BLACKDUCK

# Installing, Configuring, and Using the Protex Plugin for Jenkins - CI

Version 1.4.1

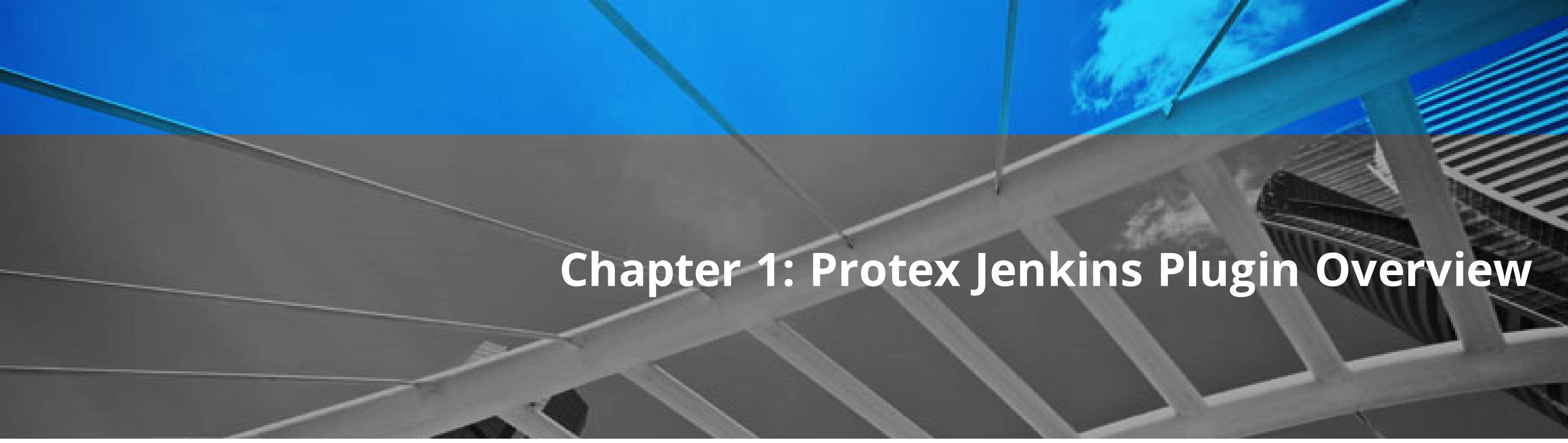
This edition of the *Installing, Configuring, and Using the Protex Plugin for Jenkins - CI* refers to version 1.4.1 of the Protex plugin.

This document created or updated on Tuesday, May 31, 2016.

**Please send your comments and suggestions to:**

Black Duck Software, Incorporated
800 District Avenue
Suite 221
Burlington, MA 01803 USA.

# Contents



# Chapter 1: Protex Jenkins Plugin Overview

Black Duck® Protex™ helps you reduce business risks and complete software projects on-time and on-budget by automatically analyzing software contents, providing a bill of materials (BOM), uncovering potential risks early in the development cycle, and identifying due diligence-related concerns well in advance of an audit. The goal of using Protex is to ensure that your software does not contain licensing conflicts.

Jenkins is an open source continuous integration tool that monitors executions of repeated jobs, such as building a software project or cron jobs. Jenkins focuses on building and testing software projects continuously, and monitoring execution of externally run jobs.

Protex integrates with existing development tools to automatically scan, discover, and identify software origins, an integral step in the development process and essential for enforcing license compliance and corporate policy requirements.

Using the Protex Jenkins Plugin lets you use Jenkins to automatically create and scan Protex projects from your Jenkins projects. The Protex Jenkins plugin lets you integrate Protex and Jenkins to:

- Create a new project in Protex, based on a Jenkins project or a clone of a Jenkins project.
- Run a Protex scan of the Jenkins project and import the data to Protex.
- Pass or fail the Jenkins build based on the results of the Protex analysis:
    - Fail the build if there are new code discoveries (files pending identification).
    - Fail the build if there are license violations.

## 1.1 Protex Jenkins Plugin Installation Overview

The installation process for the Protex Jenkins plugin consists of the following:

1. Downloading and installing the plugin on your Jenkins server.
2. Configuring global configuration options in Jenkins.
3. Configuring project-level options by adding a post-build step to your Jenkins jobs.
4. Running jobs in Jenkins.

## 1.1.1 Protex Jenkins Plugin Requirements

### Software Requirements

An operating system supported by Black Duck Protex:

- Red Hat Enterprise Linux 6.x or 7.0
- Oracle Linux 6.4 or 7.0
- Novel SuSE Linux Enterprise Server 10 or 11
- Microsoft Windows Server 2008 R2 or Server 2012 R2

**Note:** Other platforms supported by Jenkins should work, but are not supported by Black Duck.

The Protex Jenkins plugin requires the following software:

- Protex version 7.0 or greater
- Jenkins 1.509.4 or greater

**Note:** Black Duck recommends the latest Long-Term Support (LTS) release.

- Java SE 7
- With Protex 7.x, you must install the Java Runtime Environment (JRE) version 1.7.0_40 or later.

**Note:** The 32-bit Java Virtual Machine (JVM) is not supported in the Protex Jenkins plugin.

- Microsoft Build version 11 or 12. Note that Visual Studio 2012 installs Microsoft Build 11, and Visual Studio 2013 installs Microsoft Build 12.
- Microsoft .NET Framework version 4.5

### Network Requirements

The Protex Jenkins plugin requires internet connectivity. The machine that hosts your Jenkins server must be able to connect to the Protex server.

### Additional Requirements

Before you install the Protex Jenkins plugin, ensure that you have the following information:

- You know the host name and port for the Protex server.
- You have a user account on the Protex system that you can use for the integration. This account must have the ability to create, clone, and scan a project, so either the **Manager** or **Project Leader** role.
- You have connectivity to the internet.

## 1.1.2 Downloading and Installing the Protex Jenkins Plugin

You download the Protex Jenkins plugin from the Black Duck Customer Hub, then install the plugin using Jenkins.

✱ **To download the Protex Jenkins plugin:**

1. Launch a browser.
2. Navigate to the Black Duck Customer Hub:

   `https://customerhubblackducksoftware.force.com/login`
3. In the menu bar, click **The Exchange**.
4. On the **Exchange** page, click the link for the Protex Jenkins plugin.
5. On the **Protex Jenkins** plugin page, click the **Download** tab.
6. Click the download URL.
7. On the download page, click the download link and save the file to a temporary location.
8. If the download is a zip file, extract the `.hpi` file.

✱ **To install the Protex Jenkins plugin:**

1. Log in to Jenkins as admin.
2. Go to **Manage Jenkins** > **Manage Plugins**.
3. Click the **Advanced** tab.
4. In the **Upload Plugin** section, click **Browse** and navigate to the location of the plugin `.hpi` file and select it.
5. Click **Upload**.
6. Restart the Jenkins instance.
7. After the server has restarted, go to **Manage Jenkins** > **Manage Plugins** > **Installed** and verify the version of the Black Duck Protex Jenkins plugin.

## 1.2 Configuring the Protex Jenkins Plugin - Global Configuration

✱ **To configure the Jenkins plugin:**

1. Log in to Jenkins as admin.
2. Go to **Manage Jenkins > Configure System**.
3. Under the heading **Black Duck Protex**, click **Add Server**.
   a. Enter the **Server Name** of the Protex server.
   b. Enter the **Server URL** of the Protex server, including the port number; for example, `http://protex.blackducksoftware.com:8080`.
   c. To provide default credentials to use when connecting to this server, click **Advanced**.
   d. Next to the **Credentials** field, click **Add**.

1. From the **Kind** menu, select the type of authentication you want to use.
   - User name with password (default).
     a. Enter the **Scope**: global or system.
     b. Enter the user name used to connect to the Protex server; for example, `user@bds.com`. This user must have either the manager or project leader role to be able to create, clone, and analyze projects.
     c. Enter the **Password**.
     d. Enter a **Description**.
   - SSH user name with private key.
     a. Enter the **Scope**: global or system.
     b. Enter the user name.
     c. Enter a **Description**.
     d. Enter the private key directly, or you can link to a file on the Jenkins server.
     e. To enter a passphrase, click **Advanced**.
   - Certificate - Protex uses the PKCS12 keystore format.
     a. Enter the **Scope**: global or system.
     b. Enter the certificate; you can specify the path to a file on the Jenkins master, or upload the certificate.
     c. Enter the password.
     d. Enter a **Description**.
2. Click **Test Connection** to confirm that Jenkins can connect to the Protex server. If you cannot connect, verify or change the credentials that you entered and test the connection again.
3. Specify the **SDK Timeout** in seconds. The default is 300 seconds (5 minutes).

4. Click **Save**.

## 1.2.1 Configuring the Protex Jenkins Plugin - Project Configuration

**Tip:** If you are upgrading from the Protex Jenkins plugin version 0.x to the Project Jenkins plugin version 1.2.0 and higher, see Migrating From Plugin Version 0.x on page 24 for information about how to migrate your existing Jenkins job configurations.

**✱ To configure a project for the Protex Jenkins plugin:**

1. In Jenkins, select a project.
2. In the left sidebar, select **Configure**.
3. In the section **Add post-build action**, select **Protex - Create and Scan Actions**.

4. Complete the required fields:

**Tip:** Click the question mark next to each field for help.

- **Server** (required) - Select a Protex server from the menu.
- **Credentials** (required) - Select the credentials to use to connect to the Protex server.

  To add new credentials, next to the **Credentials** field, click **Add**.

  a. From the **Kind** menu, select the authentication type.

     - User name with password - default.

       a. Enter the **Scope**: global or system.
       b. Enter the **User name**.
       c. Enter the **Password**.
       d. Enter a **Description**.

     - SSH User name with private key.

       a. Enter the **Scope**: global or system.
       b. Enter the **User name**.
       c. Enter a **Description**.
       d. Enter the **Private key**; either directly, or you can link to a file on the Jenkins server.
       e. To enter a Passphrase, click **Advanced**.

     - Certificate - Protex uses the PKCS12 keystore format.

       a. Enter the**Scope**: global or system.
       b. Enter the **Certificate**; you can either specify the path to a file on the Jenkins master or upload the certificate.
       c. Enter the **Password**.
       d. Enter a **Description**.

  b. Click **Add**.

- **Project Name** (required) - Enter the name of the Protex project. If the name provided already exists in Protex, the plugin will not create a new project. If the name provided does not match an existing Protex project, then when you build the job, the plugin will create the Protex project. You can use Jenkins environment variables in this field:
  - **$BUILD_NUMBER** - Creates a Protex project using the build number as the project name.
  - **$BUILD_TAG** - Creates a Protex project using the built tag as the project name.
  - **$JOB_NAME** - Creates a Protex project using the job name as the project name.

For example,

MyPrefix_${JOB_NAME}_MySuffix

OR

$JOB_NAME

**Note:** Note that Protex limits project names to 250 characters in length.

- **Template Project Name** (optional) - Select an existing Protex project to use as a template. The plugin uses the analysis configuration settings from this project for the settings of the new Protex project. If you do not specify a template, the plugin creates the project using the default settings. You can use Jenkins environment variables in this field.
- **Protex Scan Memory (in GBs)** (required) - The default value is 2 GB, which is the minimum required memory. You can specify additional memory, but do not enter a value larger than your total system memory. This field also accepts percentages expressed as decimals; for example, you could specify 4.5 GBs of memory.
- **Source Path** (optional) - Path to the project code. If you do not specify a path, the plugin uses the same workspace folder as the Protex Project Source Path. You can use `src/main/java` or any target within the workspace. You can use Jenkins environment variables in this field.
- **Protex Report Template/Advanced** (optional): Enter the name of a Protex report template. At the end of the build, the report for the build is generated. It is available on the **Protex Report** tab.
- Click **Advanced**:
  - **Protex Report Template** (optional): This is the name of the Protex report template used to generate a Protex report after the scan. If this field is left blank, the Protex report is not generated. The Protex report template must already exist in Protex.

5. Click **Save**.
6. If you want Jenkins to fail the build based on the Protex analysis, add a second post-build action to the configuration. Select the **Protex - Failure Conditions** option from the menu.
   - **Fail build if code has pending identifications?** check box (optional) - Select to fail the build if the Protex analysis finds files that have not been identified.

   **Note:** Note that the **Fail build if code has pending identifications** setting is not recommended for the first build of a project, but is intended to capture changes to an already identified project.

   - **Fail build if code has license violations?** check box (optional) - Select to fail the build if the Protex analysis finds files that violate the project license.
7. Click **Save**.

## 1.3 Using the Protex Jenkins Plugin

✱ **To view the results of the build in Jenkins:**

1. In Jenkins, navigate to the **Project** page.
2. Select the **Build Now** link.
3. Once the build completes, under **Build History**, click the **Build** link.
4. To view the output of the build, select the **Console Output** link.
5. If the build was successful, the console output shows the following at the Protex build step:

   The Protex build step starts with the following text:

   ```
   Initializing - Protex Jenkins Plugin - <version>
   ```

   A successful build step ends with the following text:

   ```
   Protex Jenkins Plugin, Protex build step, Processing done
   Finished: SUCCESS
   ```

   If there are errors, they display in the console output. Possible errors include the following:

   - *Protex source path not found* The source path is a field in the configuration; if you receive this error, check the source path entry in the configuration.
   - *Protex credentials not valid* Wrong user name/password.

## 1.4 Using Exposed Variables

When a Jenkins job is configured to run the post-build action *Protex - Create and Scan Actions*, the server URL, Protex project name, Protex template project name, and the Protex project source path are added to the environment variables. They are accessible to any actions that run after this post-build action during a build.

The variables are accessed using the following keys:

`PROTEX_SERVER_URL` - The value is the Protex server URL used in this build.

`PROTEX_PROJECT_NAME` - The value is the Protex project name used in this build.

`PROTEX_TEMPLATE_PROJECT_NAME` - The value is the template Protex project name used in this build.

`PROTEX_PROJECT_SOURCE_PATH` - The value is the Protex project source path; in other words, the path of the target scanned during this build.

## 1.5 Archiving the Log File

You can archive the log file from the Protex scan. This enables you to maintain a history of log files.

✱ **To archive the Protex scan log file:**

1. Navigate to the job configuration.
2. In the **Add post-build action**, select **Archive the artifacts**.
3. In the **Files to archive** field, you can specify a comma-separated list of files to archive from the workspace each time a build is successful.
4. You must enter `BDSToolLog/bdstool.log` in the **Files to archive** field. The current `bdstool.log` file is attached to each new build, enabling you to see the **bdstool** log file for each build.
5. If you have selected the **Archive artifacts only if build is successful** option, then only the builds that pass have the `bdstool.log` file attached to them. Because the log file is most useful when builds fail, selecting this option may not be helpful.

## 1.6 Protex Jenkins Best Practices

### 1.6.1 Concurrency

**Post-build Action Blocking**

Post-build actions use a `BuildStepMonitor` as returned by `getRequiredMonitorService` in the post-build action. This defines how jobs are queued, based on the post-build action.

If this method returns the value of `BuildStepMonitor.BUILD`, then another job that needs to execute the same post build action is queued up and waiting for the previous job to complete its execution before the next job that uses the same post build action can execute. This is the current behavior of the Black Duck scan post-build action. Note that this provides a degree of safety if concurrent builds are executed; however, it blocks multiple jobs to run in a synchronous manner. This may not be desirable because the job is meant to build independent projects, and the scans should be done in parallel. For example, if the Protex scan is configured in a parameterized job, it does not build the project, it only scans it. Multiple projects are blocked, waiting for unrelated projects to finish their scans because the parameterized build is queued up for each project.

If this method returns the value of `BuildStepMonitor.NONE`, then any job executing the same post-build action runs immediately when possible. There is no blocking provided; therefore, no waiting occurs. This allows for concurrent execution.

**Important:** This can be dangerous because the Jenkins workspace for the job could be overwritten by another execution of the same job. Care must be taken to preserve the workspace between dependent jobs.

**Sharing Workspaces with a Downstream Job**

**Note:** Using a custom workspace can be dangerous. The project can perform a cleaning of the workspace, meaning it deletes the files in the workspace. Therefore, the workspace folder should not be a folder containing files required for the OS, or files intended to be saved.

If a job needs to share a workspace with a downstream project, then one way of achieving this is by configuring the downstream project using a custom workspace. The value in the directory text field for the custom workspace can be a path to a specified location. Or, for greater flexibility, the value of the directory text field can be an environment variable. Then an upstream project can define the variable with the value of the workspace used by the upstream project. Each job has a `$WORKSPACE` environment variable defined by Jenkins. The upstream project can set an environment variable to the value of its `$WORKSPACE` variable to share the workspace between the jobs. For example, `CUSTOM_WORKSPACE=$WORKSPACE`. This is one way of sharing the workspace between jobs.

The Jenkins plugins *Multijob* and the *Parameterized Trigger* provide additional functionality to make it easier to set the environment variables available to downstream projects. Note the *Multijob* Jenkins plugin has a dependency on the *Parameterized Trigger* plugin.

You can access the *Multijob* plugin at: https://wiki.jenkins-ci.org/display/JENKINS/Multijob+Plugin.

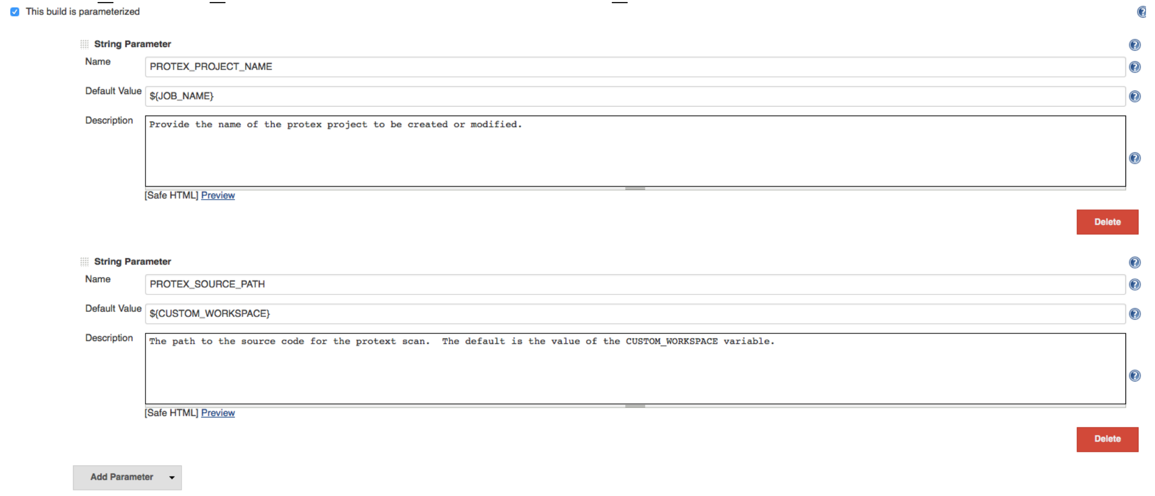
## 1.6.2 Parameterized Job Configuration

To invoke the Protex scan as a separate job, then you must configure a parameterized Job to invoke this job as a downstream job. The *Parameterized Trigger* plugin is required to trigger parameterized jobs.

To create a parameterized job, you must define parameters by selecting the check box **This build is parameterized**, which appears in the configuration after the description text area.

### ✱ To create a new freestyle project:

1. Select this build is parameterized and define the following parameters:
    a. `PROTEX_PROJECT_NAME` Default value = `${JOB_NAME}`
    b. `PROTEX_SOURCE_PATH` Default value = `${CUSTOM_WORKSPACE}`



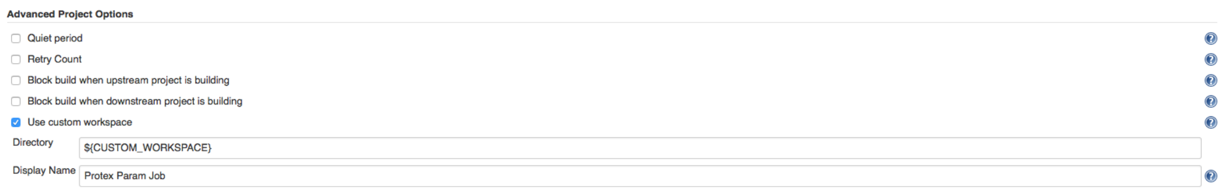
☑ This build is parameterized

String Parameter
Name: PROTEX_PROJECT_NAME
Default Value: ${JOB_NAME}
Description: Provide the name of the protex project to be created or modified.

String Parameter
Name: PROTEX_SOURCE_PATH
Default Value: ${CUSTOM_WORKSPACE}
Description: The path to the source code for the protext scan.  The default is the value of the CUSTOM_WORKSPACE variable.

**Note:** The parameter `PROTEX_SOURCE_PATH` has a default value defined by the environment variable `${CUSTOM_WORKSPACE}`. This is a variable that must be set by the upstream project. This variable value is also intended to be the same workspace as the upstream project. Therefore, if the path of the project is known, enter the default value accordingly. Using the `CUSTOM_WORKSPACE` variable assumes that the workspace between the upstream job and the parameterized job is shared. Extra caution must be taken to assure the integrity of the workspace during the invocation of the parameterized job.

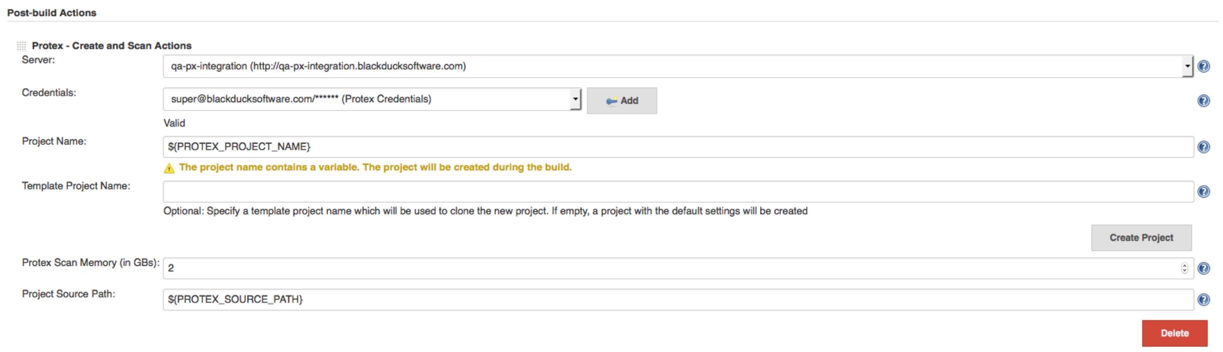
2. Configure the job to execute concurrent builds.
3. Select the check box **Execute concurrent builds if necessary**. This allows multiple upstream jobs to invoke this parameterized job so that each instance of the parameterized job can execute independently of the other invocations of the job.
4. Configure the workspace for the project to share the workspace with the upstream project.
5. Select the check box **Use custom workspace**.
6. In the **Directory** field, type the variable `${CUSTOM_WORKSPACE}`.
7. In the **Display Name** field, type the project name.


**Advanced Project Options**

- ☐ Quiet period
- ☐ Retry Count
- ☐ Block build when upstream project is building
- ☐ Block build when downstream project is building
- ✓ Use custom workspace

Directory: ${CUSTOM_WORKSPACE}
Display Name: Protex Param Job

8. Configure the Protex scan post-build action.
9. Select the Protex server to use.
10. Select the credentials for logging into Protex.
11. In the **Project Name** field, type the previously-defined variable `${PROTEX_PROJECT_NAME}`.
12. In the **Project Source Path** field, type the previously-defined variable `${PROTEX_SOURCE_PATH}`.


**Post-build Actions**

**Protex - Create and Scan Actions**

Server: qa-px-integration (http://qa-px-integration.blackducksoftware.com)
Credentials: super@blackducksoftware.com/****** (Protex Credentials)

Valid
Project Name: ${PROTEX_PROJECT_NAME}
The project name contains a variable. The project will be created during the build.
Template Project Name:
Optional: Specify a template project name which will be used to clone the new project. If empty, a project with the default settings will be created

Protex Scan Memory (in GBs): 2
Project Source Path: ${PROTEX_SOURCE_PATH}

## 1.6.3 Downstream Job Configuration

Downstream job configuration is one of the ways you can configure a job to invoke a parameterized build. There are two methods:

- Configure a downstream job by using the trigger parameterized build post build action.
- Configure a MultiJob.

The method you choose depends on your preference.

In this scenario, you will configure a job to invoke a parameterized job in the post-build action of the job, and you will create a job that builds the source code. Then, a parameterized job is invoked in the post-build action of the job to perform the Protex portion.

**Important:** This scenario utilizes the sharing of a workspace between the jobs and a parameterized build. Therefore, extra care is needed when configuring this type of job.

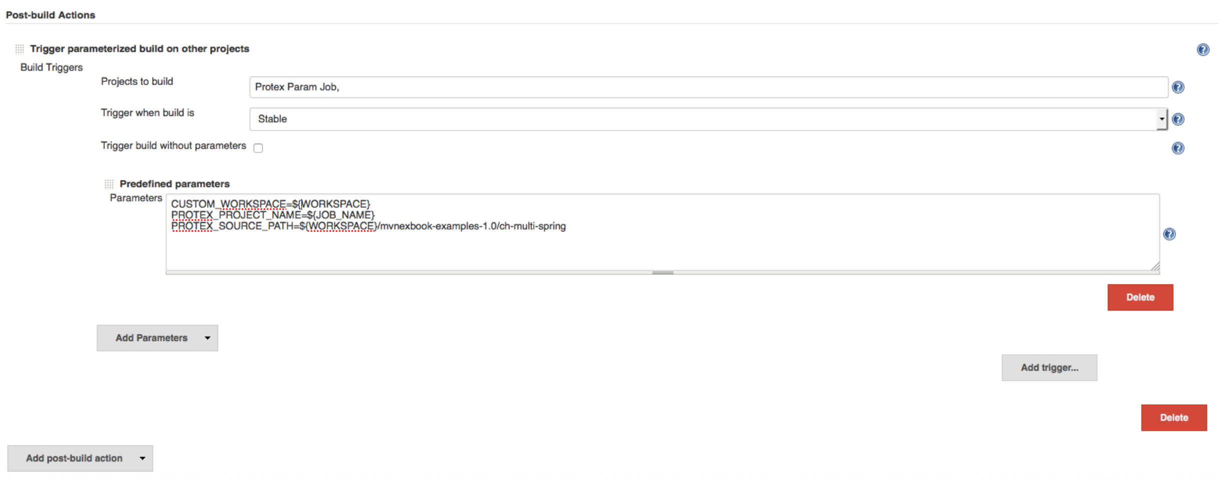
### Configuring a parameterized job for Protex scanning

First a parameterized job that can be invoked by another job must be defined. Refer to the *Parameterized Job Configuration* section for more information.

Once you have define the parameterized job, the next step is to update the job that invokes the parameterized job.

#### To configure the build job:

1. Configure a Jenkins job to build a project.
2. Click **Add a post-build Action**.
3. Select **Trigger parameterized build on other projects**.
4. Select the parameterized Protex build job defined above.
5. Click **Add Parameters**.
6. Select **Build on Same Node**. Since the workspace is being shared by defining the `CUSTOM_WORKSPACE` variable, the build of the downstream project should occur on the same node as the current project.
7. Select **Predefined Parameters**.
8. Define `CUSTOM_WORKSPACE=${WORKSPACE}`. This is the parameter containing the path to the workspace in use. The `${WORKSPACE}` variable contains the current workspace of the job running; therefore, we want to pass this path to the job downstream for it to use the same workspace as this job.
9. Define `PROTEX_PROJECT_NAME` (the name of the Protex project).
10. Define `PROTEX_SOURCE_PATH` (the root directory containing the source code to run the Protex scan).



Projects to build: Protex Param Job,

Trigger when build is: Stable

Trigger build without parameters ☐

Predefined parameters

Parameters:

```
CUSTOM_WORKSPACE=${WORKSPACE}
PROTEX_PROJECT_NAME=${JOB_NAME}
PROTEX_SOURCE_PATH=${WORKSPACE}/mvnexbook-examples-1.0/ch-multi-spring
```

## 1.6.4 Multi-Job Configuration

This section discusses how to configure the jobs to handle defining a single parameterized job to run a Protex scan. This requires sharing the workspace between jobs which can cause instability in your builds if a job modifies the files in a workspace required by a downstream job.

**Note:** This scenario utilizes the sharing of a workspace between the jobs and a parameterized build. Therefore, extra care is needed when configuring this type of job.

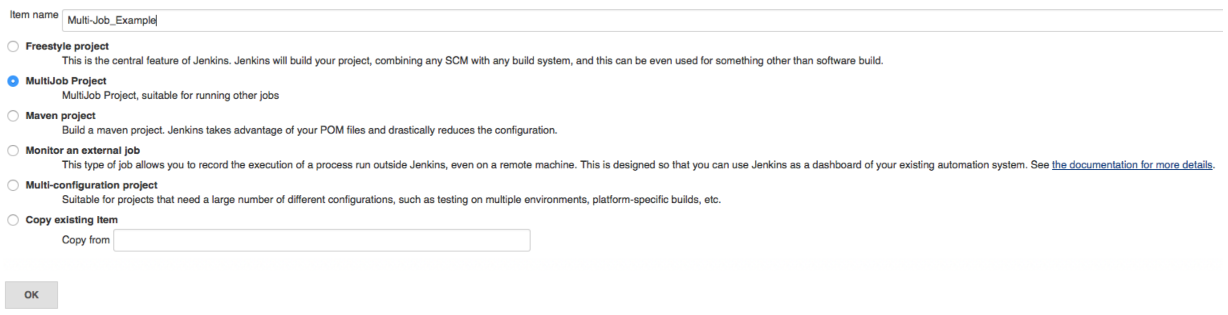
The parameterized job must run in a separate phase than the phase to build the source code. This is because jobs in the same phase can be executed concurrently. Adding a build step to define an additional phase in the multi-job ensures that the first phase (the build phase) completes before the phase invoking the parameterized job.

### Configure a Parameterized Job for Protex Scanning

First, a parameterized job that can be invoked in one of the Multi-Job phases must be defined. Refer to the Parameterized Job Configuration on page 12 section for more information.

### ✱ To configure a MultiJob:

1. Create a new MultiJob project.



Item name: Multi-Job_Example

○ **Freestyle project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

● **MultiJob Project**
MultiJob Project, suitable for running other jobs

○ **Maven project**
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

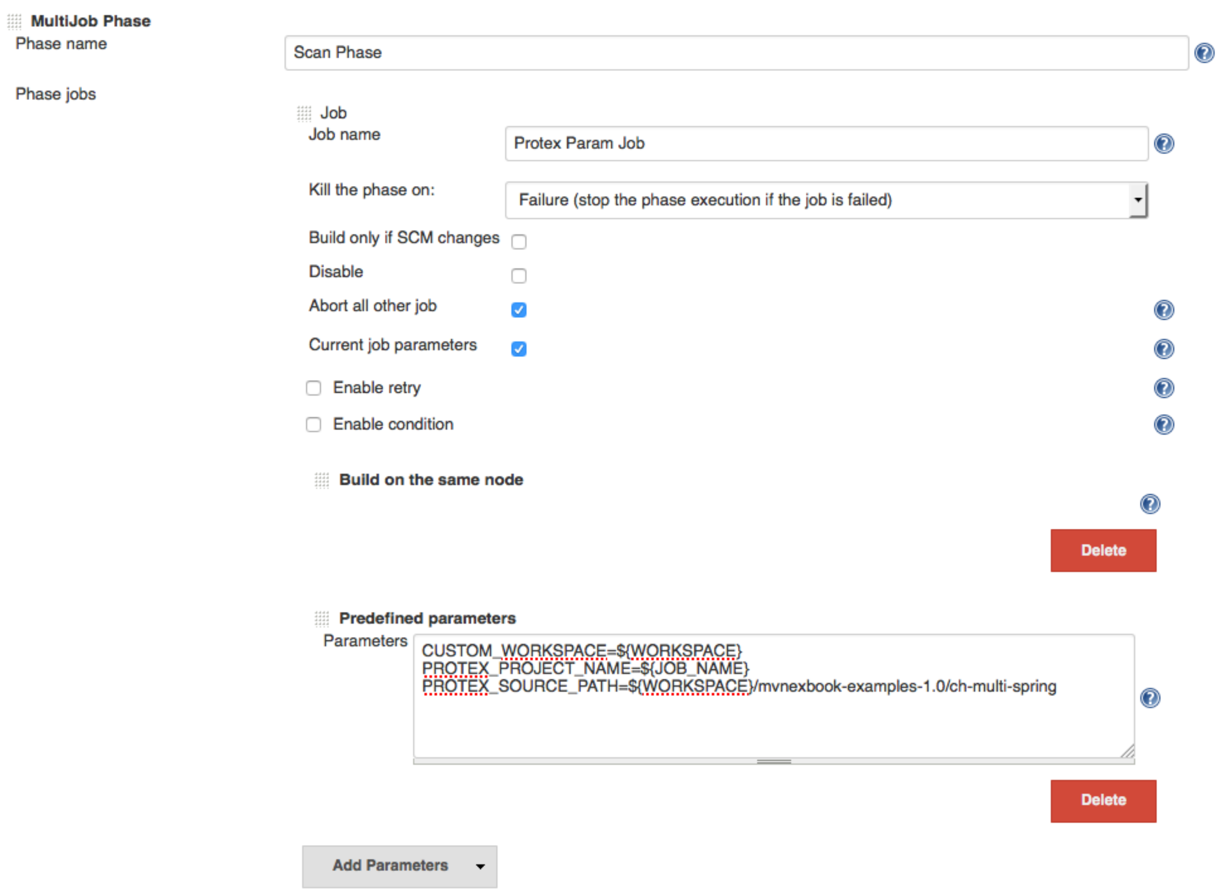
○ **Monitor an external job**
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See the documentation for more details.

○ **Multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

○ **Copy existing Item**
Copy from

2. In the Build section, click **Add build step**.

3. Select **MultiJob Phase**.
4. Type a name into the **Phase Name** field; for example, *Build Phase*.
5. Click **Add jobs...**, and select the job to build the source code.
6. Click **Add build step**.
7. Select **MultiJob Phase**.
8. Type a name into the **Phase Name** field; for example, *Dependency Resolution*.
9. Click **Add jobs...**.
10. Select the name of the parameterized job defined in the **Parameterized Job Configuration** section.
11. Click **Advanced**.
12. Click **Add Parameters**.
13. Select **Build on the same node**.
14. Select **Predefined parameters**.
15. Define `CUSTOM_WORKSPACE=${WORKSPACE}`. This is the parameter containing the path to the workspace in use. The `${WORKSPACE}` variable contains the current workspace of the running job; therefore, we want to pass this path to the job downstream for it to use the same workspace as this job.
16. Define `PROTEX_PROJECT_NAME` (the name of the Protex project).
17. Define `PROTEX_SOURCE_PATH` (the root directory containing the source code to run the Protex scan).



MultiJob Phase
Phase name: Scan Phase
Phase jobs
Job
Job name: Protex Param Job
Kill the phase on: Failure (stop the phase execution if the job is failed)
Build only if SCM changes: ☐
Disable: ☐
Abort all other job: ✓
Current job parameters: ✓
☐ Enable retry
☐ Enable condition
Build on the same node
Predefined parameters
Parameters:

```
CUSTOM_WORKSPACE=${WORKSPACE}
PROTEX_PROJECT_NAME=${JOB_NAME}
PROTEX_SOURCE_PATH=${WORKSPACE}/mvnexbook-examples-1.0/ch-multi-spring
```

**Note:** You can either define the build steps in the MultiJob to invoke and build the source code, or you can define separate parameterized jobs to build the source code. If you specify separate parameterized jobs to build the source code, they must share the same workspace as the scan job; therefore, they must share the same workspace and build on the same node as the scan.

## 1.7 Troubleshooting the Protex Jenkins Plugin

**Symptom:** Out of Memory error message:

```
FATAL: PermGen space java.lang.OutOfMemoryError: PermGen space …..
```

**Cause:** Insufficient memory.

Based on your environment, you may have to increase the Java memory allocations.

**Solution 1:** In the job configuration, increase the value in the **Protex Scan Memory (inGBs)** field. For more information, refer to Configuring the Protex Jenkins Plugin - Project Configuration on page 7.

**Solution 2:** If that proves unsatisfactory, you can change the Tomcat memory configuration. On Tomcat, ensure you have higher `Xmx` and `MaxPermSize` values in **setenv.sh** or **.bat**.

```
export JAVA_OPTS="-Xmx512m -XX:MaxPermSize=256m"
```

**Symptom:** Protex scan fails in Jenkins with *Out of Memory* error:

```
Exception Logged[Out of memory] java.lang.OutOfMemoryError:
```

With the following stack trace:

```
ERROR: [ERROR]
com.blackducksoftware.integration.protex.exceptions.ProtexScannerEx
ception: BuildToolIntegrationException :

at
com.blackducksoftware.integration.protex.jenkins.ProtexScanner.runP
rotexScan(ProtexScanner.java:298)

at
com.blackducksoftware.integration.protex.jenkins.ProtexScanner.call
(ProtexScanner.java:166)

at
com.blackducksoftware.integration.protex.jenkins.ProtexScanner.call
(ProtexScanner.java:27)

at hudson.remoting.LocalChannel.call(LocalChannel.java:45)

at
com.blackducksoftware.integration.protex.jenkins.PostBuildProtexSca
n.perform(PostBuildProtexScan.java:262)

at hudson.tasks.BuildStepMonitor$3.perform(BuildStepMonitor.java:36)

at hudson.model.AbstractBuild$AbstractBuildExecution.perform
(AbstractBuild.java:780)

at
hudson.model.AbstractBuild$AbstractBuildExecution.performAllBuildSt
eps(AbstractBuild.java:752)

at hudson.model.Build$BuildExecution.post2(Build.java:183)

at hudson.model.AbstractBuild$AbstractBuildExecution.post
(AbstractBuild.java:705)

at hudson.model.Run.execute(Run.java:1617)

at hudson.model.FreeStyleBuild.run(FreeStyleBuild.java:46)

at hudson.model.ResourceController.execute
(ResourceController.java:88)

at hudson.model.Executor.run(Executor.java:237)

Caused by: java.lang.Exception:

at
com.blackducksoftware.protex.plugin.BuildToolIntegrationException.u
nknownCommandFailure(BuildToolIntegrationException.java:136)
```

```
    at
    com.blackducksoftware.protex.plugin.BlackDuckCommand$ForkedClientDr
    iver.execute(BlackDuckCommand.java:350)

    at com.blackducksoftware.protex.plugin.BlackDuckCommand.run
    (BlackDuckCommand.java:113)

    at
    com.blackducksoftware.integration.protex.jenkins.ProtexScanner.runP
    rotexScan(ProtexScanner.java:272)

    ... 13 more
```

**Cause:** Possible insufficient scan memory. Examine the **bdstool.log** file to see if errors are logged that could explain the failure.

If the scan has insufficient memory, the following error displays at the bottom of the log file :

```
    ERROR : 20150928T182807Z {BDSClientManager.execute:411} Exception
    Logged[Out of memory] java.lang.OutOfMemoryError: GC overhead limit
    exceeded

    GC overhead limit exceeded[java.lang.OutOfMemoryError: GC overhead
    limit exceeded
```

**Solution:** Contact the Black Duck Support team.

**Symptom:** If you see the error:

```
[INFO] --> java.lang.ClassNotFoundException:
com.blackducksoftware.bdsclient.BDSClientDriver

at java.net.URLClassLoader$1.run(URLClassLoader.java:372)

at java.net.URLClassLoader$1.run(URLClassLoader.java:361)

at java.security.AccessController.doPrivileged(Native Method)

at java.net.URLClassLoader.findClass(URLClassLoader.java:360)

at java.lang.ClassLoader.loadClass(ClassLoader.java:424)

at java.lang.ClassLoader.loadClass(ClassLoader.java:357)

at com.blackducksoftware.protex.plugin.BDSToolRemoteRunner.main
(BDSToolRemoteRunner.java:48)
```

**Solution:** Verify that your Java Developers Kit (JDK) is using the correct certificates.

# Chapter 2: Protex Jenkins Plugin Release Notes

## Protex Jenkins Plugin Release 1.4.1

- Resolved an issue regarding the *Class not found* exception error.

## Protex Jenkins Plugin Release 1.4.0

- New action option to force a full rescan with the next build.
- Excludes *BDSToolLogs* from scans, preventing the log files being scanned as part of the project.
- Ability to pull a Protex report and archive in Jenkins continuous integration.
- Inclusion of the plugin update notification for Jenkins (notifications and check box for update).
- Relaxes the Jenkins job synchronization so that parametrized jobs can be run in parallel.

## Protex Jenkins Plugin Release 1.3.2

- Fixed an issue where some jobs were hanging.
- Jenkins jobs can now be aborted.

## Protex Jenkins Plugin Release 1.3.1

- Fixed an issue where the password was logged in clear text in some instances of **bdstool** failures.
- Retired 32-bit VM support.
- Improved warnings for failed LDAP logins.
- Exposed the Protex Project/Template as an environment variable for additional post-build steps.

## Protex Jenkins Plugin Release 1.3.0

- The job configuration contains a new field called **Protex Scan Memory (in GBs)**. This field enables you to specify the heap size that the Protex scan should use.
- When the plugin updates a Protex project before a scan, it no longer updates the entire project. The plugin now performs a sparse update on the project, preventing an entire re-calculation of the Bill of Materials.
- The `bdstool.log` file is now located in the job workspace when a build runs.
  - On Masters, the log file is located at `${WORKSPACE}/BDSToolLog/bdstool.log.`
  - On Slaves, the log file is located at `${WORKSPACE}/<job-name>//BDSToolLog/bdstool.log`.

### Protex Jenkins Plugin Release 1.2.0

- Protex version 7.1.2 and higher is now required for the v. 1.2.0 Protex Jenkins plugin.
- Authenticated proxies are now supported for SDK calls and Protex scanning.
- If a proxy is defined, the proxy settings are now used when validating the server URL.
- The migration toolset is no longer included in the Protex Jenkins plugin versions 0.12 to 1.1.0. For more information on migration, refer to Migration - Manually Migrating a Project Configuration.
- Improved stability and avoiding memory leaks by running the bdstool analyze command as a sub-process.
- Bug fixes for proxies between the Jenkins (master/salve) and the Protex system.
- Now supporting authenticated proxies with Basic authentication.
- Proxy authentication with Digest not yet supported.
- Tip: this version does not support the migration from 0.12 to 1.1.0. If migrating, install 1.1.0 first, migrate the jobs, and then upgrade to 1.2.0.

### Protex Jenkins Plugin Release 1.1.0

- Post-build step to run Protex Analysis.
- Post-build step to fail build based on files pending ID or license issue.
- Configure globally Protex servers.
- Configure login credentials for Protex server, globally and secure with the Jenkins credentials plugin.
- Create Protex project interactively or dynamically.
- Configure Protex project name based on variables, including environment variables.
- Create Protex project as clone (settings) from another template.
- Interactive migration of a job configured for Protex plugin 0.12.
- Migration tools to migrate existing jobs using Protex plugin 0.12.

# Chapter 3: Black Duck Support

If you have questions or find issues, contact Black Duck Software.

For the latest in web-based support, access the Black Duck Software Customer Support Web Site: https://www.blackducksoftware.com/support/contact-support

To access a range of informational resources, services and support, as well as access to Black Duck experts, visit the Black Duck Customer Success portal at: https://www2.blackducksoftware.com/support/customer-success

You can also contact Black Duck Support in the following ways:

- **Email**: support@blackducksoftware.com
- **Phone**: +1 781.891.5100, ext. 5
- **Fax**: +1 781.891.5145
- **Standard working hours**: Monday through Friday 8:00 AM to 8:00 PM EST

**Note:** Customers on the **Enhanced Customer Support Plan** are able to contact customer support 24 hours a day, 7 days a week to obtain Tier 1 support.

If you are reporting an issue, please include the following information to help us investigate your issue:

- Name and version of the plugin.
- Black Duck product name and version number.
- Third-party integrated product and version; for example, Artifactory, Eclipse, Jenkins, Maven, and others. For Black Duck Hub, only Jenkins, TeamCity, and Bamboo is supported.
- Java version.
- Black Duck KnowledgeBase version, where applicable.
- Operating system and version.
- Source control management system and version.
- If possible, the log files, configuration files, and Project Object Model (POM) XML files.

## Services

If you would like someone to perform Black Duck Software tasks for you, please contact the Black Duck Services group. They offer a full range of services, from planning, to implementation, to analysis. They also offer a variety of training options on all Black Duck products. Refer to https://www.blackducksoftware.com/services/ for more information.

## 3.1 Training

Black Duck training courses are available for purchase. Learn more at https://www.blackducksoftware.com/services/training.

View the full catalog of our online offerings: https://www.blackducksoftware.com/academy-catalog.

When you are ready to learn, you can log in or sign up for an account: https://www.blackducksoftware.com/academy.

# Appendix A: Migrating From Plugin Version 0.x

You must install Protex Jenkins plugin Version 1.1.0 to migrate from Protex Jenkins plugin Version 0.12. Next, follow the instructions in Appendix A: *Migrating From Previous Plug-in Versions* in the *Installing, Configuring, and Using the Protex Jenkins Plug-in Version 1.1.0* user guide, which was included with the version 1.1.0 installer. Once you have migrated all jobs, you can upgrade to the Protex Jenkins plugin version 1.2.0 and higher, and uninstall Protex Jenkins plugin Version 0.12. The steps are:

1. Upgrade version 0.x to 0.12.
2. Upgrade 0.12 to 1.1.0.
3. Upgrade 1.1.0 to 1.2.0 or higher.
4. Uninstall version 0.12.

For details of the migration, refer to *Appendix A: Migrating From Previous Plug-in Versions* in the *Installing, Configuring, and Using the Protex Jenkins Plug-in Version 1.1.0* user guide.

## A.1 Migration - Uninstall the version 0.x Plugin

After you have completed the migration of all your Jenkins jobs, uninstall the version 0.x Protex Jenkins plugin.

**To uninstall the plugin:**

1. Log in to Jenkins as admin.
2. Go to **Manage Jenkins > Manage Plugins**.
3. Click the **Installed** tab.
4. Locate the **version 0.x BD Protex Plugin for Jenkins CI**.
5. Click **Uninstall**.