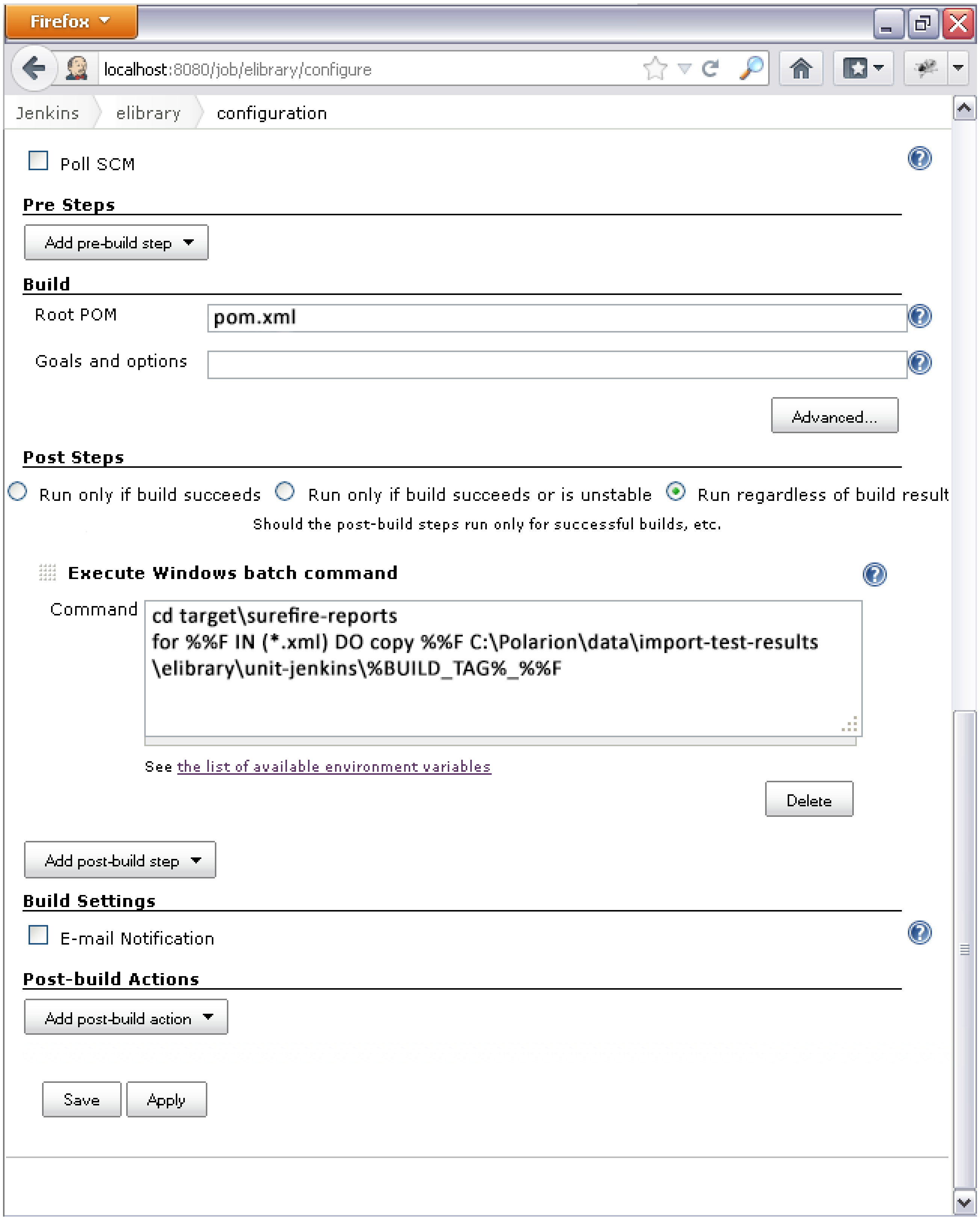
2. Create a new Polarion job to watch the shared folder and import test results when they appear in the folder. See the following code example.

```
<job id=”xUnitFileImport” name=”Import Elibrary Tests Results” scope=”system”>
    <path>C:\Polarion\data\import-test-results\elibrary\unit-jenkins</path>
    <project>elibrary</project>
    <userAccountVaultKey>xUnitFileImportUser</userAccountVaultKey>
    <maxCreatedDefects>10</maxCreatedDefects>
    <maxCreatedDefectsPercent>5</maxCreatedDefectsPercent>
    <templateTestRunId>JUnit Build Test</templateTestRunId>
    <idRegex>(.*).xml</idRegex>
    <groupIdRegex>(.*)_.*.xml</groupIdRegex>
</job>
```

   For the purposes of this example, Polarion and Jenkins are running on the same server. In a production setup, you would use different network drives.

3. Configure a Jenkins post-action to copy test results to the test results folder on the Polarion system, created in Step 1. In this example, a Windows command is used to copy the results. You can also use an ANT-post build action, and, for example, merge the test results into one file using http://ant.apache.org/manual/Tasks/junitreport.html.

```
cd target\surefire-reports
for %%F IN (*.xml) DO copy %%F C:\Polarion\data\import-test-results\elibrary\unitj
```



Poll SCM ☐

**Pre Steps**

**Build**

| | |
|---|---|
| Root POM | pom.xml |
| Goals and options | |

**Post Steps**

Run only if build succeeds ○ Run only if build succeeds or is unstable [✓ Run regardless of build result]

Should the post-build steps run only for successful builds, etc.

**Execute Windows batch command**

Command

```
cd target\surefire-reports
for %%F IN (*.xml) DO copy %%F C:\Polarion\data\import-test-results
\elibrary\unit-jenkins\%BUILD_TAG%_%%F
```

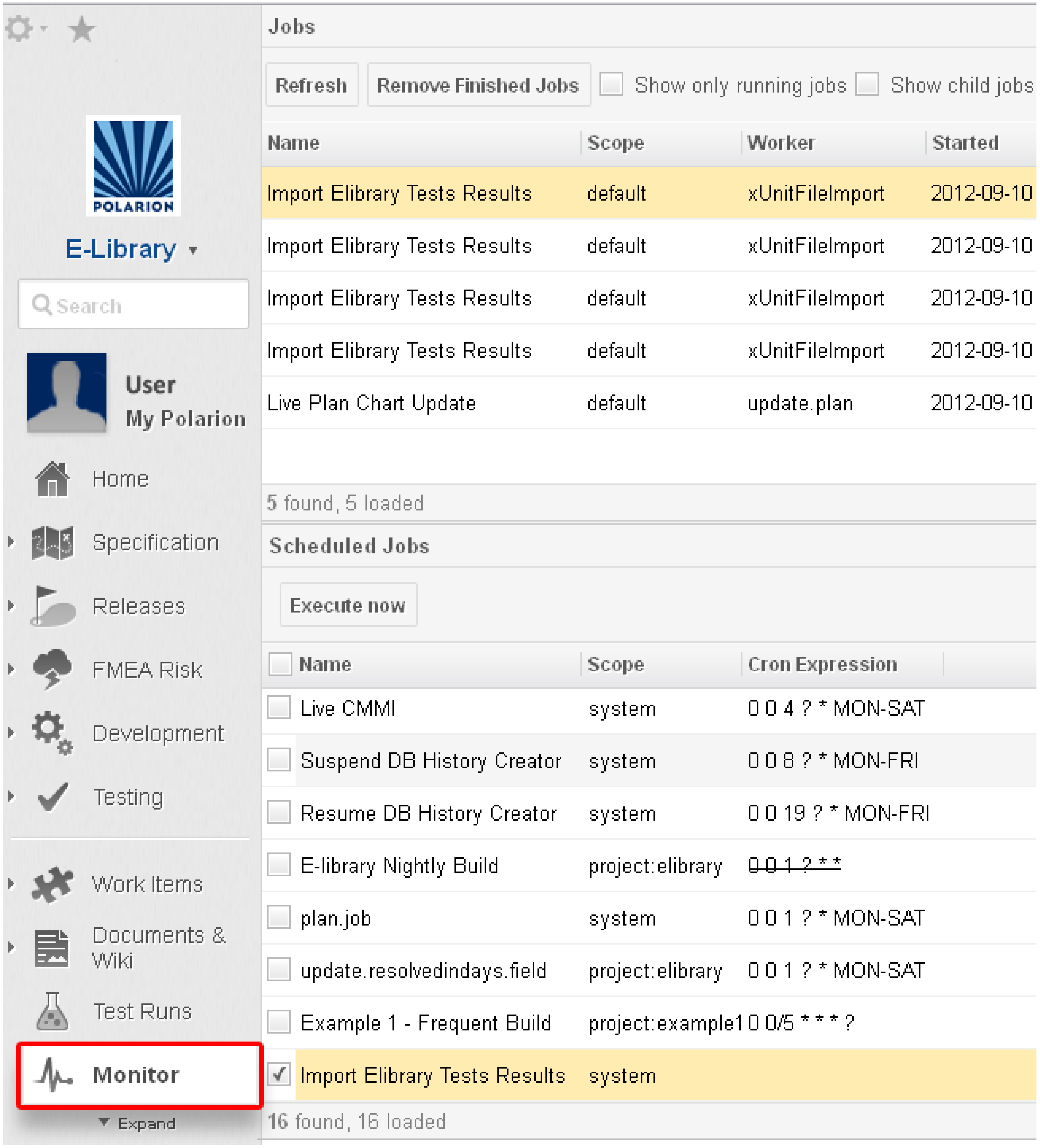
See the list of available environment variables

**Build Settings**

E-mail Notification ☐

**Post-build Actions**

4. The final step in the process is to launch the job that runs the Jenkins build. Assuming you have the necessary permissions, you can launch the job manually in the **Monitor** topic of Navigation. Alternatively, you can wait for the Scheduler to run it according to the job described in **Receive test results**, above.



| Name | Scope | Worker | Started |
|---|---|---|---|
| Import Elibrary Tests Results | default | xUnitFileImport | 2012-09-10 |
| Import Elibrary Tests Results | default | xUnitFileImport | 2012-09-10 |
| Import Elibrary Tests Results | default | xUnitFileImport | 2012-09-10 |
| Import Elibrary Tests Results | default | xUnitFileImport | 2012-09-10 |
| Live Plan Chart Update | default | update.plan | 2012-09-10 |

5 found, 5 loaded

| | Name | Scope | Cron Expression |
|---|---|---|---|
| ☐ | Live CMMI | system | 0 0 4 ? * MON-SAT |
| ☐ | Suspend DB History Creator | system | 0 0 8 ? * MON-FRI |
| ☐ | Resume DB History Creator | system | 0 0 19 ? * MON-FRI |
| ☐ | E-library Nightly Build | project:elibrary | ~~0 0 1 ? * *~~ |
| ☐ | plan.job | system | 0 0 1 ? * MON-SAT |
| ☐ | update.resolvedindays.field | project:elibrary | 0 0 1 ? * MON-SAT |
| ☐ | Example 1 - Frequent Build | project:example1 | 0 0/5 * * * ? |
| ✓ | Import Elibrary Tests Results | system | |

16 found, 16 loaded

For more information on jobs and scheduling, see the **Monitor** and **Scheduler** topics in Polarion's Help.