The Hudson Book

Working with your helpful, extensible continuous integration server.

Oracle, Inc.

Manfred Moser
Tim O’Brien

The Hudson Book

Manfred Moser

Tim O’Brien

The Hudson Book

Ed. 1.0

Contents

Abstract

Eclipse Hudson is a widely used, open source continuous integration server. The Hudson Book aims to be the authoritative and up to date resource about Hudson written by the community for the community.

Copyright

Foreword: 1.0

This book covers Hudson, the most widely used open source Continuous Integration Server. Oracle is excited to support the continued development of Hudson as the Hudson community completes its transition to the Eclipse Foundation. An important part of making sure that Hudson offers a supportable, world-class experience to end users is quality documentation.

If you have any feedback or questions, you are encouraged to post on the Hudson project mailing lists, or to contact the authors directly via the Hudson project. We're here to help, and we look forward to your feedback.

Manfred Moser, Tim O'Brien

August, 2011

Edition: 1.0

Chapter 1

Introducing Hudson

What is Hudson?

Hudson is a powerful and widely used open source continuous integration server providing development teams with a reliable way to monitor changes in source control and trigger a variety of builds. Hudson excels at integrating with almost every tool you can think of. Use Apache Maven, Apache Ant or Gradle or anything you can start with a command line script for builds and send messages via email, SMS, IRC and Skype for notifications.

In addition to providing a platform for continuous integration builds, Hudson can also be extended to support software releases, documentation, monitoring, and a number of use cases secondary to continuous integration. In short, if you can think it, Hudson can do it. From automating system administration tasks with Puppet and verifying infrastructure setup with Cucumber, to building and testing PHP code, to simply building Enterprise Java applications - Hudson stands ready to help.

1.1 Continuous Integration

Martin Fowler and Kent Beck are largely credited with the first use of the term "Continuous Integration" as applied to software development with Beck's seminal 1999 book *Extreme Programming Explained* being the first published work touching upon the idea of creating systems to continuously build and test systems in response to changes in source control. In the decade since this concept was introduced, continuous integration is now an established, standard practice used across an entire industry.

Note

Beck, Kent (1999). Extreme Programming Explained. ISBN 0-201-61641-6.

The idea, put simply, is that software development efforts are much easier to manage when test failures and other bugs can be identified closer to the time they were introduced into a complex system. Let's examine the differences between a development effort using a Continuous Integration server, and a development effort not using Continuous Integration.

1.1.1 A Hypothetical Development Team

Consider a hypothetical group of 20-30 developers working on a large enterprise application.

This development team consists of 20-30 developers divided into groups of 5-12 developers working on focused features and components in a larger system. One group focuses on the database and a collection of APIs shared by all other groups, one group focuses on a complex front-end web application, and another group focuses on back-end systems such as billing and inventory which interface with a large Enterprise Service Bus (ESB).

In this environment the business drives a number of overlapping development schedules. The web application team tends to work on two week development schedules, and the back-end group performs releases in response to changes in the ESB. Any given month resemble a highly choreographed sequence of code freezes, development sprints, and products releases with little room for error and inefficiency.

In summary, these 20-30 developers are under constant pressure to deliver, and the technical managers are tasked with orchestrating the efforts of these various teams to deploy code to production on a regular schedule. This is what the enterprise feels like in 2011: it's very busy, and if you happen to find a bit of free time there is always someone who can create a new requirement.

1.1.2 Without Continuous Integration

Without Continuous Integration the various teams have to factor in "integration time" into the schedule. Unit tests might be run by individual developers, but integration tests are only executed when the system is ready to be deployed. There's nothing automatic about the process, and, very often only a handful of developers have the appropriate environment to build and deploy to production.

Each team, each group is responsible for testing the larger system and making sure that their changes don't interfere with another group's changes. One week before a release, everything stops so that a single "release manager" can build a canonical build on a single build machine and deploy this build to a testing environment.

Each time one group needs to deploy a specific part of the system, every group needs to stop, synchronize, and test. If the web application group needs to deploy to production, all of the other groups need to synchronize with that group and make sure that all of the tests pass during a build.

This constant need to stop, integrate, and verify introduces a synthetic limit to the scale of development. As development teams scale and grow larger, and as systems become increasingly complex and intertwined your team spends more time on integration and testing.

Without continuous integration, developing modern applications is practically impossible without dramatic inefficiency. There is too much unproductive down-time, and far too much waste. The development group described above would be hard pressed to release anything more than a global update once a month.

1.1.3 With Continuous Integration

With Continuous Integration, the system completes a build, test, deploy, and integration in response to every single commit. If a developer in the web application group checks in code at 2 AM, Hudson kicks off a build, runs unit tests, deploys the code to a new server, and performs a set of integration tests. If this build fails or the tests encounter an unexpected condition, everyone is notified of this failure.

With continuous integration, no one needs to drop everything and run a release build, these builds are generated every single day, and in the most mature environments, a fully tested and verified system can be deployed to production at any time. In other words, when you automate build, test, and verify using a tool like Hudson you can continue developing your applications without having to wait (or synchronize) on some manual build, test, verify process.

Making these processes automatic has another important side-effect, it makes the development process more scaleable. When your teams don't have to stop to actively test and collaborate with one another, it is much easier to add additional developers to a project. Without continuous integration you have to stop and synchronize release schedules. With continuous integration you reduce the risk associated with a particular software development cycle.

1.2 Minimizing Technical Debt

Another way to think about Hudson is in terms of technical debt.

When you amass "technical debt" you are making a decision that yields a short-term benefit while creating a problem to be solved in the future. If you've developed code under strict time lines, technical debt is nothing new to you. Every system has some form of technical debt. If you've ever sacrificed quality for the expedient solution, you've experienced technical debt: "the code had to be deployed tomorrow and the deadline was fast approaching".

From the perspective of builds and tests, it is much easier to develop a system without having to worry about unit tests and integration tests. You could speed up implementation time for a feature by just skipping the tests, but you'll eventually pay for this approach ten-fold in the form of bug fixes.

Continuous integration along with a commitment to test-driven development helps minimize your exposure to risky technical debt. Because you are running the unit and integration tests every few minutes, you don't have the opportunity to let problems sit and fester unaddressed. When you develop without continuous integration, you are amassing the potential for technical debt. You can choose to fix bugs as they are introduced, minimizing both the scope and severity of defects by catching them early, or you can develop for weeks and run tests only at the end of the development cycle.

While you might only deploy to production once a month, your Hudson installation deploys once a commit, and when you are dealing with increasingly complex systems this granularity makes sure that any glaring quality detours are confined to a single commit.

1.3 Push it to Production with Hudson

The infrastructure that powers applications has grown more and more complex over time. The web application that ran on a production network of 10 hefty Unix machines in 1999 has evolved into an architecture that can potentially span hundreds (or thousands) of nodes on a computing grid. Companies like Apple and Oracle are deploying infrastructure on a massive scale compared to where the industry was operating just a few years ago. This movement toward Cloud-based deployments and Platform-as-a-Service (PaaS) has meant that infrastructure is now more likely to be compiled and deployed on an as-needed basis.

Organizations are using tools like Hudson to setup production deployments and automate the management of cloud-based infrastructure. This carries the idea of Continuous Integration forward into the realm of Continuous Deployment. While the idea may sound risky if you work in a relatively slow-moving industry, social networks like Facebook and social media services such as Flickr have been building complex developer/operations (devops) systems that allow them to deploy code to production multiple times a day.

While this emerging trend is still confined to the largest deployments, the idea of connecting your Continuous Integration server directly to your production systems is gaining support. Hudson can be a critical part of automating these approaches to continuous deployment. Proven integration with tools like Puppet and Cucumber allow innovative devops staff to take continuous integration in some surprising directions.

1.4 General Purpose Scheduler

Hudson can also be viewed as a general purpose scheduler, a replacement for cron or Quartz in your production network. While Hudson is generally viewed as development infrastructure, a class of infrastructure usually considered separate from the rarefied world of production, there's nothing stopping you from using Hudson as a general purpose scheduler or server running within a production network.

If you have a requirement for scheduled services or if you need to perform a set of on-demand jobs in a production network, install Hudson and you've gained a friendly interface and a tool that can integrate with anything in the world. There's nothing stopping you from using Hudson in production.

1.5 The Hudson Community at Eclipse

Hudson has an evolving community of developers and organizations interested in taking Hudson to the next level of stability and reliability. Following the creation of the Jenkins fork in early 2011, several members of the community decided to commit themselves to reconstituting the Hudson community and moving the project to the Eclipse Foundation. This move promises to create a more stable approach to open source governance while creating a level playing field for interested individuals and organizations.

Throughout its history the Hudson project lacked a formal structure and avoided process. Grown from a pet project, the governance of Hudson was ad-hoc. While this ad-hoc approach scaled for a few years, as more organizations decided to participate and contribute to the project the Hudson community identified a need for more formal procedures on issues such as release management, code provenance, and general technical direction. It is now the desire of the Hudson project to move toward a more formal process and structure with the end-goal of creating a healthy, open environment with well-defined rules, as well as a development process and release processes for building a high-quality and reliable continuous integration server.

Eclipse has a history of encouraging healthy environments for open source projects. It is an organization supported and respected by millions of end-users, thousands of contributors, and hundreds of active companies. In addition to being a great home for the Hudson project, Eclipse is also an important user of the software. Hudson has become an integral part of the development infrastructure at Eclipse, and the requirements of the Eclipse Foundation mirror the requirements of some of the largest enterprises in the world.

1.6 The Hudson Project

The main source of information about Hudson are the Eclipse Hudson website and the legacy website. The sites contain links to resources like the wiki, an issue tracker, the user forum, a few mailing lists, the source code and more. If you are looking for download Hudson or learn about participating in the, you can start with the Eclipse Hudson project's web site.

At the time this document was developed, the Hudson project at Eclipse was focused on the following tasks:

- Improving the core of Hudson by bolstering the test harnesses,
- Creating a new performance harnesses allowing for more detailed performance assessment and testing.
- Developing a fully automated test and release infrastructure

Some of the larger, medium-term goals of the Hudson project are:

- **Improving stability** - The Eclipse project is investing in QA efforts to certify Eclipse to run on a set of standard platforms. Improving the reliability and stability of Hudson will enable more predictable behaviour for end-users and also enable an ecosystem of interested third-party providers of Hudson support services.
- **Improving performance** - The project is focused on addressing performance, especially for organizations using Hudson to support large numbers of projects.
- **Improving Maven 3.x integration** - Sonatype has contributed a completely reworked Maven 3 plugin which takes advantage of changes to Maven 3 which were made to increase the ease with which Maven could be integrate in IDEs such as Eclipse and CI servers such as Hudson.
- **Improve the UI** - Prior to the migration to the Eclipse Foundation, the Hudson project had standardized on Jelly as a UI technology. The Hudson is currently focused on moving the UI system way from Jelly and toward a more modern approach to creating a solid, reactive user interface for Hudson.
- **Closer Integration for Core Plugins** - One of the first steps of the Hudson project during the transition to Eclipse was to define a tiered set of Hudson plugins and establish a level of commitment and support to the most widely used plugins. It is the intention of the Eclipse project to increase the level of support and integration for these core plugins.
- **A standard OSGi Run-time Model based on Eclipse Equinox** - Hudson's legacy architecture was full of issues arising from the lack of a unified architecture. The Hudson project is work to move Hudson toward an OSGi model based on Eclipse Equinox. This will not only lead to a more stable CI server, it will make it easier to integrate and embed Hudson in systems like the Eclipse IDE and other, widely-used OSGi containers.
- **Support the a standard component model using JSR 330** - Early efforts were made to move the plugin API toward an approach compatible with JSR 330. This effort not only makes it easier to write and test Hudson plugins, it also aligns Hudson will the way that Java is developed in 2011 using frameworks and libraries such as Google Guice and VMWare's Spring Framework.
- **Standard web services using JAXRS** - Many of the plugins and core systems in Hudson are being migrated to a system that presents UI services as a collection of REST services. This foundational effort will set the stage for Hudson to move toward a richer, more reactive UI architecture possibly involving a migration to a SOFEA approach to user interface.
- **Backwards Compatibility** - Despite this list of ideas and plans, the Hudson community is also focused on supporting existing plugins and Hudson APIs.

1.7 Hudson's License

As Hudson has completed the transition to the Eclipse Foundation, the core of Hudson is covered by the Eclipse Public License version 1.0. The Open Source Initiative categorizes the Eclipse Public License as a "popular and widely used license with a strong community". The EPL is a limited copy-left license designed to be a "business-friendly" alternative to the GPL. More information about the Eclipse Public License can also be found on Wikipedia.

If you have any questions about the Eclipse Public License, the Eclipse Foundation also maintains a helpful FAQ about the license. A big reason for the move to the Eclipse Foundation was to make sure that the project was on solid ground for licensing issues. Eclipse takes intellectual property and code provenance very seriously and maintains a strict contribution review process to make sure that every project release complies with a set of licensing standards. If you download it from Eclipse, if it is stored in Eclipse's source control systems, it is released under the EPL.

Note
While the core of Hudson is licensed under the EPL, Hudson is a container for running various Hudson Plugins. These plugins may be covered by a variety of licenses, and if you are redistributing Hudson along with a set of 3rd party plugins, you should consult each individual plugin project to find a specific license.

Chapter 2

Installing and Running Hudson

Hudson has a reputation for being both easy to install and very adaptable to running in a variety of environments. You can run Hudson as a stand-alone web application, or you can run Hudson within an existing servlet container. The Hudson project also maintains OS-specific packages for Redhat, Debian, Ubuntu, and other Linux distributions which make installing Hudson on these platforms almost effortless.

The following sections detail the installation process for Hudson. There are two different approaches to installing Hudson:

WAR File
: The Hudson web site provides a Java web archive file (WAR) for download. This file can either be started directly or used in an existing Java servlet container or application server.

Native Package
: Besides the web archive you can download packages for Hudson suitable for the use with the native package management systems on Ubuntu/Debian, Oracle Linux, Redhat/Fedora/CentOS and openSUSE.

It is a best practice to install Hudson as a service automatically started when an operating system boots. On a Windows machine this can be as straightforward as configuring a new Windows Service, and on a Linux machine this is as easy as dropping the appropriate script in */etc/init.d*. The following sections outline the process for configuring Hudson as a service on various operating systems.

2.1 Prerequisites

While Hudson can run on a variety of machines under an almost infinite combination of JVMs, Operating System, and infrastructure. The Hudson project is targeting a set of standard operating systems and Java versions. This section outlines some of the expectations - the preconditions that are necessary to install and run a Hudson server.

2.1.1 Software Prerequisites

Hudson only has one prerequisite, a Java Runtime Environment (JRE) compatible with Java 6 or higher. Hudson is most often run with the JRE that is bundled with a Java Development Kit (JDK) installation. We recommend using the latest version of the JDK/JRE Java 6. If you are running on a modern Linux distribution such as Ubuntu, it is often possible to install the OpenJDK 6 project using a tool like apt-get (or yum on a Redhat distribution).

Download Oracle Java 6 JDK
The latest version of Oracle's JDK 6 is the officially supported runtime. To download the latest release of the Oracle JDK, go to the Java JDK 6 Download Page, and download the latest Java 6 JDK.

Installing OpenJDK 6 on Linux
OpenJDK packages are available in both RPM and DEB format which should cover most Linux distributions. Installing OpenJDK 6 on your Linux machine can be done by following the instructions OpenJDK project site.

2.1.2 Hardware Prerequisites

As there is such a wide variety of Hudson deployments, from the Hudson server that runs a single project once every few days, to the Hudson server which serves as a master orchestrator of a hundred or more nodes in a massively distributed build grid, it is impossible to predict and recommend the necessary CPU power and disk space your particular installation will demand. Your own builds and your own systems will often dictate the specific hardware and network requirements for your Hudson instance.

What this section can do is provide a few recommendations that will help size and configure your infrastructure:

Network
If your Hudson server is very active you should make sure that there is sufficient bandwidth and low latency between your Hudson server and your source control system. Hudson is configured to periodically test the source control system and look for changes. In general, the critical components of your development infrastructure should be co-located with one another. Your CI server should be next to your SCM server. If your Hudson server is configured to automatically publish build artifacts to a repository manager, you should also make sure that the network connection between these two machines can support the expected level of activity.

CPU

In the absence of build activity, Hudson doesn't consume very much in the way of CPU power. You can let Hudson run in the background, waiting for a build to trigger without having to worry about consuming more than a few processor cycles. With build activity, you should take the CPU requirements of your own development workstation and use that as a baseline for your Hudson server. Multiply the power you use as a single developer by the number of concurrent builds your Hudson server needs to handle. In general a two or four processor build machine with a powerful processors is a good starting point for a medium-sized work-group of 5-10 developers.

Memory

Compilers can often eat massive amounts of RAM. As with CPU configuration, take your own development workstation as a baseline and try to multiple that requirement by the number of expected concurrent builds. If your local build can easily run with 2 GB of RAM and you expect three concurrent build threads at any given time, then make sure that your Hudson instance can consume 6 GB of RAM on a server. You'll have to use your own builds as a guide, but 4-8 GB of RAM should be adequate for a medium-sized work-group.

I/O

As with Memory requirements, builds, compilers, and tools like Maven and Ant tend to make disk drives go berserk with activity. If your builds are particularly I/O intensive, you are going to want to make sure that Hudson has sufficient local storage (or high-speed SAN storage). If you try to run a massive build job using an NFS mount to store output you are likely to see unacceptable build performance. Don't "skimp" on hard drive quality or available storage space. I/O is often the primary gating factor in a large enterprise-class builds. Builds eat I/O.

Storage

You should be able to get off the ground with 5-10 GB, but once you start building your projects, you are going to want to preserve some build history. This build history can provide invaluable insight into the trends surrounding a particular build. If you were able to look at your project's build history over a few weeks, you might notice that one developer in particular continues to break the build for your team. Alternatively, you can also configure Hudson to keep track of important metrics and trends such as test failures over time.

If you think that keeping build history around is important, you should configure Hudson with an appropriate amount of disk space. What this number is is heavily dependent on the size of your projects and the size of the generated binary artifacts. Larger projects with tens of thousands of lines of code can often get by with 60-100 GB, but the largest Hudson installations can easily reach a TB or more depending on the activity in a given development group.

Again, these recommendations come with a caveat: the authors of this document are not familiar with your application's build. If your tests generate a huge amount of I/O activity or if your builds are particularly difficult to compile, you should use your best judgement. A good rule of thumb with Hudson deployments is to make sure that your Hudson server can satisfy a reasonable number of concurrent builds. Should it be able to run two builds in parallel? How about ten? This all depends on the size of your development and the frequency with which code changes are committed to source control.

Warning
Continuous Integration servers drive quality, and if they are seriously under-powered they tend not to work. That tiny afterthought of a machine that management could spare for Hudson may end up taking several hours to run a build that takes minutes on your workstation. If software is something you produce, and if you value quality, you need to run a machine that will be able to keep up with your developers. If you treat the hardware running your CI system as an afterthought, it you can't afford to invest in the necessary hardware, don't be surprised if your quality, morale, and productivity take a nose dive.

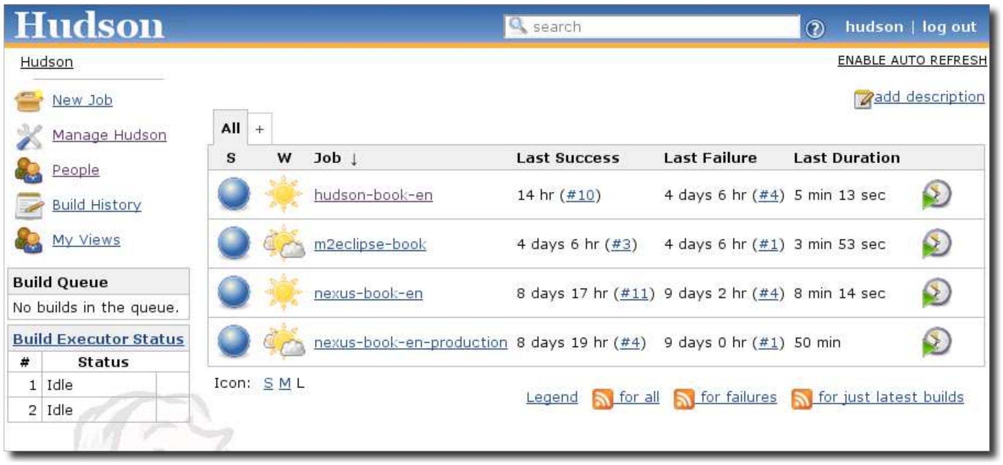
2.2 Installing Hudson with the WAR File Distribution

The WAR file available for download on the Hudson web site is an executable WAR that has a servlet container embedded within it. Once downloaded and copied to the desired directory, you can start Hudson with the following command:

```
java -jar hudson.war
```

Tip
That's it. The Hudson WAR ships with everything it needs to start itself.

This will start the servlet container as the current operating system user, inheriting access rights to the file system and so on. The Hudson home directory will be set to the `.hudson` folder in the user's home directory. Once started the web-based Hudson user interface will be available at http://localhost:8080/hudson



Build Queue
No builds in the queue.

Build Executor Status	
#	Status
1	Idle
2	Idle

S	W	Job ↓	Last Success	Last Failure	Last Duration
		hudson-book-en	14 hr (#10)	4 days 6 hr (#4)	5 min 13 sec
		m2eclipse-book	4 days 6 hr (#3)	4 days 6 hr (#1)	3 min 53 sec
		nexus-book-en	8 days 17 hr (#11)	9 days 2 hr (#4)	8 min 14 sec
		nexus-book-en-production	8 days 19 hr (#4)	9 days 0 hr (#1)	50 min

Figure 2.1: Hudson Application Window

Note

This approach is suitable for testing and exploring Hudson, but it is not recommended to run Hudson as an executable WAR in production. Sure, it only took you a few seconds to download and start Hudson, but running dependable development infrastructure usually requires more than just a developer running "java -jar" from the command-line. If you really want to get serious about running Hudson, configure Hudson as a service on Windows or Linux.

2.3 Deploy Hudson to a Servlet Container

Conveniently the WAR file is suitable to be deployed in most commonly used Java servlet containers and application servers. The detailed process differs for these containers but in general the required steps are as follows:

Set up HUDSON_HOME
: Hudson locates its configuration files and all other data in one folder and a collection of sub-folders to hold Job and build data. This folder should be configured by setting up an environment variable: `HUDSON_HOME`. The application will pick up this setting and use the specified folder to store job and configuration data.

Deploy to the server
: Depending on the application server and your access rights you can deploy the WAR file via a web-based administration console or by copying the WAR into a deployment folder. The details of each servlet container and application server are beyond the scope of this book.

When using Hudson on your application server, you should ensure that the server is set up as an operating system service. The details of this setup widely vary between operating systems and application servers. The Hudson project provides helpful instructions for installing Hudson on Glassfish, WebSphere, JBoss, Jetty, Tomcat, Jonas, Weblogic and Winstone. More information about different containers and their specific needs in terms of installing and running Hudson is maintained on the Hudson wiki.

Note
The recommended installation process is to install Hudson as a stand-alone service on a dedicated host using the operating system-specific packages supplied by the Hudson web site and documented in the following sections. If you choose to run Hudson in an application server or a servlet container, you may need to perform complicated configuration changes to ensure that Hudson adheres to a particular servers expectations.

2.4 Installing Hudson on Ubuntu/Debian

Hudson provides a package repository of deb files for users of Debian based distributions such as Debian, Ubuntu and others. This package will install Hudson and set up the CI server as a service.

Step 1: Install Java runtime
In order to fulfil the prerequisite of an installed Java runtime on a Debian based distribution it is best to install the meta package `default-jdk`, which will install OpenJDK. Either use a graphical user interface like `synaptic` or install from the command-line with apt-get using the following command.

```
sudo apt-get install default-jdk
```

If you prefer to use the Oracle Java runtime install it with the following apt-get command.

```
sudo apt-get install sun-java6-jdk
```

Step 2: Add Hudson repository URL to package management
The Hudson project hosts its packages in its own repository server. In order to use it you have to add its URL to your list of package sources with the following command.

```
sudo sh -c "echo 'deb http://hudson-ci.org/debian binary/' \
> /etc/apt/sources.list.d/hudson.list"
```

You can also add the APT line `deb http://hudson-ci.org/debian binary/` in your GUI package manager as a repository URL. Future upgrades will not require this step to be repeated.

Step 3: Update the list of available packages

Once you've installed Java and added the Hudson repository URL you can update the list of available packages with the following command.

```
sudo apt-get update
```

This step has to be repeated whenever you want to check for the availability of upgrades. A common, best practice is to run `yum update` on a regular basis using cron this will alert you to updates as they become available.

Step 4: Install Hudson

Once your list of available packages is updated, you can install Hudson with the following command:

```
sudo apt-get install hudson
```

As the hudson packages are signed with a key that isn't trusted by default, the installation process requires your confirmation. Once you have verified that you would like to install Hudson without verification, the installation process will then proceed to install and start Hudson. Your console output will look similar to the following output.

```
$ sudo apt-get install hudson
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  hudson
0 upgraded, 1 newly installed, 0 to remove and 4 not upgraded.
Need to get 38.8 MB of archives.
After this operation, 39.7 MB of additional disk space will be used.
WARNING: The following packages cannot be authenticated!
  hudson
Install these packages without verification [y/N]? y
Get:1 http://hudson-ci.org/debian/ binary/ hudson 2.0.0 [38.8 MB]
Fetched 38.8 MB in 39s (981 kB/s)
Selecting previously deselected package hudson.
(Reading database ... 180192 files and directories currently installed.)
Unpacking hudson (from .../archives/hudson_2.0.0_all.deb) ...
Processing triggers for ureadahead ...
ureadahead will be reprofiled on next reboot
Setting up hudson (2.0.0) ...
Adding system user `hudson' (UID 114) ...
Adding new user `hudson' (UID 114) with group `nogroup' ...
Not creating home directory `/var/lib/hudson'.
 * Starting Hudson Continuous Integration Server hudson  [ OK ]
```

As you can see from the output above, a hudson user was created. This user will run the Hudson server. The Hudson home folder is configured to point to the `/var/lib/hudson` directory. This is an important directory to remember, as it contains configuration, work-spaces, and other files related to your project's builds.

Note

Take note, if you are backing up Hudson, you will want to configure your backup systems to archive the contents of `/var/lib/hudson`. Everything interesting that relates to your projects is contained in this directory.

Step 5: Upgrade Hudson

To upgrade Hudson when a new release is available you would run the following apt-get command.

```
sudo apt-get upgrade
```

This will stop the running Hudson server, upgrade Hudson and restart the server. Prior to upgrading you should always backup your Hudson configuration and workspace data located in `/var/lib/hudson`.

2.4.1 Hudson File-system on Ubuntu

Running `dpkg` provides a list of the files installed by the Hudson DEB package. This list is a helpful guide for Hudson administrators as it helps locate interesting configuration files and directories you will need to know about when configuring and troubleshooting a Hudson installation.

```
$ dpkg -L hudson
/.
/var
/var/log
/var/log/hudson
/var/lib
/var/lib/hudson
/var/run
/var/run/hudson
/usr
/usr/share
/usr/share/doc
/usr/share/doc/hudson
/usr/share/doc/hudson/changelog.gz
/usr/share/doc/hudson/copyright
/usr/share/hudson
/usr/share/hudson/hudson.war
/usr/bin
/usr/sbin
/etc
/etc/apt
/etc/apt/sources.list.d
/etc/apt/sources.list.d/hudson.list
/etc/default
/etc/default/hudson
```

```
/etc/init.d
/etc/init.d/hudson
/etc/logrotate.d
/etc/logrotate.d/hudson
```

2.4.2 Starting and Stopping Hudson on Ubuntu

When you installed the Hudson DEB package on Ubuntu the installation process configured Hudson as a service. You can stop and start the service with the following commands:

```
sudo service hudson stop
sudo service hudson start
```

Alternatively, you can use the following commands:

```
sudo /etc/init.d/hudson stop
sudo /etc/init.d/hudson start
```

2.4.3 Hudson Log Files on Ubuntu

Following the Linux Standard Base conventions Hudson creates its log file in `/var/log/hudson/hudson.log`. This log file will be rotated automatically in the same way your syslog files are rotated on a nightly schedule.

This log rotation schedule can be configured by altering the configuration in `/etc/logrotate.d/hudson`.

2.4.4 Hudson Configuration on Ubuntu

The list of files installed by the Hudson DEB package reveal that a configuration file `/etc/default/hudson` was created. This file contains a number of configuration parameters that you might want to adapt to your needs. These include:

JAVA_ARGS
Used to increase the memory allocation for Hudson

HTTP-PORT
Sets the Hudson port to a default of 8080.

HUDSON_HOME
Defines the default location of Hudson's working directory.

HUDSON_LOG
Defines the location of Hudson's log file.

MAXOPENFILES
Defaults to 8192. This is an important configuration value for larger Hudson installations. If you are running out of file handles you can increase this limit here.

AJP_PORT
This configuration parameter is disabled by default, but if you are planning on exposing your Hudson service via a web server using mod_jk or mod_proxy_ajp, you can configure the AJP port in this configuration file.

JAVA
Defaults to /usr/bin/java. If you have a custom installation of Java, you can point your Hudson instance at a specific executable here.

HUDSON_USER
The DEB installer creates "hudson" user. This value is configurable in case you need to run Hudson under a different user.

Tip
If you modify this file to suit your needs, you should add it to your backup strategy.

2.5 Installing Hudson on Redhat, CentOS, and Fedora

Oracle Linux, Redhat Enterprise Linux, CentOS and Fedora all use the same RPM package provided by the Hudson project. This package will install Hudson and set it up as a service.

STEP 1: Install Java runtime
In order to fulfil the prerequisite of an installed Java runtime on a Red Hat-based distribution it is best to install the meta package `java`, which will install OpenJDK, with your preferred package manager user interface. Either use a graphical user interface like `Add/Remove Software` or install from the command-line with the following command.

```
sudo yum install java
```

STEP 2: Add Hudson repository URL to package management
The Hudson project hosts its packages in its own repository server. In order to reference these packages you have to add the repository metadata to your list of package sources with the following command.

```
sudo wget -O /etc/yum.repos.d/hudson.repo http://hudson-ci.org/redhat/hudson.repo
```

Future upgrades will not require this step to be repeated.

STEP 3: Update the list of available packages
Once the prior steps are completed you can update the list of available packages in your graphical package manager or using the following yum command.

```
sudo yum check-update
```

This step has to be repeated whenever you want to check for the availability of upgrades. A suggested best practice is to run "yum check-update" on a regular basis and configure a system to notify you when your system is eligible for updates.

STEP 4: Install Hudson
Once your list of available packages is updated, you can install Hudson with

```
sudo yum install hudson
```

This command will require your confirmation. Once you've confirmed that you want to install Hudson, yum will then proceed to install and start Hudson. Your console output will look similar to the following output.

```
$ sudo yum install hudson
Loaded plugins: langpacks, presto, refresh-packagekit
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package hudson.noarch 0:2.0.1-1.1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

======================================================================
 Package        Arch            Version                Repository      Size
======================================================================
Installing:
 hudson         noarch          2.0.1-1.1              test            37 M

Transaction Summary
======================================================================
Install         1 Package(s)

Total download size: 37 M
```

```
Installed size: 37 M
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
Processing delta metadata
Package(s) data still to download: 37 M
hudson-2.0.1-1.1.noarch.100% [=======] 953 kB/s | 37 MB         00:40
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : hudson-2.0.1-1.1.noarch          1/1

Installed:
  hudson.noarch 0:2.0.1-1.1

Complete!
```

STEP 5: Upgrade Hudson

To upgrade Hudson when a new release is available run the following command:

```
sudo yum update
```

This command stops the running Hudson server, upgrade Hudson and restart the server. Prior to upgrading you might want to backup your Hudson data configuration located in `/var/lib/hudson` and owned by the hudson user.

2.5.1 Hudson File-system on Redhat

Looking at the list of files installed by the package we see that this RPM created the following files.

```
$ rpm -ql hudson
/etc/init.d/hudson
/etc/logrotate.d/hudson
/etc/sysconfig/hudson
/usr/lib/hudson
/usr/lib/hudson/hudson.war
/usr/sbin/hudson
/var/lib/hudson
/var/log/hudson
```

As part of the install a hudson user was created . This user will run the Hudson server. The Hudson home folder is configured to be located in `/var/lib/hudson`, which will contain configuration, work-spaces and so on and should be added to your backup strategy.

2.5.2 Starting and Stopping Hudson on Redhat

The install configured Hudson as a service so that you can stop and start the service with the following commands:

```
sudo service hudson stop
sudo service hudson start
```

2.5.3 Hudson Log Files on Redhat

Following the Linux standard base convention Hudson will create its log files into `/var/log/hudson/hudson.log` and the log files will be rotated on a nightly basis to preserve disk space.

2.5.4 Hudson Configuration on Redhat

This rpm command reveals that a configuration file `/etc/sysconfig/hudson` was created. It contains a number of configuration parameters that you might want to adapt to your needs. These include e.g. the `HUDSON_JAVA_OPTIONS` that can be used to increase the memory allocation for Hudson or the `HUDSON_PORT` parameter set to the common 8080. If you modify this file to suit your needs, you should add it to you backup strategy.

2.6 Installing Hudson on OpenSUSE

OpenSUSE uses a special rpm package provided by the Hudson project. This package will install Hudson and set it up as a service.

STEP 1: Install Java runtime
In order to fulfil the prerequisite of an installed Java runtime on openSUSE it is best to install the meta package `java`, which will install OpenJDK, with your preferred package manager user interface. Either use a graphical user interface like `YaST` or install from on the command-line with

```
sudo zypper install java
```

STEP 2: Add Hudson repository URL to package management
The Hudson project hosts its packages in its own repository server. In order to use it you have to add the repository meta data to your list of package sources with

```
sudo wget -O /etc/zypp/repos.d/hudson.repo http://hudson-ci.org/opensuse/hudson.repo
```

Future upgrades will not require this step to be repeated.

STEP 3: Update the list of available packages
Once the prior steps are completed you can update the list of available packages in your graphical package manager or with

```
sudo zypper refresh
```

This step has to be repeated whenever you want to check for the availability of upgrades. Common practice is for the update of the list to run automatically on a regular basis.

STEP 4: Install Hudson
Once your list of available packages is updated, you can install Hudson with

```
sudo zypper install hudson
```

This command requires you to confirm and will then proceed to install and start Hudson. Your console output will look similar to this

```
$ sudo zypper install hudson
Loading repository data...
Reading installed packages...
Resolving package dependencies...

The following NEW package is going to be installed:
  hudson

1 new package to install.
Overall download size: 37.1 MiB. After the operation, additional 37.1 MiB will be
used.
Continue? [y/n/?] (y):
Installing: hudson-2.0.1-1.1 [done]
Additional rpm output:
hudson                    0:off  1:off  2:off  3:on   4:off  5:on   6:off
```

STEP 5: Upgrade Hudson
To upgrade Hudson when a new release is available you would run the following command:

```
sudo zypper update
```

This command stops the running Hudson server, upgrade Hudson and restart the server. Prior to upgrading you might want to backup your Hudson data configuration located in `/var/lib/hudson` and owned by the hudson user.

The install configured Hudson as a service so that you can stop and start the service with the following commands:

```
sudo /etc/init.d/hudson stop
sudo /etc/init.d/hudson start
```

Following the Linux standard base convention Hudson will create its log files into `/var/log/hudson/hudson.log` and the log files will be rotated so you will no accumulate large log files using up disk space.

Looking at the list of files installed by the package

```
$ rpm -ql hudson
/etc/init.d/hudson
/etc/logrotate.d/hudson
/etc/sysconfig/hudson
/etc/zypp/repos.d/hudson.repo
/usr/lib/hudson
/usr/lib/hudson/hudson.war
/usr/sbin/rchudson
/var/lib/hudson
/var/log/hudson
```

This command reveals that a configuration file `/etc/sysconfig/hudson` was created. It contains a number of configuration parameters that you might want to adapt to your needs. These include e.g. the `HUDSON_JAVA_OPTIONS` that can be used to increase the memory allocation for Hudson or the `HUDSON_PORT` parameter set to the common 8080. If you modify this file to suit your needs, you should add it to you backup strategy.

As part of the install a hudson user was created . This user will run the Hudson server. The Hudson home folder is configured to be located in `/var/lib/hudson`, which will contain configuration, work-spaces and so on and should be added to your backup strategy.

2.7 Hudson Related Files and Directories

Depending on your installation method for Hudson using the standalone war, running it with an application server or using one of the native packages, all files required by Hudson will be located in different locations on the file system. In all cases however the main location for configuration of Hudson, plugins, builds and so on will be in your Hudson home directory.

It is defined as `HUDSON_HOME` and visible in the administration interface as Home directory in Figure 3.2. With a native package install this would typically be `/var/lib/hudson`. When running Hudson in an application server or from the standalone war it typically is the .hudson folder in the home folder of the user running the application server or war file.

In addition you installation will include the Hudson war file itself as well as scripts for service start-up and log rotation. These vary depending on your install method in location as well as content. The sections for the native installer include details for these files. For application server based install or running from the standalone war file you will have set up these individually files yourself and therefore know about their location.

The Hudson home directory itself contains a number of files and directories that we will examine now.

The main Hudson configuration is contained in `config.xml`. It contains large number of configuration settings for Hudson itself including security and authorization details, views, nodes, JDK's and other global parameters.

Further system-wide configuration specific to installed plugins are contained in numerous xml files, which represent a serialized form of the various plugin configuration settings. File names used vary widely, but include `hudon.tasks.*.xml`, `hudson.plugins.*.xml`, `hudson.scm.\*.xml` and others like `maven-installation`
or `rest-plugin.xml`.

Plugins themselves are installed subfolder of the `plugins` folder. It typically contains the plugins as `.hpi` archives as well as extracted in a folders using the plugin name.

Any build tools like Apache Ant, Apache Maven or a JDK installed by one of the Hudson installers will be installed their own directories in the `tools` folder.

The `fingerprints` folder contains the fingerprint records created by build artifacts created and tracked by Hudson.

The `users` folder contains subfolders for each user of Hudson or committer on the projects built in a `config.xml` file, that contains email configuration as well as configured views and other user specific data.

The `userContent` folder can contain static content that Hudson will serve in the `userContent` context. It can be used to make e.g. software installation packages or web site content available to Hudson users without requiring a separate web server.

Beyond all the global configuration described so far all the job related configuration and data is stored in individual subfolder of the `jobs` folder. Each project has a folder using the project name as configured in Section 6.2.1. The `config.xml` file within this folder contains all project specific configuration parameters. The `workspace` contains the project as currently checked out from the used SCM. If the Maven 3 integration is set to use a private repository, it will be located in the `.maven/repo` folder in the workspace. The individual build results, logs and artifacts are stored in directories with date and time stamps as folder names within the `builds` folder. In addition symlinks with build numbers are created to point to the time stamped folders.

2.8 Backing up Hudson Data

A full backup would be the complete content of your Hudson home directory as explained in Section 2.7 and any customized files for service start-up and log rotation.

This folder can also be used to migrated your Hudson instance to a different server by simply copying it over after installing Hudson on the new machine.

For a limited backup you could e.g. omit some of the build history of various jobs or potentially large Maven repositories for the various builds.

Tip
The JobConfigHistory Plugin, the thinBAckup Plugin as well as the Backup Plugin can be used to simplify your backup strategy, if you want to reduce the need for external tools. Some users are using a version control system like Git or Subversion to track changes of the Hudson home directory. You could even use a Hudson job to e.g. do regularly scheduled commits of config changes.

2.9 Upgrading Hudson

Since Hudson separates it's configuration and data storage from the application, it is easy to upgrade an existing Hudson installation. After a full backup of all configuration and data, you should be ready to proceed after notifying your users about potential down-time. You might also want to disable all jobs before proceeding.

For native package users this will be handled transparently with their package management system. For WAR file based installs, you only have to remove the old version WAR and replace it with the new version in your application server.

Typically upgrading Hudson should include upgrading any used plugins as well to avoid incompatibilities.

2.10 Running Hudson Behind a Proxy

If you installed Hudson as a stand-alone application, Hudson is running on a high-performance servlet. From a performance perspective, there is no reason for you not to run Hudson by itself without a proxy. Yet, more often than not, organizations run applications behind a proxy for security concerns and to consolidate applications using tools like `mod_rewrite` and `mod_proxy`. For this reason, we've included some brief instructions for configuring Apache HTTP. We assume that you've already installed Apache 2, and that you are using a Virtual Host for www.example.com.

Let's assume that you wanted to host Hudson behind Apache HTTP at the URL http://www.example.com. To do this, you'll need to change the context path that Hudson is served from.

1. Need to explain how to run Hudson in the root context here
2. Restart Hudson and verify that it is available on http://localhost:8080/.
3. Clear the Base URL in Hudson Application Server Settings in the administration interface.

At this point, edit the HTTP configuration file for the www.example.com virtual host. Include the following to expose Hudson via `mod_proxy` at http://www.example.com/.

```
ProxyRequests Off
ProxyPreserveHost On

<VirtualHost *:80>
  ServerName www.example.com
  ServerAdmin admin@example.com
  ProxyPass / http://localhost:8080/
  ProxyPassReverse / http://localhost:8080/
  ErrorLog logs/example/hudson/error.log
  CustomLog logs/example/hudson/access.log common
</VirtualHost>
```

If you just wanted to continue to serve Hudson at the */hudson* context path, you would include the context path in your `ProxyPass` and `ProxyPassReverse` directives as follows:

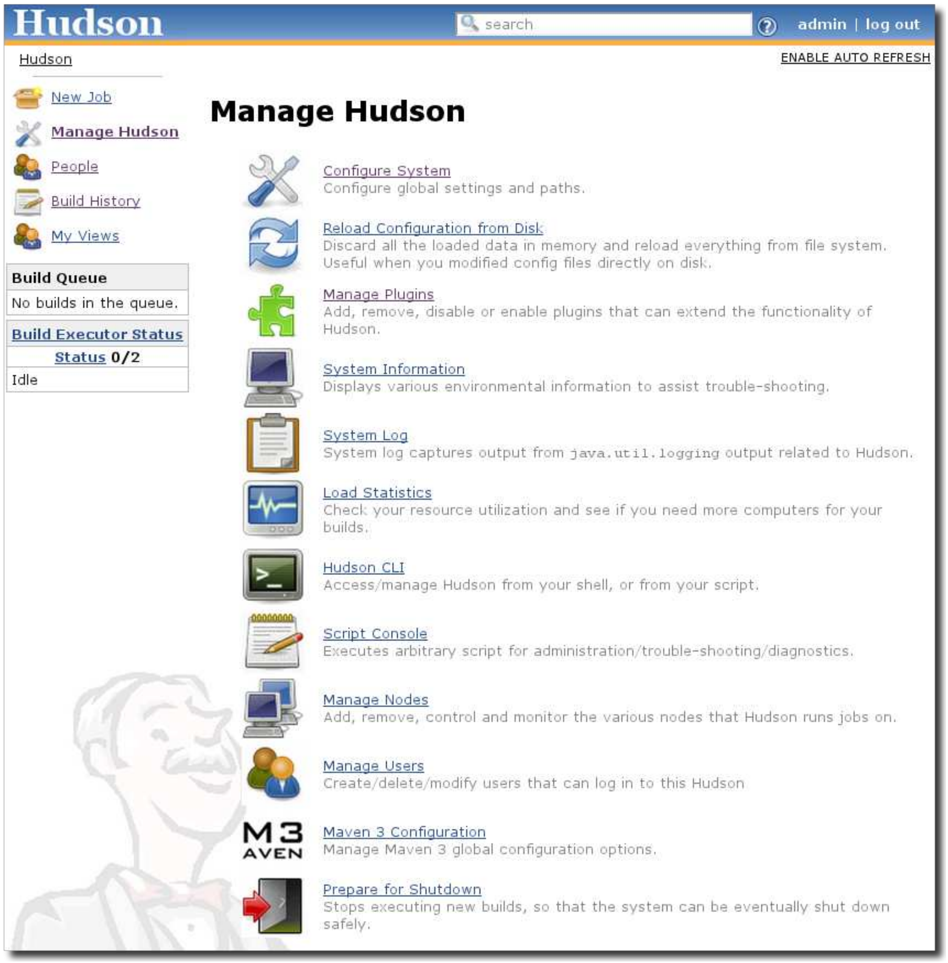
```
ProxyPass /matrix/ http://localhost:8082/matrix/
ProxyPassReverse /matrix/ http://localhost:8082/matrix/
```

Apache configuration is going to vary based on your own application's requirements and the way you intend to expose Hudson to the outside world. If you need more details about Apache httpd and `mod_proxy`, please read the documentation on the project website.

Chapter 3

Hudson Configuration

To configure Hudson, click on the Manage Hudson link in the left-hand navigation menu, which will display the screen shown in Figure 3.1. This chapter will focus on the Configure System section.



Hudson | search | admin | log out

Hudson | ENABLE AUTO REFRESH

New Job
Manage Hudson
People
Build History
My Views

Build Queue
No builds in the queue.

Build Executor Status
Status 0/2
Idle

Manage Hudson

Configure System
Configure global settings and paths.

Reload Configuration from Disk
Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Hudson.

System Information
Displays various environmental information to assist trouble-shooting.

System Log
System log captures output from `java.util.logging` output related to Hudson.

Load Statistics
Check your resource utilization and see if you need more computers for your builds.

Hudson CLI
Access/manage Hudson from your shell, or from your script.

Script Console
Executes arbitrary script for administration/trouble-shooting/diagnostics.

Manage Nodes
Add, remove, control and monitor the various nodes that Hudson runs jobs on.

Manage Users
Create/delete/modify users that can log in to this Hudson

Maven 3 Configuration
Manage Maven 3 global configuration options.

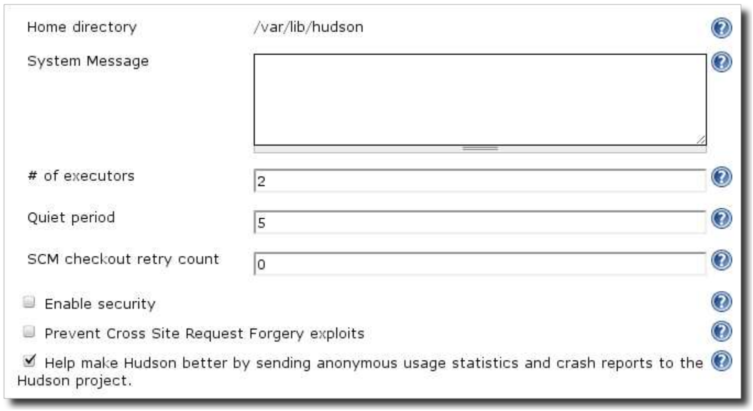
Prepare for Shutdown
Stops executing new builds, so that the system can be eventually shut down safely.

Figure 3.1: Managing Hudson

Depending on the plugins installed and activated on your Hudson system, different sections will be available in the system configuration section. These will either be explained below or with a plugin-specific section. For example the source code management-related global configurations for the different SCM systems is available in Chapter 9.

3.1 Global Hudson Configuration

The first section in the Configure System screen contains options that allow you to configure global Hudson configuration attributes. This section is shown in Figure 3.2.



Home directory /var/lib/hudson

System Message

# of executors 2

Quiet period 5

SCM checkout retry count 0

☐ Enable security

☐ Prevent Cross Site Request Forgery exploits

☑ Help make Hudson better by sending anonymous usage statistics and crash reports to the Hudson project.

Figure 3.2: Configuring Global Hudson Configuration

Home Directory
This parameter displays the absolute installation path of the currently running Hudson system. It is not a runtime configurable parameter. It is set by the server on start-up. By default it will be the value of the HUDSON_HOME environment variable or the `.hudson` folder in the home directory of the operating system user running Hudson. The value is displayed here to allow the administrator to verify the correct setting.

System Message
This message is displayed by Hudson in the main screen above the list of projects. It can be used as a welcome message or to e.g. broadcast upcoming maintenance to users of the Hudson instance via the user interface. It supports plain text as well as HTML snippets for formatting and enriching the message with dynamic content.

# of Executors
This parameter controls the number of concurrent builds Hudson is configured to run. Optimal values in terms of performance will depend on the number of CPUs, IO performance and other hardware characteristics of the server running Hudson as well as the type of builds configured to run. A good starting point for experimentation is the number of CPUs.

Quiet period
A Quiet Period as specified in this configuration causes Hudson to wait the specified number of seconds before a triggered build is started. If your Hudson project is constantly "flapping" (switching between failure and success frequently), you may want to set the Quiet period to achieve more build stability. Another scenario this can be helpful is when large commits to your source control system are typically carried out in multiple smaller commits within a short time frame, so that the committer has a chance to get everything in without a build kicking off straight after the first commit causing a build failure. Setting this number to a large amount can generally reduce the number of builds running for this project, which will reduce the overall load for your Hudson server.

SCM checkout retry count
The SCM checkout retry count determines the number of attempts Hudson makes to check out any updates when polling the SCM system for changes and finds the system to be unavailable.

Enable Security
The Enable Security check box switches on the security system that will require user-name and password for any access to run builds or change configurations of Hudson and build projects. A large number of configuration options and security providers can be used and more information can be found in Chapter 4

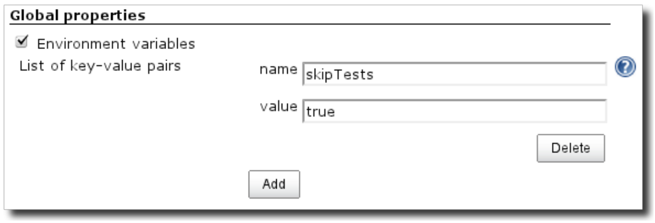
Prevent Cross Site Request Forgery exploits
This feature will enable improved security against Cross Site Request Forgery exploits and is recommended to be turned on when your Hudson instance is available to the public Internet. On the other hand it can be necessary to have this feature disabled, when your Hudson web interface is embedded in a dashboard type interface that also contains web content from other domains or even only internal server names or sub networks. In general it will not be necessary to enable this feature on an internal network, where only trusted parties have access to Hudson.

Help make Hudson better…
By selecting this feature to be enabled you agree for anonymous usage statistics about your Hudson installation to be created and securely sent to the Hudson development team and made available to the user community. The data sent consists of

- the Hudson version you are using
- operating system, JVM and number of executors for your master Hudson and any slaves being used
- the name and version of all activated plugins
- the number of each project type configured to run
- HTTP information as provided by your Hudson instance

3.2 Global Properties Configuration

The global properties configuration allows the definition of key-value pairs that are exposed to all running builds as environment variables. Simply select the check box Environment Variables and add the desired name and value for the property in the interface displayed in Figure 3.3.


Global properties

☑ Environment variables

List of key-value pairs

name: skipTests

value: true

Figure 3.3: Configuring global properties

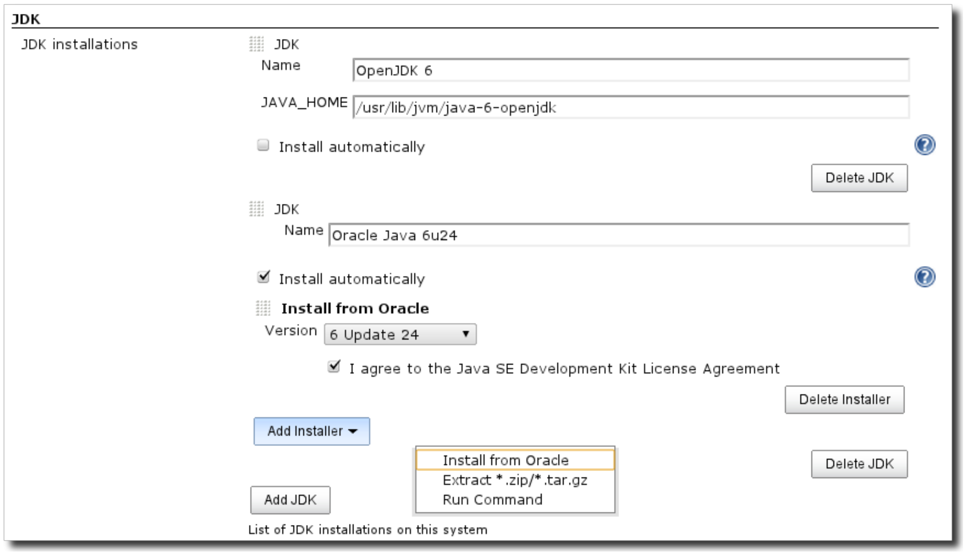
Depending on the build system used they can be picked up with variables are populated by Hudson automatically. These include job-related ones like `JOB_NAME`, `BUILD_TAG` or `BUILD_NUMBER`, Hudson node-related ones like

NODE_NAME or more global ones like JAVA_HOME. A comprehensive list is available at env-vars.html on your Hudson server e.g. http://localhost:8080/env-vars.html as linked from the in-line help for the properties configuration.

3.3 Configuring JDK Installations

Hudson can support one or more JDK installations used for running your builds. Setting up multiple JDK installs allows the configuration of different projects being built by different Java versions in separate jobs. You can use this to ensure e.g. that builds as well as test suites run fine on an older Java version to ensure compatibility. Another application would be to run with JDK versions supplied by different vendors.

The most common configuration of a JDK is to point to the already installed instance as used for running Hudson itself. This can be achieved simply by supplying a name like `Open JDK 6` in the Name input field and the absolute path in the JAVA_HOME input field in the screen as shown in Figure 3.4.



JDK

JDK installations

JDK
Name: OpenJDK 6
JAVA_HOME: /usr/lib/jvm/java-6-openjdk
☐ Install automatically

JDK
Name: Oracle Java 6u24
☑ Install automatically
Install from Oracle
Version: 6 Update 24
☑ I agree to the Java SE Development Kit License Agreement

List of JDK installations on this system

Figure 3.4: Configuring JDK Installations

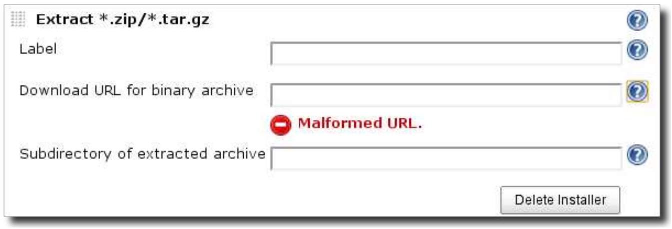
Furthermore it is possible to configure a JDK to be installed automatically by specifying a name as before and then selecting the Install Automatically check box. This exposes a drop-down labelled Add installer which lets you choose from the options `Install from Oracle`, `Extract *.zip/*.tar.gz` and `Run Command` as visible in Figure 3.4.

Warning
The automatic installation from Oracle is currently disabled, while Oracle implements a web service for JDK installation.

All the automatic install configurations cause Hudson to wait for the first build, which is configured to use a named instance of the JDK to initiate the JDK installation. The JDK will be installed into a folder in the tools directory in Hudson home using the tool name specified as the folder name.

The option Install from Oracle brings up a drop-down to choose the version as well as a check box that needs to be clicked to the Java SE license agreement.

If you select to use `Extract *.zip/*.tar.gz` as shown in Figure 3.5 you will be able to configure a Label, the Download URL for binary archive and the Subdirectory of extracted archive.



Extract *.zip/*.tar.gz

Label

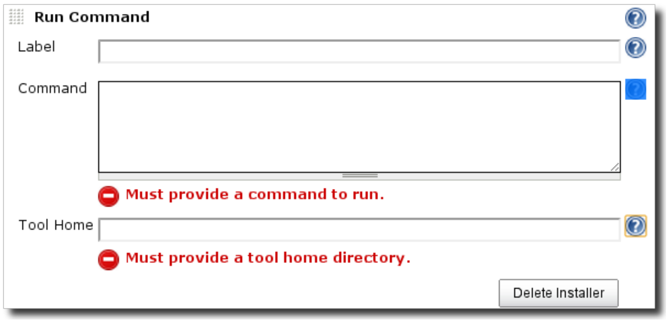
Download URL for binary archive

Malformed URL.

Subdirectory of extracted archive

Figure 3.5: `Extract *zip/*.tar.gz` archive Installer Configuration

If you specify a label, only Hudson nodes with the same label will use this installer. By using different labels it is possible for example to get the same tool installed on different nodes with different operating systems from different automatic install setups. The download URL specifies the full URL from which the JDK will be downloaded. The actual download is run off the Hudson master, so that any Hudson nodes that need the JDK installed do not need to have access to the URL location. After successful download the JDK will be installed in the specified sub directory in the tools folder of the Hudson home directory.



Must provide a command to run.

Must provide a tool home directory.

Figure 3.6: Run Command Installer Configuration

The last automatic installation option is the Run a command option displayed in Figure 3.6. The Label options works the same as for the archive extraction based install. The Command input allows you to specify the shell command to execute on the node for the install. Typically this is some package management invocation. The resulting tool directory has to be specified in the Tool Home input box.

Once more than one JDK is configured in the global settings, each project configuration has an additional drop-down, which allows the selection of the JDK to be used to build the project and is visible in Figure 3.5

3.4 Configuring Ant Installations

In a similar fashion to the JDK install Apache Ant can be installed in multiple versions to be available for your Hudson configured builds. The default configuration is to supply a name like `Apache Ant 1.8.2` for the Ant installation and a value in the ANT_HOME input that is defined by the absolute path to the folder containing your pre-existing local Apache Ant install e.g. `/opt/apache-ant-1.8.2`

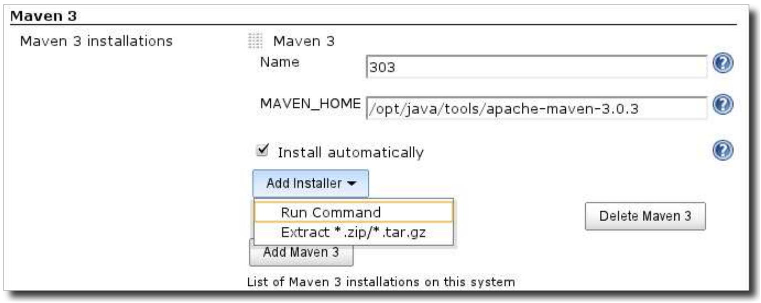
Using a pre-installed Ant requires manual install or the use of your operating system package management system, a provisioning system or as part of a virtual machine image management. To avoid this need Hudson can install a required Apache Ant version automatically when needed.

The simplest way to achieve this is to select the Install automatically check box and select Install from Apache and choose the desired version from the drop-down.

Similar to the JDK installation from Oracle it is possible to use Install from Apache to get Ant installed into a sub-directory of the tools folder in Hudson home. The options to install from an archive or by running a command are available as well and work in the same way as for JDK installs. A use case for an install from a file would be a custom Ant distribution with libraries for in-house tasks and maybe Ant contrib included as documented in detail in Chapter 8

3.5 Configuring Maven Installations

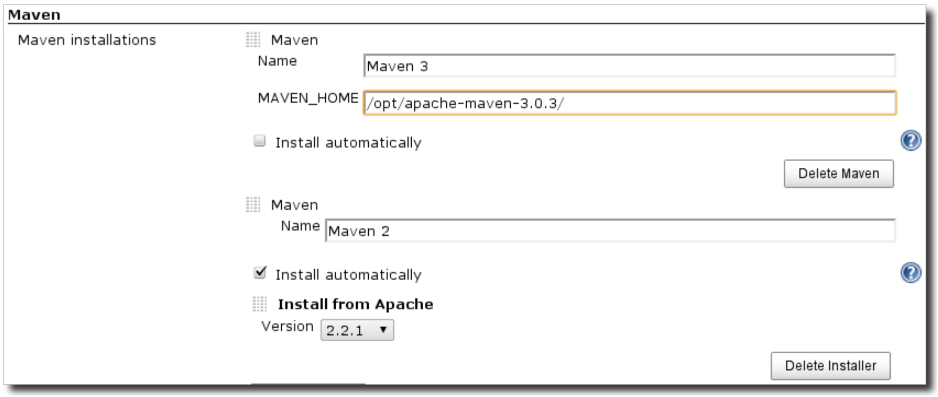
One of the main uses cases for Hudson is building projects with Apache Maven. As explained in more detail in Chapter 7 the preferred way to build Maven projects is the Maven 3 integration. It comes with a bundled Maven 3 install so you do not actually need to install Maven 3 at all to get started. However if you want, you can install additional Maven 3 installs with the user interface displayed in Figure 3.7.


Maven 3
Maven 3 installations
Maven 3
Name: 303
MAVEN_HOME: /opt/java/tools/apache-maven-3.0.3
☑ Install automatically

List of Maven 3 installations on this system

Figure 3.7: Configuring Maven 3 Installations

The legacy Maven project type and Maven plugin use a separate installer as displayed in Figure 3.8. In addition to the same features as the Maven 3 installer it can be configured to download a Maven version from the Apache web site when required.


Maven
Maven installations
Maven
Name: Maven 3
MAVEN_HOME: /opt/apache-maven-3.0.3/
☐ Install automatically

Maven
Name: Maven 2
☑ Install automatically
Install from Apache
Version: 2.2.1

Figure 3.8: Configuring Maven Installations

Both the Maven 3 and the Maven installation work in a similar way to the JDK and Ant installation options:

Use an existing installation
Specify a Name and add the path to your Maven install in the MAVEN_HOME input control.

Automatically install from Apache
Select the Install automatically check box and Install from Apache in the drop-down and choose the Maven version, you wish to install. This option is only available for the Maven installation.

Automatically install from an archive file
Select the Install automatically check box and `Extract *zip/*.tar.gz` and configure the installation as documented in Section 3.3

Automatically install via a command
Select the Install automatically check box and Run Command and configure the installation as documented in Section 3.3.

In general we recommend that you run your build using the latest Maven 3.0 release. With multiple Maven installations configured a drop down in the project build step configuration will allow you to choose the desired Maven version. Chapter 7 provides an in-depth documentation for using Maven with Hudson.

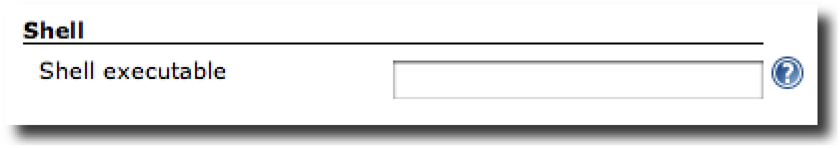
3.6 Maven 3 Builder Defaults

The Maven 3 integration allows for a set of default values to be defined that are used when a new build step for invoking Maven 3 is added. These values can be defined in the section Maven 3 Builder Defaults in the global Hudson configuration here. The individual fields and their purpose and usage are documented in Section 7.3.

3.7 Configuring the Shell Executable

Hudson allows for the ability to configure shell builds. If you have a build that requires the execution of shell scripts Hudson will by default execute+/bin/sh+. For more complex builds scripts running on different *nix environments, this can cause problems. `/bin/sh` often symlinks to a concrete shell like bash, ash, zsh or ksh. This setup of a specific shell will change from operating system to operating system as well as from user to user. If your scripts depend on a specific shell you should therefore specify your default shell in this input to e.g. `/bin/bash`. In a similar way you can add the path to a cygwin install of e.g. bash on your Windows server to run unix scripts as part of your build.

Configuring Shell Executable

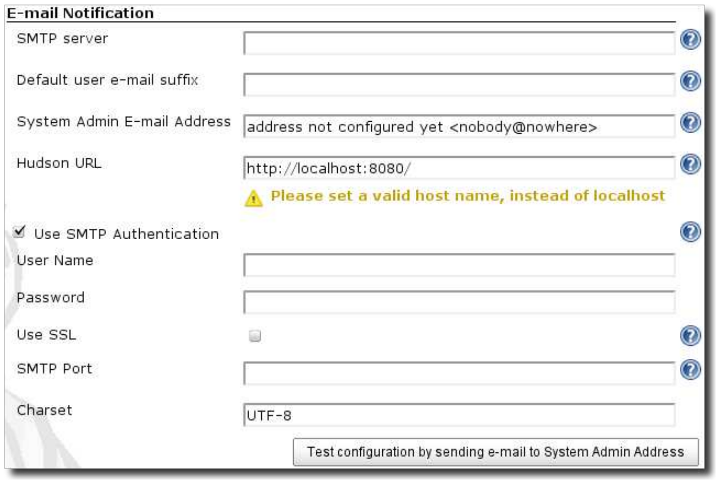


3.8 Configuring E-mail Notification

Notification of build results and email-based notification specifically is a core feature of a continuous integration server. This configuration section as displayed in Configuring Email Notification allows you to configure the SMTP-related settings to connect to the server and send the emails.

Tip
In general it can be advantageous to configure all email recipients in Hudson as mailing list addresses. Combined with a mailing list management system available to your potential recipients e.g. development and QA team members, this setup allows users to join any mailing list and therefore notifications for specific jobs without any configuration changes on Hudson.

Configuring Email Notification


E-mail Notification

SMTP server	
Default user e-mail suffix	
System Admin E-mail Address	address not configured yet <nobody@nowhere>
Hudson URL	http://localhost:8080/
	Please set a valid host name, instead of localhost
✓ Use SMTP Authentication	
User Name	
Password	
Use SSL	☐
SMTP Port	
Charset	UTF-8

Test configuration by sending e-mail to System Admin Address

The following options can be configured:

SMTP Server

The SMTP server configuration is typically the IP number of the mail server or a fully qualified name including the domain e.g. `smtp.example.com` . If the mail server is reachable by host name or some alias e.g. `hermes` from the Hudson server you can use it as the SMTP server configuration.

Default user e-mail suffix
This suffix is appended to the Hudson user names used to log in to Hudson and the result can be used for e-mail notification. E.g. if the Hudson instance runs for `example.com` you could supply the suffix of `@example.com`. A Hudson user with user name `jane.doe` would then receive email notifications at the email `jane.doe@example.com`. This can be especially useful with security setups using an identity management system like LDAP for Hudson access as well as email address setup as documented in Chapter 4.

System Admin E-Mail Address
This is the email address used as the email sender in any E-mail notification sent by the server. When configuring this email you should either ensure that emails sent back as a reply are monitored by somebody or bounced by the server with some meaningful error message.

Hudson URL
The Hudson URL value will be used in the email notifications sent out to provide links to build results and so on. Provide a URL that will be valid for the audience of your notifications. If all recipients will be on an intranet or VPN you can use a non-public URL or IP number.

In addition to basic SMTP configuration parameters, you can click the Advanced Options button for further configuration that allows you to send email via servers that require authentication. Most SMTP servers will require at least user name and password to be accessed.

Use SMTP Authentication
Clicking on the check box will reveal User Name and Password input fields. Depending on the server configuration your user name will be just the log-in name or the full email address or either.

Use SSL
Select this check box if your SMTP server supports connecting with SSL activated.

SMTP Port
This configuration allows you to specify a custom port for the communication with mail server. If the field is left empty the default ports are used. These are 25 for SMTP and 465 for SSL secured SMTP. It is a common practice to configure a different port, so be sure to check with the administrator of the mail server what port you should be using.

Charset
The Charset configuration determines the character set used for the composed e-mail message.

Test configuration
Pressing this button will execute the current configuration for sending emails. Depending on your configuration and network setup you should receive an email after a short while.

3.8.1 E-mail Notification Via Gmail

In order to use GMail to send your emails you will need to configure the SMTP server to `smtp.gmail.com`. In addition you will have to have a Gmail account and provide the GMail e-mail address, or any other email address configured to be accepted in your Gmail account, as the User Name and configure the Password.

3.9 Troubleshooting E-mail Notification

3.9.1 Spam filter related problems

One of the common problems for build server notification emails not being received are spam filter settings on the server and/or client side of the recipient. Most spam filter systems will allow you to access a list of filtered message and configure a white list of senders. Adding the System Admin E-mail Address to the white list will ensure that your build notifications reach you.

3.10 Managing Maven 3 Configuration

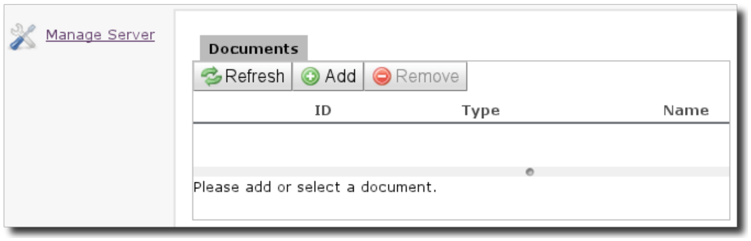
The Maven 3 integration of Hudson provides you with the ability to manage custom Maven configuration files directly through the Hudson user interface. You can manage:

- Maven Settings Configuration
- Maven Toolchains Configuration

3.10.1 Opening the Maven 3 Configuration Page

To open the Maven 3 Configuration page, click on Manage Server in the left-hand Hudson menu, and then select the Maven 3 Configuration item shown in Figure 3.1.

Once you select the Maven 3 Configuration option, you will see the page shown in Figure 3.9. If you have already configured Maven Settings or Maven Toolchains configuration documents they will appear in the list of documents shown on this page. If you have not configured any Maven configuration documents, you will see the empty configuration screen shown in Figure 3.9.



Please add or select a document.

Figure 3.9: The Maven 3 Configuration Page

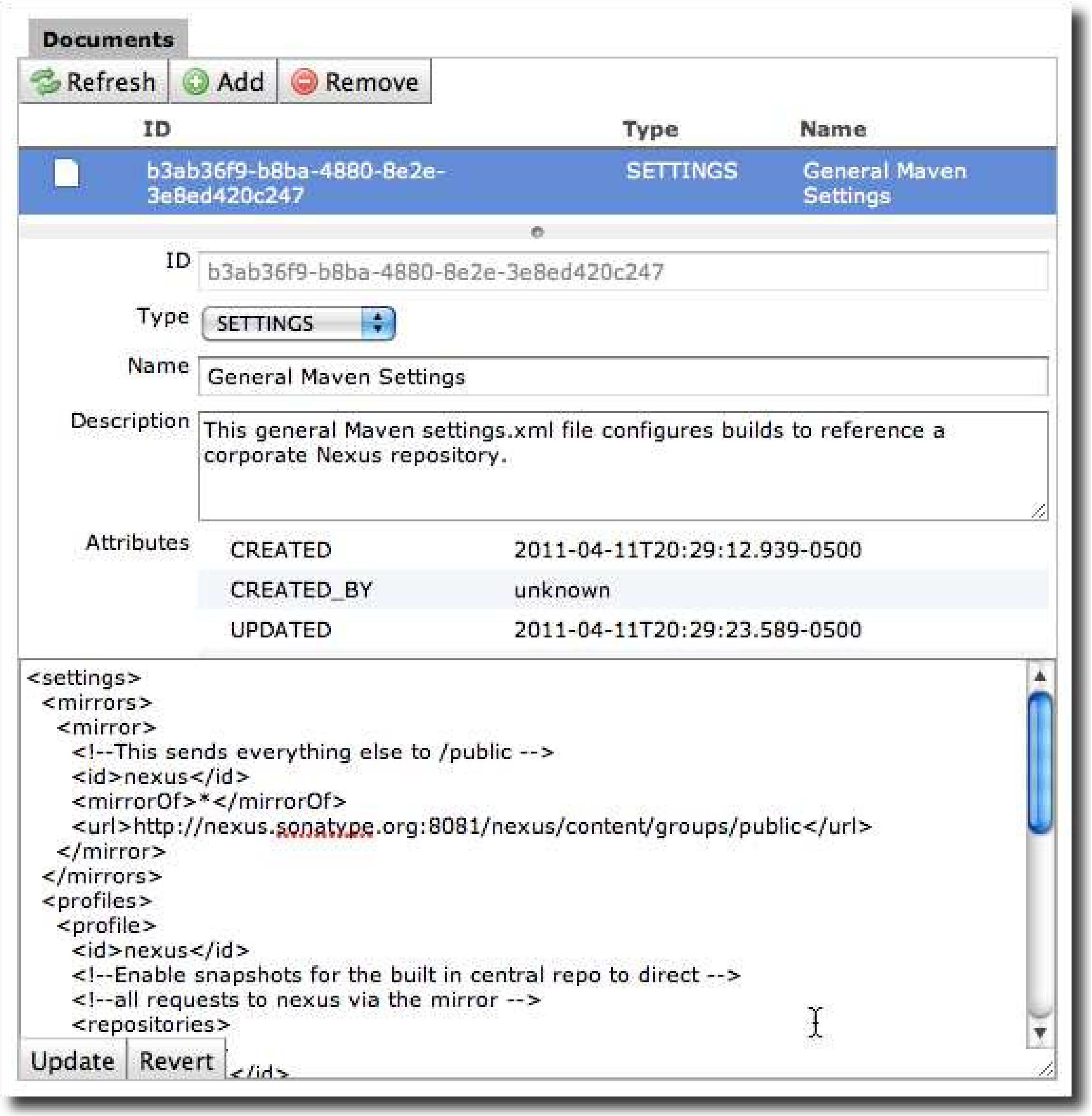
To create a new Maven 3 Configuration document, click on the Add button. This will create a new configuration document and display a form that will allow you to name the document, describe the document, select a document type, and supply configuration content for a configuration document.

To remove an existing document, select the document from the list of documents shown and click on the Remove button. This will load a confirmation dialogue. If the action is confirmed, the document will be permanently removed from your Hudson instance.

Click the Refresh button in the interface to reload the Hudson configuration and display any configuration documents which may have been altered since you first loaded this page.

3.10.2 Managing Maven 3 Settings Configuration

To create a new Maven 3 Setting configuration file which can be referenced by a Hudson Maven 3 build step, click on the Add button as shown in Figure 3.9. Clicking on Add will display a form containing the ID, Type, Name, Description, and Attribute fields as shown in Figure 3.10. Select SETTINGS for the Type field.



ID	Type	Name
b3ab36f9-b8ba-4880-8e2e-3e8ed420c247	SETTINGS	General Maven Settings

ID: b3ab36f9-b8ba-4880-8e2e-3e8ed420c247

Type: SETTINGS

Name: General Maven Settings

Description: This general Maven settings.xml file configures builds to reference a corporate Nexus repository.

Attributes

CREATED	2011-04-11T20:29:12.939-0500
CREATED_BY	unknown
UPDATED	2011-04-11T20:29:23.589-0500

```
<settings>
  <mirrors>
    <mirror>
      <!--This sends everything else to /public -->
      <id>nexus</id>
      <mirrorOf>*</mirrorOf>
      <url>http://nexus.sonatype.org:8081/nexus/content/groups/public</url>
    </mirror>
  </mirrors>
  <profiles>
    <profile>
      <id>nexus</id>
      <!--Enable snapshots for the built in central repo to direct -->
      <!--all requests to nexus via the mirror -->
      <repositories>
        </id>
```

Figure 3.10: Managing Maven 3 Settings in Hudson

The sample Maven 3 Settings configuration shown in Figure 3.10 define a General Maven 3 Settings file which configures all Maven 3 builds to read artifacts from a corporate Nexus repository. This sample XML was copied from the Maven 3 Settings example in the Sonatype Nexus book and customized to reference a hypothetical server running on `nexus.sonatype.org:8081`.

Usage of the Maven 3 Settings configuration file in a Hudson job as well as Maven toolchains is documented in Section 7.3.

3.11 Configuring Global and Individual Project List Views

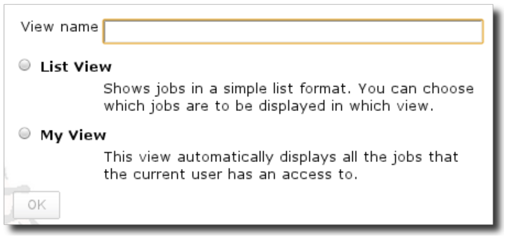
Running a Hudson server or grid with many projects can make the list of configured jobs very long and therefore make it difficult for users and administrators to gather the information they need at a glance and find their way to the job they are interested in. To provide this needed ease of access Hudson supports the addition of globally visible as well as user specific views to the main page displayed in Figure 2.1.

To add a new view simply click the + button beside the default list called All on top of the job list. If this button is not visible you do not have the access right to create views. Creating a view this way will make it globally visible to any user.

Logging in as a specific user shows the My Views option in the left hand navigation menu. When you select this option the bread crumb navigation will contain your user name and the My Views option. Pressing the + button on the main job list now will create a personal view only visible to the current user when logged in.

Each view can be given a description by clicking the edit description on the right side above the job list.

When adding a new global view as displayed in Figure 3.11 you have to provide a name and then select what type of view you want to add. You can either add a My View, which will display all jobs the current users has access to or you can add a List View, which is a highly configurable list of job types. The different configuration options are documented below.

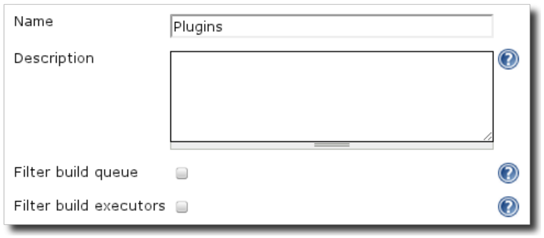


Shows jobs in a simple list format. You can choose which jobs are to be displayed in which view.

This view automatically displays all the jobs that the current user has an access to.

Figure 3.11: Adding a New Global View

After you have created the view you can configure the main filter properties displayed in Figure 3.12, configure, add and delete job filters as displayed in Figure 3.13 and add and delete columns for the list view as displayed in Figure 3.23.


Name: Plugins

Description

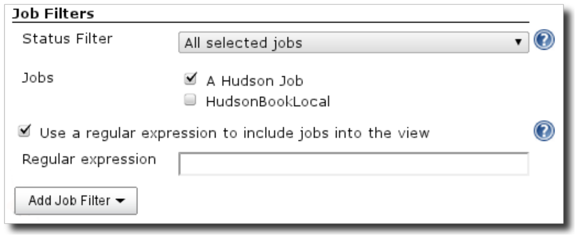
Filter build queue ☐

Filter build executors ☐

Figure 3.12: Main Properties for Adding a New View

Besides the name provided at creation you can add a description that will be displayed above the list of jobs and below the title of the view. It can use HTML for e.g. hyperlinks to other related resources. The Filter build queue option causes Hudson to only show jobs from the view in the build queue. The Filter build executors option causes Hudson to only show build executors that could execute shows from this view.

The Job Filters configuration below as visible in Figure 3.13 shows the Status Filter, a list of all jobs and allows the configuration of a regular expression for inclusion of jobs on the list view. The regular expression is applied to the job name.


Job Filters

Status Filter: All selected jobs

Jobs: ✓ A Hudson Job; ☐ HudsonBookLocal

✓ Use a regular expression to include jobs into the view

Regular expression

Figure 3.13: Configuring View Filters

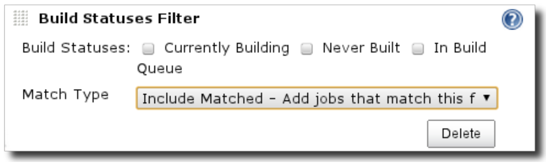
The Status Filter has the option to use all selected jobs or only enabled or disabled builds from the list below or the result of the regular expression configured. The Jobs list itself is an alphabetically sorted list of all jobs configured with a check box for each job that you potentially want to include in the view. As an alternative to manually selecting jobs from the list you can configure a regular expression to match on the job names. With a good naming convention for the jobs this can be a good way to build the list. An example could be naming all release build jobs x-y-z-release or naming all plugin builds Plugin-x and then using a regular expression like `.*-release` or `Plugin-.*` to select them.

In order to have access to more powerful configuration options for your list views you have to install the View Job Filters plugin. It will hook into the Job Filters section creating the Add Job Filter button and providing the filters documented in the following.

The filters will be applied in the order in the configuration screen. Each filter has a Match Type drop down selector that allows you to include matched or unmatched jobs as well as exclude matched and unmatched jobs from further filtering.

A myriad of combinations is possible allowing you to implement easy access to all your builds for all your users catered to their varying needs

With the Build Statuses filter displayed in Figure 3.14 you can include or exclude builds that are currently building, that have never been built or that are currently in the build queue.

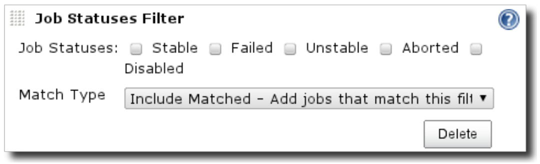

Build Statuses Filter

Build Statuses: ☐ Currently Building ☐ Never Built ☐ In Build Queue

Match Type: Include Matched - Add jobs that match this f

Figure 3.14: Configuring the Build Status View Filter

The Job Statuses filter in Figure 3.15 allows you to combine Stable, Failed, Unstable, Aborted and Disabled statuses to create a filter for including or excluding jobs from your view.

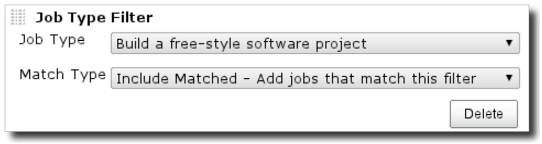

Job Statuses Filter

Job Statuses: ☐ Stable ☐ Failed ☐ Unstable ☐ Aborted ☐ Disabled

Match Type: Include Matched - Add jobs that match this filt

Figure 3.15: Configuring the Job Status View Filter

Similarly the Job Type filter in Figure 3.16 can be used to include or exclude based on the job being a free-style project, a Maven2/3 project, a multi-configuration project or a monitor of an external job.

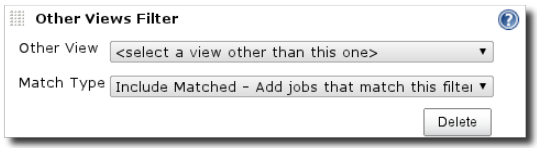

Job Type Filter

Job Type: Build a free-style software project

Match Type: Include Matched - Add jobs that match this filter

Figure 3.16: Configuring the Job Type View Filter

The Other Views filter in Figure 3.17 has a selector for the other view which can be used for including or excluding jobs to the current view being configured. This allows you to easily create negations of other views e.g. all non release builds, or all non plugin builds or simply create a finer grained view of a different view. E.g. you could easily create a view of all plugin builds that are currently with the job status Failed using a combination of the Other Views and the Job Statuses filter. Another good use case for this filter is to use multiple Other Views filters to aggregate the content of a number of other views.

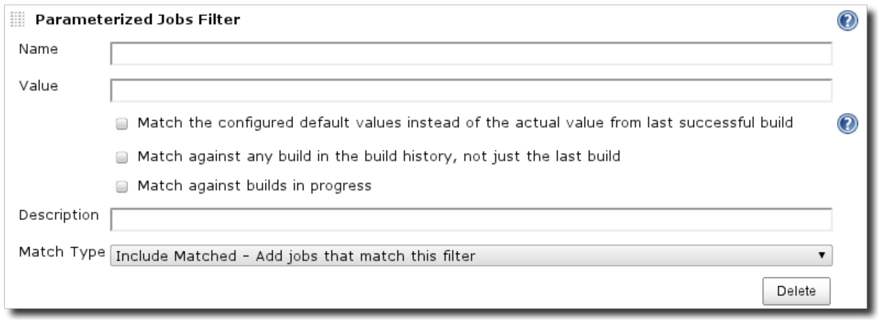

Other Views Filter

Other View: <select a view other than this one>

Match Type: Include Matched - Add jobs that match this filter

Figure 3.17: Configuring the Other Views Filter

The Parameterized Jobs Filter in Figure 3.18 can be used to define a Name and Value that will match against the build parameters.


Parameterized Jobs Filter

Name:

Value:

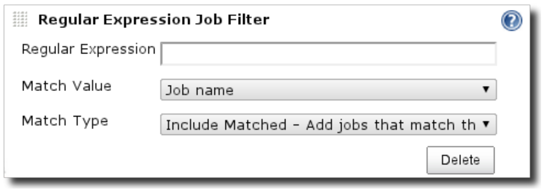
- Match the configured default values instead of the actual value from last successful build
- Match against any build in the build history, not just the last build
- Match against builds in progress

Description:

Match Type: Include Matched - Add jobs that match this filter

Figure 3.18: Configuring the Parameterized Jobs View Filter

The Regular Expression Job Filter displayed in Figure 3.19 can be used to run a pattern matching against Hudson job name, job description, job SCM configuration, email recipients or Maven configuration.


Regular Expression Job Filter

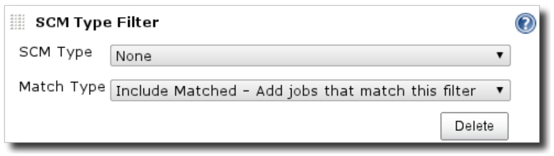
Regular Expression:

Match Value: Job name

Match Type: Include Matched - Add jobs that match th

Figure 3.19: Configuring the Regular Expression Job View Filter

The SCM Type filter allows you too filter based on the SCM used for your builds.

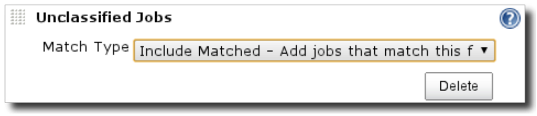

SCM Type Filter

SCM Type: None

Match Type: Include Matched - Add jobs that match this filter

Figure 3.20: Configuring the SCM Type View Filter

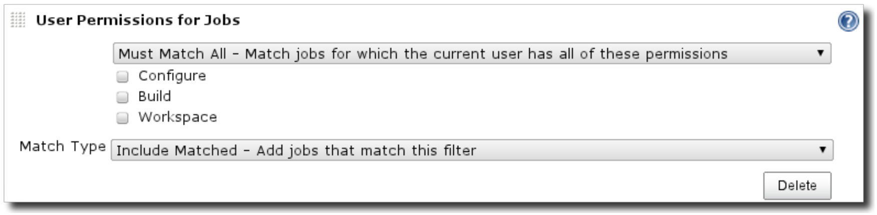
The Unclassified Jobs view filter can be used to show all jobs that do not show up in any other view.


Unclassified Jobs

Match Type: Include Matched - Add jobs that match this f

Figure 3.21: Configuring the Unclassified Jobs View Filter

The User Permissions view filter can be used to only show jobs with a specific permission for the currently logged in user. A drop down allows you to define the matching used and the permissions are displayed below as check-boxes.


User Permissions for Jobs

Must Match All - Match jobs for which the current user has all of these permissions

- [ ] Configure
- [ ] Build
- [ ] Workspace

Match Type: Include Matched - Add jobs that match this filter

Figure 3.22: Configuring the User Permissions for Jobs View Filter

The All Jobs filter for simply adding all jobs and then using further filters to reduce the matches ending up in the view list.

The section below the Job Filters allows you to configure the columns used in the list view. Add and delete columns with the provided buttons as displayed in Figure 3.23. To change the order of columns you can drag the title and drop them in the desired position.

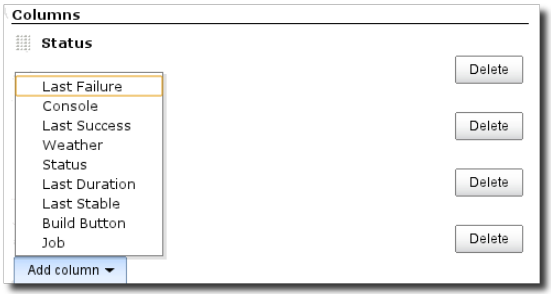
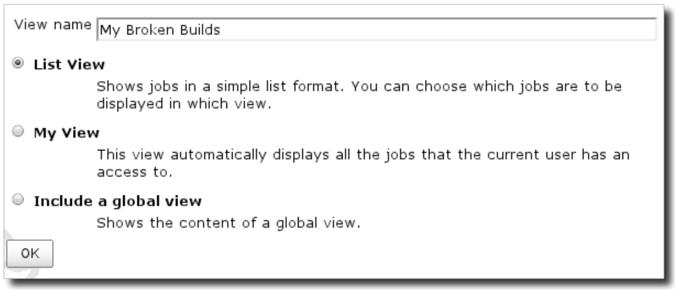


Figure 3.23: Configuring View Columns

When you are logged into Hudson as a user you can also create a different view type called My View. It automatically contains all projects you currently have access to.



View name: My Broken Builds

[✓ **List View**]
Shows jobs in a simple list format. You can choose which jobs are to be displayed in which view.

My View
This view automatically displays all the jobs that the current user has an access to.

Include a global view
Shows the content of a global view.

OK

Figure 3.24: Adding a New Personal View (My View)

3.12 Hudson Monitoring with RSS

Hudson exposes various RSS feeds that you can subscribe to in your favourite RSS feed reader. The server needs to be reachable via http by your RSS reader for this feature to be usable.

Various pages in the web interface feature RSS buttons in the bottom left corner as visible in Figure 2.1.

3.12.1 Receiving Build Notifications via RSS

Build notifications are available on the default Hudson page. You can receive a feed of all builds at `/rssAll`, a feed of failed builds at `/rssFailed` and a feed for latest builds at `/rssLatest`.

3.12.2 System Logs via RSS

If you access the Hudson logs exposed in the web interface at `/log/all` you will notice that you can subscribe to All log messages or Severe and Warning level warnings with the URLs `/log/rss`, `/log/rss?level=SEVERE` and `/log/rss?level=WARNING`.

Chapter 4

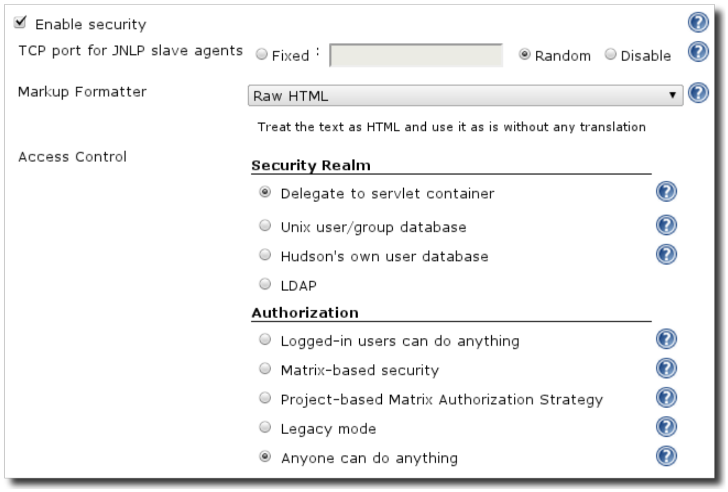
Securing Hudson

Hudson's security configuration can span a wide spectrum: from a simple Hudson instance running on a local network with no configured security, to an instance of Hudson supporting thousands of developers and thousands of projects with a security model providing isolation and access-control supporting a highly secured environment. Whether you support a local team of developers accessing an unsecured Hudson instance or a distributed enterprise team involving internal and external collaborators such as vendors and consultants, Hudson's security supports almost any conceivable use-case for authorization and access control.

This chapter provides an overview of the approaches to authentication and authorization in Hudson. After reading this chapter you will be able to take a Hudson instance and secure it allowing only authorized users to administer, access, and alter Hudson build jobs.

4.1 Security Settings Overview

Out of the box Hudson has no security enabled. To enable security, check Enable security in the global configuration of Hudson as displayed in Figure 3.2. Once you have enabled security you will be able to set up your desired security settings with the options visible in Figure 4.1 and documented in this chapter.


[✓] Enable security

TCP port for JNLP slave agents: Fixed : [ ] [✓ Random] Disable

Markup Formatter: Raw HTML

Treat the text as HTML and use it as is without any translation

Access Control

Security Realm

[✓ Delegate to servlet container]
Unix user/group database
Hudson's own user database
LDAP

Authorization

Logged-in users can do anything
Matrix-based security
Project-based Matrix Authorization Strategy
Legacy mode
[✓ Anyone can do anything]

Figure 4.1: Overview of the available security settings

4.2 Miscellaneous Security Related Settings

4.2.1 TCP port for JNLP slave agents

The port configuration should be set to Disable for Hudson deployments without any slave nodes. With slave nodes you can set the port to the default Random port. Hudson will randomly choose a port avoid port collisions with other services. When running Hudson cluster within a firewall-secured environment, you can choose a fixed port and then ensure that the port is open on the respective servers.

4.2.2 Markup Formatter

Raw HTML is the default setting that causes Hudson to render any input data from text fields as HTML in the user interface. This allows for added links and more, but also has the potential for cross site scripting XSS attacks. Currently plugins that implement other markup are planned, but not yet available for Hudson.

4.3 Authentication and Authorization

With security enabled, Hudson supports the following authentication security realms out of the box. Further options are available as plugins.

- Delegate to servlet container
- Unix user/group database
- Hudson's own user database
- LDAP (Lightweight Directory Access Protocol

Access-control schemes available are:

- Logged-in users can do anything
- Matrix-based security
- Project-based Matrix Authorizations Strategy
- Anyone can do anything

These security options can be seen in Figure 4.1 and are documented in detail below.

4.4 Configuring Security Realms: Authentication

The configuration of the security realm allows you to define where user names and passwords are stored and administered. Depending on your deployment it can be useful to tie into already existing systems or run a separate realm for Hudson.

4.4.1 Delegating to a Servlet Container

The default setting for a Hudson instance is no security. If you are running Hudson from a servlet container you may have access to management consoles that allow you to maintain and administer users and groups. Hudson can be configured to both delegate authorization to a servlet container and use these users and groups for access-control.

This feature can be especially useful if other application are already using the servlet container authorization and you want to achieve a single-sign on for all applications running on this server or cluster of servers or you simply prefer to manage your users from the application server user interface.

4.4.2 Relying on Unix Users and Groups

If you select this option, Hudson will consult the Unix user/group database on the machine it is running on. To do so it will use Pluggable Authentication Modules (PAM). The user running Hudson has to be able to access PAM and be a member of the `shadow` group. As visible in Figure 4.2 the setup allows for the definition of a service name and test via provided button.

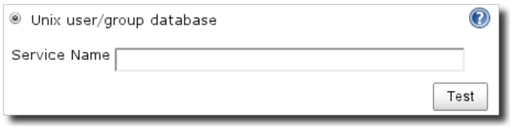


Figure 4.2: Unix security configuration

Tip
If you get a stack trace about not being able to find the file libpam.so you may need to create a symbolic link using the command line shown below.

```
sudo ln -s /lib/x86_64-linux-gnu/libpam.so.0 /lib/x86_64-linux-gnu/libpam.so
```

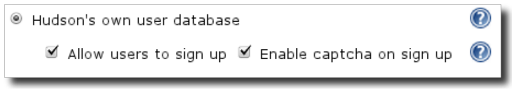
On most Unix systems, the users and groups are stored in `/etc/passwd` and `/etc/group`, but your results may vary depending on the Unix/Linux version and the security setup of your specific machine. With this setup user and group administration is entirely separate from your Hudson install. Use the your preferred administration tool on the command line or a graphical user interface.

A consequence of using this security realm is that the Hudson instance and access details are tied to the specific server Hudson is running on. This means that the user and group setup should be backed up in addition to the Hudson data itself and security information can not be easily migrated unless some sort of single sign-on (SSO) is used across all servers.

This approach is ideal for simple installations of Hudson on a single machine, or when you have integrated PAM with a single sign-on solution such as LDAP.

4.4.3 Using the Hudson Internal User Database

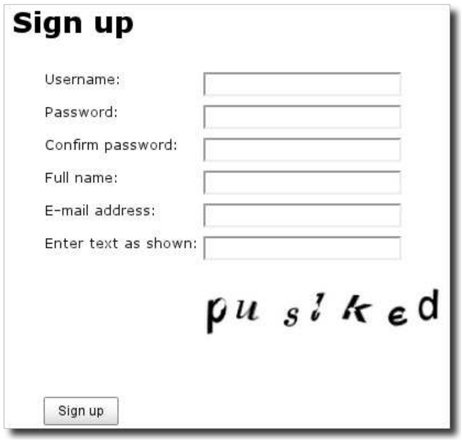
A convenient method to control access to Hudson without external dependencies is to use the internal user database of Hudson itself. To activate this feature select Hudson's own user database as displayed in Figure 4.3



[✓ Hudson's own user database]
[✓ Allow users to sign up] [✓ Enable captcha on sign up]

Figure 4.3: Security settings for using the Hudson internal database

The option "Allow user to sign up" activates the Sign Up screen as displayed in Figure 4.4, which also displays a captcha that is displayed when the option "Enable captcha on sign up" is activated. The option "Notify user of Hudson account creation" will trigger an email to be sent to the new user, when a new account is created either by signup by the user or an administrator.

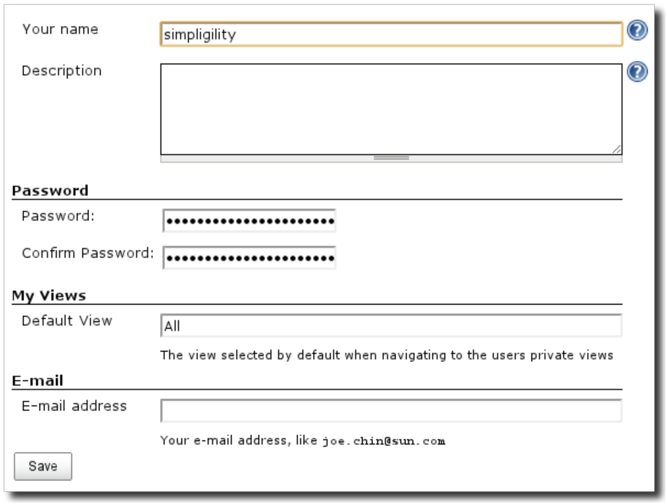


Sign up

Username:	
Password:	
Confirm password:	
Full name:	
E-mail address:	
Enter text as shown:	

Figure 4.4: Sign up screen for new users

By providing all details in the sign up form a user can create an account to access Hudson. Once a user is signed up and logged in they can use then click on their user name in the top left corner and then on the Configure option in the left hand menu to access their user configuration screen. A user's configuration screen gives users the ability to reset and change passwords as well as update contact information such as name, description, and email.



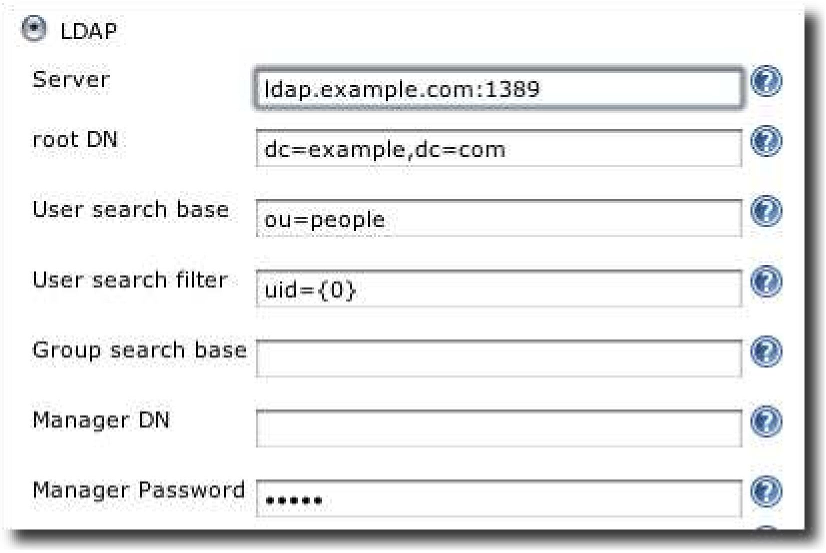
Your name	simpligility
Description	
Password	
Password:	••••••••••••••••••••••
Confirm Password:	••••••••••••••••••••••
My Views	
Default View	All
	The view selected by default when navigating to the users private views
E-mail	
E-mail address	
	Your e-mail address, like joe.chin@sun.com

Figure 4.5: User configuration

To manage users click on Manage Users in the global Manage Hudson screen displayed in Figure 3.1. In this management section the administrator has the option to create and manage users. The Create User link in the left-hand navigation menu presents the Sign Up screen from Figure 4.4 without the captcha to the administrator allowing account creation to be controlled by an administrator rather than a self-serve signup. In addition this screen shows a list of users already registered with Hudson and allows the administrator to delete users and edit them via the user configuration screen from Figure 4.5.

4.4.4 Light-weight Directory Authentication Protocol (LDAP)

If you have an LDAP server, Hudson can be configured to use this server to authenticate users. The administration interface in Figure 4.6 allows you to provide all the necessary details for Hudson to connect to your LDAP server. You will be able to connect to most common LDAP servers. If you want to configure security with Microsoft Active Directory you should also have a look at the specific plugin for Active Directory.



◉ LDAP

Server	ldap.example.com:1389
root DN	dc=example,dc=com
User search base	ou=people
User search filter	uid={0}
Group search base	
Manager DN	
Manager Password	•••••

Figure 4.6: Security settings for LDAP

Server
The server connection string is typically composed of a protocol of ldap:// or ldaps://, a server name and a colon followed by a port. The default ldap protocol and port 389 can be omitted, so the simplest server would be the server name or IP number only. Further valid examples are e.g. directory.example.com:1389 or ldaps://directory.example.com:1636. The default port for ldaps is 636.

root DN
Specify the root DN for you want to use for authentication. Typically this would be a value like dc=example,dc=com

User search base and User search filter
The user search base and use search filter fields allow you to define a sub-tree to look for user records within the root DN. E.g the search base could be ou=people or ou=developers and the search with their email address. If you need to further define your search user name in the search.

Group search base
Define the query to locate the list of groups for a user in this query that is typically just empty to search in the root DN or for the user name replacement.

Manager DN and Manager Password
The Manager DN and password are required for Hudson to connect to your LDAP server to query for the user details. The DN is only necessary if you server does not support anonymous bind and typically looks something like this CN=MyUser,CN=Users,DC=mydomain,DC=com.

Beyond the standard settings LDAP configuration can require further tweaks documented in the following section.

In general all the LDAP queries are space sensitive and their content is specific to your LDAP server configuration which can differ widely depending on your specific organization's LDAP configuration.

If log-in attempts result in "Administrative Limit Exceeded" or similar error message, any LDAP query but often specifically the LDAP query to determine the group membership triggers it. A general tip in this situation is to try setting the "Group search base" and other settings as specific as possible for your LDAP structure, to reduce the scope of the query.

If the error persists, you may need to edit the WEB-INF/security/LDAPBindSecurityRealm.groovy file that is included in in your LDAP for group membership, such as groupSearchFilter =

In general Hudson will prefix any LDAP group with the ROLE_ prefix. This has to be taken into account when configuring access rights in a matrix based security realm. For example, the LDAP group developers (cn=developers) would be used as ROLE_DEVELOPERS. In Hudson, Group names are translated to all uppercase and non-alpha characters such as hyphen, space and comma are not supported.

Hudson's LDAP integration currently does not support indirect group memberships.

4.5 Configuring an Access-control Strategy

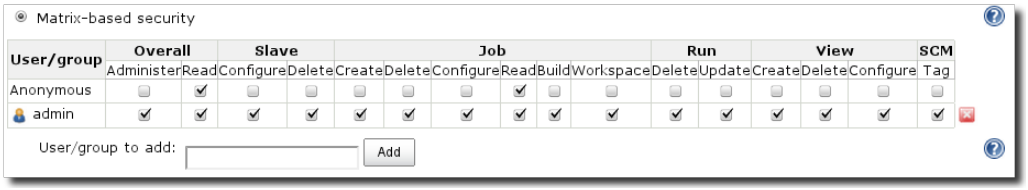
Once you've selected a security realm for authorization you'll know how users and groups are created and what has to be done for a user to be able to log in to Hudson. As a next step you need to decide on your Hudson instance's approach to access-control. Access control settings determine what a user can do and see once they are logged in.

4.5.1 Logged-in users can do anything

This authorization strategy grants read access to Hudson to anonymous users, but restricts administrative access to users with a valid account. Once the user is logged-in they have full access rights to everything including project deletion and other critical functionality. You should be certain to have a good backup strategy in place for critical data and that your users are capable of using Hudson in an administrative function before you opt to use this authorization strategy.

4.5.2 Matrix-based security

For more fine grained control over what specific users or groups of users can do, you can configure Hudson to use Matrix-based security as visible in a minimal configuration in Figure 4.7.



◉ Matrix-based security

User/group	Overall		Slave		Job						Run		View			SCM
	Administer	Read	Configure	Delete	Create	Delete	Configure	Read	Build	Workspace	Delete	Update	Create	Delete	Configure	Tag
Anonymous	☐	✓	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐
admin	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

User/group to add: Add

Figure 4.7: Matrix based security

Using the input field for User/group you can create new rows in the security matrix. The matrix provides sections to configure access rights for the following permissions categories:

- Overall - permissions which govern global activities such as administration rights
- Slave - permissions relating to the management of Hudson slave instances
- Job - job permissions whether a user can create, manage, edit, and delete jobs
- Run - permissions about specific build jobs
- View - permissions about build job views
- SCM - permissions related to source code management systems

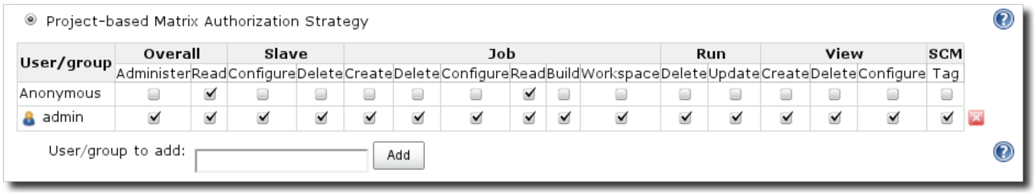
A typical minimal configuration would be to grant all rights to an administrative user or group and only read access for Overall and Jobs to anonymous users.

By adding further groups or individual users you can e.g. grant full administrative access rights to other trusted users without sharing the main admin account.

Tip
A safe administrative use would be to grant full rights to a group of admin users without granting any Delete rights.

4.5.3 Project-based Matrix Authorization Strategy

Taking the concept of matrix based authorization as described in Section 4.5.2 a step further is Project-based Matrix Authorization Strategy. The global configuration possible as displayed in Figure 4.8 works the same as matrix-based security.



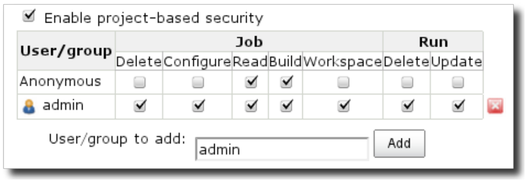
[● Project-based Matrix Authorization Strategy]

User/group	Overall		Slave		Job						Run		View			SCM
	Administer	Read	Configure	Delete	Create	Delete	Configure	Read	Build	Workspace	Delete	Update	Create	Delete	Configure	Tag
Anonymous	☐	✓	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐
admin	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

User/group to add:

Figure 4.8: Project-based Matrix Authorization Strategy

In addition you can enable project-based security in each project configuration individually for each project. Enabling the option Enable project-based security will display a matrix of access rights as visible in Figure 4.9. It will allow you to add users and groups just like for the global configuration and assign rights as desired.



✓ Enable project-based security

User/group	Job					Run	
	Delete	Configure	Read	Build	Workspace	Delete	Update
Anonymous	☐	☐	✓	✓	☐	☐	☐
admin	✓	✓	✓	✓	✓	✓	✓

User/group to add: admin

Figure 4.9: Project specific authorization

Using groups accessing a group of projects you could e.g. enable administrative rights for a limited number of projects to a specific user or user group.

4.5.4 Anyone can do anything

This option is something of a free for all - an uncontrolled Hudson instance with no access-control rules defined.

Setting your authorization strategy to the option "Anyone can do anything" effectively turns off security. In a scenario where everybody able to access Hudson e .g. in a trusted intranet this setting is recommended as an alternative to completely disabled security. The advantage for the users is that while they do not need to log-in to use Hudson and everyone has access it is possible to log-in and customize Hudson by creating custom views and take advantage of other personalization options.

Warning
Older versions of Hudson have a legacy mode authorization that provides backwards compatibility for users of early Hudson version, that have not migrated to the replacement of matrix authorization. It relied on hard coded values and should not be used anymore. With the 2.2.0 release of Hudson this mode is removed.

4.6 Hudson Security Best Practises, Tips and Tricks

4.6.1 Common Setup - Internal matrix-based authorization

A common and useful setup is a combination of using Hudson's internal user database with matrix-based authorization. It allows for a secure setup of a publicly available Hudson instance without the need for any further security components beyond Hudson itself and can therefore be managed via the Hudson user interface without any additional requirements beyond browser access.

1. Enable security in the global Hudson configuration
2. Activate the security realm for Hudson's own user database and allow users to sign up
3. Set the authorization to "Anyone can do anything" and save the configuration
4. Sign up a new user e.g. with the name `admin`
5. Log-in in as the new user
6. Change the authorization to matrix-based security
7. Add the new user to the matrix
8. Grant all right to the new user as he will the be the administrator user
9. Save the configuration

Following these steps you will have secured Hudson and the new user will be the only user with access to Hudson. If you want anonymous users to have read access you could add these rights in the matrix.

For further users you can create additional matrix rows and distribute rights as desired.

4.6.2 Disabling security when locked out

When configuring security or when relying on external security realms, you can end up in situations where you do not have any access to Hudson in the user interface. Reasons could be a forgotten admin password, offline LDAP server, broken Unix authorization after server upgrade and so on. To be able to fix your setup you can edit the file *config.xml* in your HUDSON_HOME and set

```
<useSecurity>false</useSecurity>
```

With this setting you will have full access to Hudson and be able to troubleshoot your configuration or change to a new security realm and/or authorization.

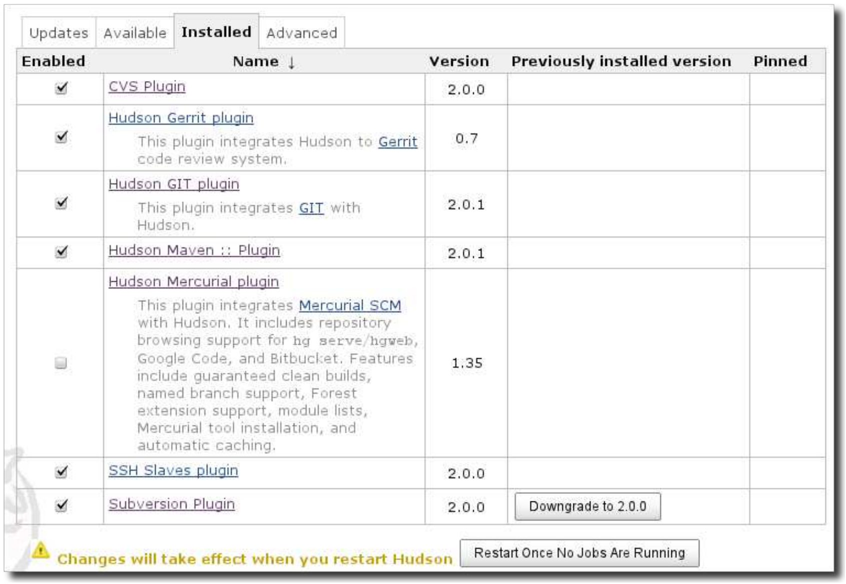
Chapter 5

Managing Hudson Plugins

Hudson plugin management is available via Manage Hudson and selecting the Manage Plugins link shown in Figure 3.1. This administration interface allows you see what plugin versions are currently installed, update them and install new ones as well as manage some advanced settings to work with the multitude of plugins available for Hudson. Using plugins allows you to support many new features beyond a basic Hudson install as well as tweak the user interface and morph Hudson into the CI server you need.

5.1 Installed Plugins

To get a list of installed plugins, click in the Installed tab. This will display a list all of the Hudson plugins currently installed on your instance of Hudson system.



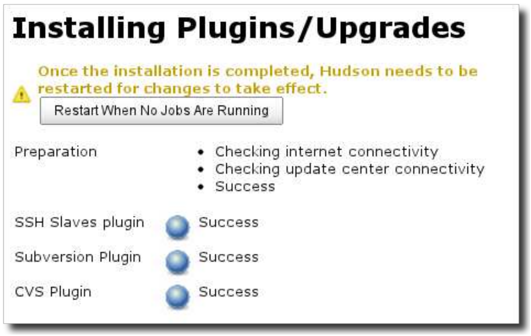
Updates | Available | Installed | Advanced

Enabled	Name ↓	Version	Previously installed version	Pinned
✓	CVS Plugin	2.0.0		
✓	Hudson Gerrit plugin This plugin integrates Hudson to Gerrit code review system.	0.7		
✓	Hudson GIT plugin This plugin integrates GIT with Hudson.	2.0.1		
✓	Hudson Maven :: Plugin	2.0.1		
☐	Hudson Mercurial plugin This plugin integrates Mercurial SCM with Hudson. It includes repository browsing support for hg serve/hgweb, Google Code, and Bitbucket. Features include guaranteed clean builds, named branch support, Forest extension support, module lists, Mercurial tool installation, and automatic caching.	1.35		
✓	SSH Slaves plugin	2.0.0		
✓	Subversion Plugin	2.0.0	Downgrade to 2.0.0	

Changes will take effect when you restart Hudson

Figure 5.1: The Installed tab for managing the installed plugins

The Enabled check box allows you to activate and deactivate specific plugins. After plugin updates your old plugin version will remain accessible and you can downgrade to the older version by pressing the Downgrade to x.y.z button. The pinned column will then be marked for this plugin so that automatic updates will not occur until you unpin the version. Any changes will require you to restart Hudson by clicking the "Restart once no jobs are running button" as displayed in Figure 5.2.



Installing Plugins/Upgrades

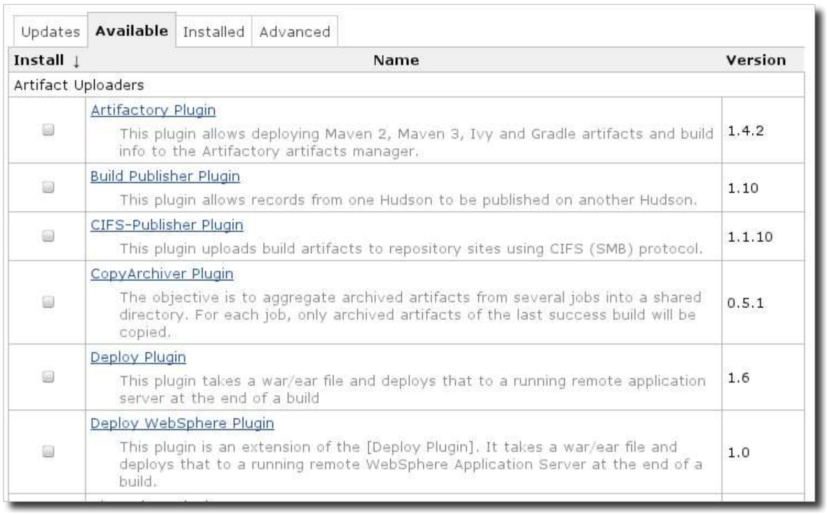
Once the installation is completed, Hudson needs to be restarted for changes to take effect.

Preparation		• Checking internet connectivity • Checking update center connectivity • Success
SSH Slaves plugin	●	Success
Subversion Plugin	●	Success
CVS Plugin	●	Success

Figure 5.2: Install and update plugin progress screen ready to restart the server

5.2 Available Plugins

This list of available plugins includes hundreds of useful utilities, tweaks, and feature sets which extend the core feature set of Hudson. The list is separated into topics and contains a short description for each plugin as well as a link to the plugin web site. In order to install a plugin, simply select the check box in the plugin row, press the Install button on the bottom of the list. This will redirect to a progress reporting page. After a success message you will have to restart Hudson by clicking the Restart once no jobs are running button for the new plugin to be available.

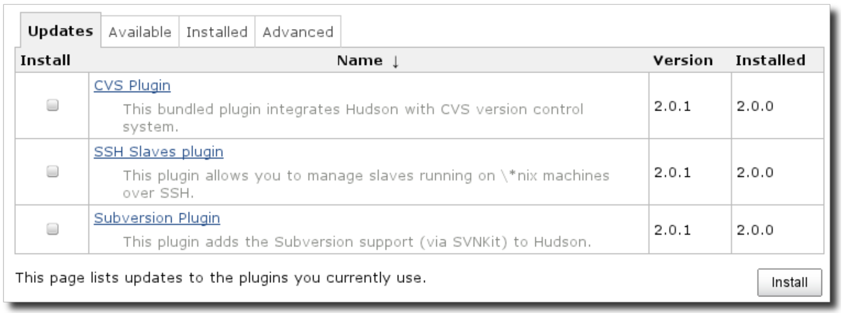


Install ↓	Name	Version
Artifact Uploaders		
☐	Artifactory Plugin This plugin allows deploying Maven 2, Maven 3, Ivy and Gradle artifacts and build info to the Artifactory artifacts manager.	1.4.2
☐	Build Publisher Plugin This plugin allows records from one Hudson to be published on another Hudson.	1.10
☐	CIFS-Publisher Plugin This plugin uploads build artifacts to repository sites using CIFS (SMB) protocol.	1.1.10
☐	CopyArchiver Plugin The objective is to aggregate archived artifacts from several jobs into a shared directory. For each job, only archived artifacts of the last success build will be copied.	0.5.1
☐	Deploy Plugin This plugin takes a war/ear file and deploys that to a running remote application server at the end of a build	1.6
☐	Deploy WebSphere Plugin This plugin is an extension of the [Deploy Plugin]. It takes a war/ear file and deploys that to a running remote WebSphere Application Server at the end of a build.	1.0

Figure 5.3: The Available tab for installing new plugins

5.3 Plugin Updates

Hudson regularly checks an available list of Hudson plugins on the Hudson web site and will notify if an update becomes available. To see if any updates are currently available click on the Updates tab in the Manage Plugins screen. Plugins eligible for an update will be displayed next to the available update version and the version which is currently installed. To update a plugin select the check box in the Install column and press the Install button on the bottom of the list. This will redirect to a progress reporting page. After a success message you will have to restart Hudson by clicking the "Restart once no jobs are running" button for the new plugin to be available.



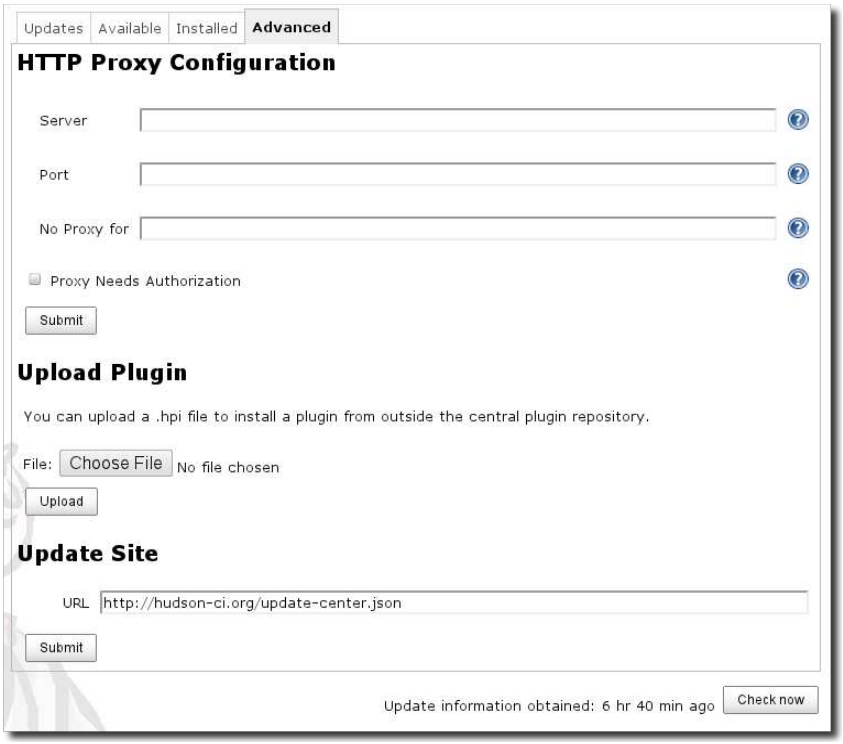
Install	Name ↓	Version	Installed
☐	CVS Plugin This bundled plugin integrates Hudson with CVS version control system.	2.0.1	2.0.0
☐	SSH Slaves plugin This plugin allows you to manage slaves running on *nix machines over SSH.	2.0.1	2.0.0
☐	Subversion Plugin This plugin adds the Subversion support (via SVNKit) to Hudson.	2.0.1	2.0.0

This page lists updates to the plugins you currently use.

Figure 5.4: The Updates tab displaying available updates of installed plugins

5.4 Advanced Plugin Settings

The Advanced tab as displayed in Figure 5.5 allows you to configure the proxy settings for Hudson to be able to connect to the plugin repository on the Hudson web site, upload Hudson plugin files (`*.hpi`) manually, and specify the URL for a custom update site.



HTTP Proxy Configuration

Server

Port

No Proxy for

Proxy Needs Authorization

Submit

Upload Plugin

You can upload a .hpi file to install a plugin from outside the central plugin repository.

File: Choose File No file chosen

Upload

Update Site

URL http://hudson-ci.org/update-center.json

Submit

Update information obtained: 6 hr 40 min ago

Check now

Figure 5.5: The Advanced tab for miscellaneous plugin management tasks

5.5 HTTP Proxy Configuration

Hudson retrieves a list of plugins and downloads plugins over the public Internet. If your server is behind a proxy server, you will need to configure the proxy settings. Supply the necessary values to connect to your internal proxy in the Server and Port fields. The No Proxy for field allows you to exclude host names from proxying by adding them to a comma separated list. If your proxy server requires authentication you will have to select the Proxy Needs Authorization check box and provide the User name and Password.

5.6 Upload Plugin

Hudson plugins are distributed as `.hpi` files and the the Advanced administration section allows you to upload them with a file chooser. This feature is especially useful for installing custom developed plugins or commercially distributed plugins that are not available on the update site. You can also use it to install plugins you build from source during development or when helping by contributing fixes to open source plugins.

5.7 Update Site

Instead of using the public update site on the Hudson web site you can host your own plugin repository and you can then add the URL in the available input field. This approach is useful if you need to maintain a secured Hudson instance that cannot be affected by the day to day changes on an external Hudson plugin update site. If you use this option, you can point your Hudson instance at a local copy of the public Hudson plugin update site and strictly control the changes that are visible to your Hudson instance.

Chapter 6

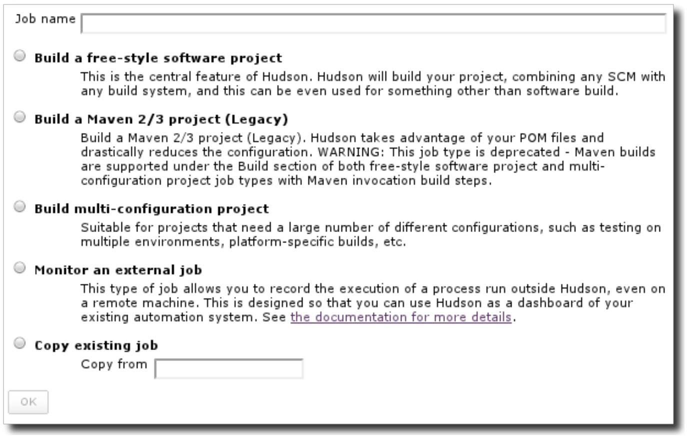
Creating Hudson Projects

Once you've downloaded, installed, and started Hudson for the first time, the next step is to start creating new Hudson build jobs. As you'll learn in the following chapter, there are an innumerable number of options, plugins, and extensions that can make the Hudson build job creation process a bit daunting to the first time user.

From SCM configuration to options relating to build triggers Hudson offers so many configuration points it is often difficult for new users to make sense of the process. This chapter takes the mystery out of build job creation and explores some of the options available when creating a new Hudson project. After reading this chapter, you'll know what fields matter (and which fields can be safely ignored), you'll understand how to stand up a new Hudson build job, and you'll be ready to get started using the best CI server in the industry.

6.1 Creating New Hudson Projects

To create a new Hudson project, click on New Job in the left navigation menu, which will display the form shown in Figure 6.1.



Job name

Build a free-style software project

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

Build a Maven 2/3 project (Legacy)

Build a Maven 2/3 project (Legacy). Hudson takes advantage of your POM files and drastically reduces the configuration. WARNING: This job type is deprecated - Maven builds are supported under the Build section of both free-style software project and multi-configuration project job types with Maven invocation build steps.

Build multi-configuration project

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

Monitor an external job

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

Copy existing job

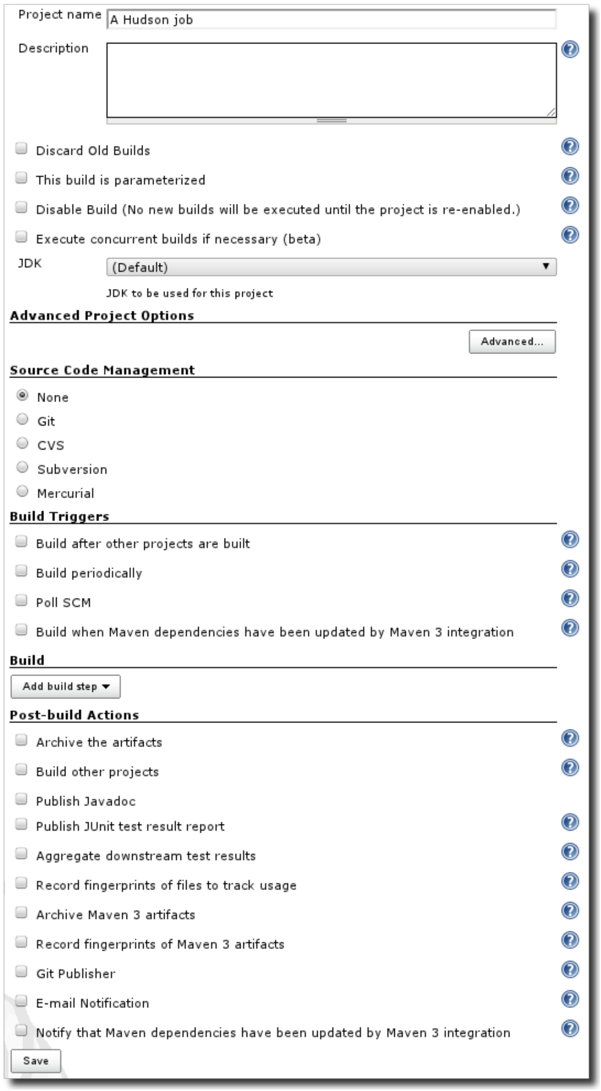
Copy from

Figure 6.1: Creating a New Job in Hudson

The most common way to create a new job using Hudson is to select "Build a free-style software project". If you are creating a project that is similar to an existing Hudson build, you may also choose "Copy an existing job" and then type in the name of the Hudson job in the Copy from text field. Once you have made your selection and provided a name for the new job, click on OK to continue.

You can also configure Hudson to Monitor an external job which can then trigger and affect other jobs. The last job type supported by Hudson is to Build a multi-configuration project, which can act like a container for a variety of different job executions.

Warning
Your Hudson installation may provide an option to Build a Maven 2/3 project (Legacy). This feature has numerous issues with the use of different Maven versions, and it is especially problematic when building a project with Maven 3. The authors of this book recommend not to using the legacy Maven project type. Instead, we encourage you to use the native Maven support which is available in a free-style software project.



Project name: A Hudson job

Description

☐ Discard Old Builds

☐ This build is parameterized

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

☐ Execute concurrent builds if necessary (beta)

JDK: (Default)

JDK to be used for this project

Advanced Project Options

Advanced...

Source Code Management

● None

○ Git

○ CVS

○ Subversion

○ Mercurial

Build Triggers

☐ Build after other projects are built

☐ Build periodically

☐ Poll SCM

☐ Build when Maven dependencies have been updated by Maven 3 integration

Build

Add build step

Post-build Actions

☐ Archive the artifacts

☐ Build other projects

☐ Publish Javadoc

☐ Publish JUnit test result report

☐ Aggregate downstream test results

☐ Record fingerprints of files to track usage

☐ Archive Maven 3 artifacts

☐ Record fingerprints of Maven 3 artifacts

☐ Git Publisher

☐ E-mail Notification

☐ Notify that Maven dependencies have been updated by Maven 3 integration

Save

Figure 6.2: Result of New Project Creation

Once your job has been created, you will see a screen similar to the one shown in Figure 6.2, depending on your job type selection. You can now proceed to configure your project. If you need to change any of these settings at a later stage, navigate to a particular Hudson Project’s Summary page by clicking on the name of the project on the main Hudson page. Once you are on a Hudson Project’s Summary page, click on Configure in the left-hand navigation to load the Project Configuration screen.

6.2 Configuring Common Job Configuration Settings

Independent of the job type you selected you will be able to configure a few common settings separated in the following sections:

General Project Settings
Configure the project name, description and other general parameters

Advanced Project Options
Configure miscellaneous settings for advanced usage.

Source Code Management
Configure source code management-related parameters for various systems

Build Triggers
Configure how builds are started

Post-build Actions
Configure steps taken after a build completion

Depending on the job type you choose additional sections may be present in the project configuration screen. The following sections provide an overview of some common project configuration options.

6.2.1 Configuring General Project Settings

The first section on the Project Screen is the general project information, which is shown in Figure 6.3.

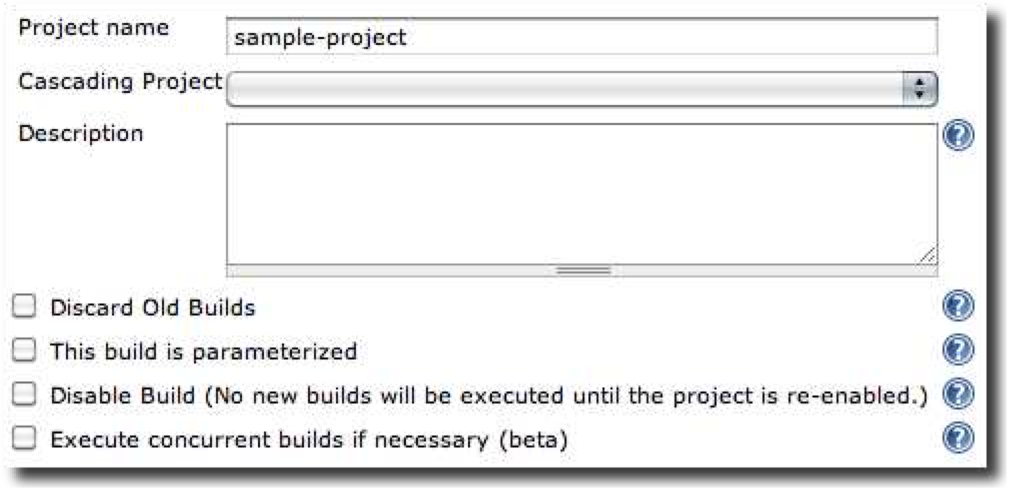

Project name
sample-project
Cascading Project
Description
Discard Old Builds
This build is parameterized
Disable Build (No new builds will be executed until the project is re-enabled.)
Execute concurrent builds if necessary (beta)

Figure 6.3: Configuring Project Information

The section shown in the previous figure contains the following fields:

Project Name
The Project Name should be a short descriptive name that easily allows your Hudson users to see what this project is. Consider this to be the identifier that Hudson uses to keep track of everything associated to this project. We recommend that your project names consist of simple alphanumeric characters and dashes. While Hudson will save a Project name containing spaces, the file path of the workspace will also contain spaces. The presence of spaces in a project name can cause unforeseen issues with builds and may results in build failure. We suggest using e.g. underscores instead. The project name will be visible in the main Hudson page on the list of jobs.

Cascading Project
The Cascading Project drop down allows you to select another Hudson project as a template to inherit settings from. This feature is supported for free-style software as well as multi-configuration projects and is documented in more details in Section 6.2.6.

Description
The Description should contain a paragraph that will inform Hudson users about the nature and purpose of a given Hudson project. Useful information for the users could for example be various source code management related parameters like branch or version or other parameters like target platform for the build artifacts. This will widely vary and depend on your job type. The description is visible on the main project-specific page.

Discard Old Builds
If Discard Old Builds is not checked, Hudson will archive the results of all the builds it performs. Depending on build log and artifact size this can produce considerable amounts of storage space being used, which should in turn be monitored carefully. If Discard Old Builds is checked, the project configuration screen will display additional input fields that allow you to specify the number of builds and the number of days to retain builds.

After pressing the Advanced button you can provide for separate configuration for the number of days the build artifacts are kept. Setting this value will cause older build artifacts to get deleted, without the logs, reports etc. getting deleted. An additional advanced configuration allows you to set maximum number of builds to keep including its artifacts.

This Build is Parameterized

Parameterized builds allow you to pass configuration to specific builds. For example, you can parametrize a build to accept a free-form variable that selects a specific branch in SCM. This way, a Hudson user can trigger a build and supply the name of this branch to the Hudson build. Hudson supports different data types adapting its user interface to it. The basic string parameter provides a simple input box. The password parameter will provide an input field with hidden characters. Boolean values are represented as check-box, while a choice parameter is rendered as a drop down box. File parameters, run parameter and subversion tags are other supported parameters. All these parameters need to be supplied when a project is build. Typically this is done via the user interface provided by the parameters when a build is kicked off manually in the user interface. Parameters can also be passed to Hudson via remote invocation with a URL like http://server/job/myjob/-buildWithParameters?PARAMETER=Value

Disable Build

If Disable Build is selected, no new builds will be executed until the project is re-enabled. This means that any builds that might have been triggered by SCM activity or by a periodic schedule will not be executed. This feature is very useful if you need to fix an issue with a build or build specific related infrastructure like source code management system and you want to temporarily take a particular Hudson job offline without affecting the rest of your Hudson setup and jobs.

Execute concurrent builds if necessary (beta)

If this check box is selected, Hudson will be able to execute more than one build for this project at the same time. This can be useful if your project is parameterized, or if you have a longer build, which may need to run multiple concurrent builds in response to independent changes to SCM. In many cases this setting is particularly useful, when Hudson is set up as a build cluster.

JDK

This drop down allows the selection of a specific Java Development Kit (JDK) for the project. It will only be available if multiple JDKs are configured in the global Hudson configuration as documented in Section 3.3.

Restrict where this project can be run

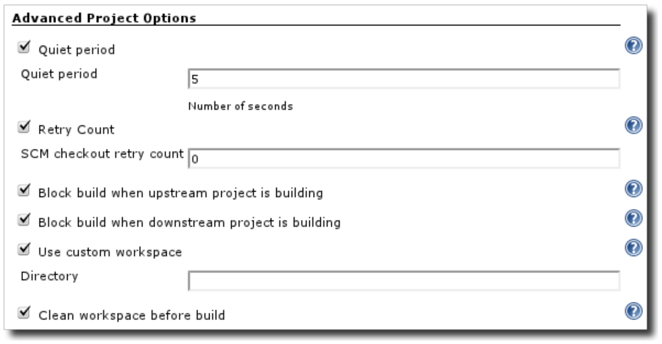
If Restrict where this project can be run is checked, Hudson will display options that will allow you to specify the nodes on which a project build can be executed.

Warning

If you are creating a continuous integration build that will run frequently, don't forget to check Discard Old Builds and configure Hudson to free up drive space. If your project is built regularly due to frequent changes in source control or small times between fixed schedule builds you can easily fill up even the largest hard drive potentially resulting in your build server going offline.

6.2.2 Configuring Advanced Project Options

The next section after the general project configuration is the Advanced Project Options section shown in Figure 6.4.


Advanced Project Options

✓ Quiet period

Quiet period: 5

Number of seconds

✓ Retry Count

SCM checkout retry count: 0

✓ Block build when upstream project is building

✓ Block build when downstream project is building

✓ Use custom workspace

Directory:

✓ Clean workspace before build

Figure 6.4: Configuring Advanced Project Options

This section contains the following fields:

Quiet period
: The project-specific Quiet Period set in this section overrides the global configuration documented in Section 3.1 and has the same effect. This setting depends on your build trigger configuration as documented below.

Retry Count
: The project-specific Retry Count overrides the global SCM checkout retry count and has the same effect.

Block build when upstream/downstream project is building
: Hudson builds can be configured to have upstream as well downstream dependencies. Upstream dependencies are projects upon which this particular project's build depends. Downstream dependencies are projects that depend on the current builds results. If Block build when upstream/downstream project is building is selected this project will not start a build, if an upstream/downstream project is in the middle of a build or in the build queue.

Use custom workspace
: If this selection is checked, you can instruct Hudson to use a custom directory for this project's workspace. If this option is not checked, Hudson will automatically assign a workspace location that is based on the project's name. As such it can be used to have projects names with spaces or other characters potentially causing file system level issues while using a save name for the workspace folder name.

6.2.3 Configuring Source Code Management

Hudson is typically used in conjunction with the source of the project available in a source code management system. Support for a large variety of SCMs is one the strengths of Hudson. This section allows for the configuration of the respective settings for your SCM of choice. The available list of choices will contain all SCM systems provided by Hudson and the installed plugins. Chapter 5 explains how to get support for your SCM installed, if it is not yet available. Once installed each SCM configuration will have different parameters, which are documented in Chapter 9.

6.2.4 Configuring Build Triggers

The next section to configure is Build Triggers. A Hudson job can be configured to build in response to build activity on a Hudson instance, in accordance with a regular schedule, or as a reaction to activity in an SCM system. Build triggers are configured on a Project's Configuration screen and the section is shown in Figure 6.5.

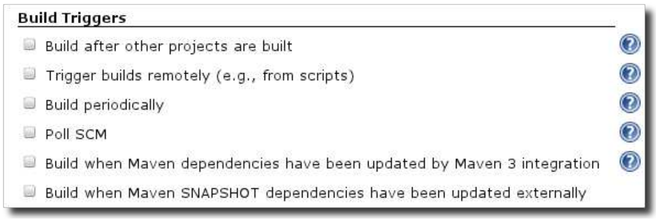

Build Triggers
- [ ] Build after other projects are built
- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build periodically
- [ ] Poll SCM
- [ ] Build when Maven dependencies have been updated by Maven 3 integration
- [ ] Build when Maven SNAPSHOT dependencies have been updated externally

Figure 6.5: Configuring Build Triggers

The following types of build triggers can be configured:

Build after other projects are built
: If this option is selected, Hudson will present you with a text field that accepts the names of one or more projects. If this field is populated, Hudson will trigger this project's build after a successful completion of one of the projects listed in this text field. The reverse behaviour of triggering a different project based on this project's build completing can be configured in the Post-build Actions.

Trigger builds remotely (e.g. from scripts)
: Enabling this option will allow you to define a authentication token for triggering a build by hitting a project specific URL together with the defined token. The URL follows the patterns http://hudsonurl/jobs/project/-build?token=mytoken&cause=mycause with hudsonurl replaced by the URL of your hudson server including port number and context (e.g. http://buildserver:8080/hudson). mytoken will be the secret token you defined in the Authentication Token field and mycause an optional text to submit as the cause. To allow users to hit the URL you can e.g. host the link on a secured web page, send it via email or also use curl or wget from a script.

Build periodically
: The Build periodically setting will cause Hudson to start a build of the project at regular intervals. Changes will

be retrieved from the configured SCM, but a build will be triggered even if no changes were found. The interval configuration in the exposed text field accepts the same cron syntax as the Poll SCM configuration.

Poll SCM
Selecting Poll SCM will cause Hudson to periodically poll your source code management system for changes and trigger a build if changes have been found and successfully retrieved. Selecting this option displays a text area, which accepts a standard cron expression. This allows for arbitrary polling interval from minutes to weekly and way beyond. It is important to consider the impact of this polling frequency setting on your SCM infrastructure, since it can cause significant load specifically when multiple projects with small polling frequencies all access the same SCM server. This expression can also be useful to set up a schedule for a project that only polls the SCM for changes e.g. outside office hours and therefore only builds then. Similar setup ups can be used to do a nightly or a weekly build type setup.

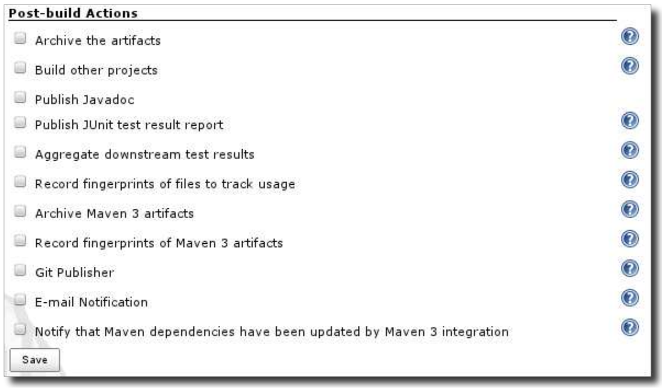
Build when Maven dependencies have been updated by Maven 3 integration
Hudson projects using the Maven 3 integration can be configured to send out a notification to all projects that Maven dependencies have been updated as post build action (see Section 6.2.5). When you activate this build trigger a build of the project will be started once a such a notification is received. The projects triggering each other in this set up needs to be configured to access the same Maven repository.

Build when Maven SNAPSHOT dependencies have been updated externally
Activating this setting allows you to specify a schedule in cron syntax to use for polling the local Maven repository for updated SNAPSHOT dependencies. When selecting the Exclude internally produced dependencies option the regular check will only cause a build trigger firing when artifacts not built in a local Hudson job are updated.

6.2.5 Configuring Post-build Actions

Post build actions are an important part of Hudson. They allow you to trigger a number of events upon build completion. These include the communication of the results of the build in various ways as well as chain other builds to this build. As such these actions fulfil a crucial role of a continuous integration server as communication tool. Other important actions allow you to deal with the various artifacts produced by the build in the form of test results, documentation as well as actual executables or archives produced.

Beyond the core post-build actions documented in the following, various plugins will make additional actions available. Source code management-related plugin documentation can be found in Chapter 9. The minimum list of build actions available is visible in Figure 6.6 and documented in more detail below.


Post-build Actions

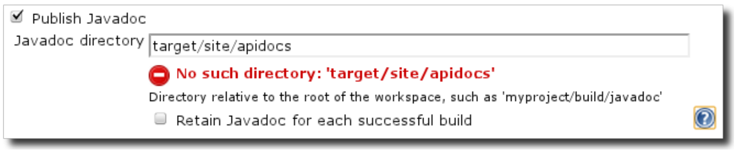
- ☐ Archive the artifacts
- ☐ Build other projects
- ☐ Publish Javadoc
- ☐ Publish JUnit test result report
- ☐ Aggregate downstream test results
- ☐ Record fingerprints of files to track usage
- ☐ Archive Maven 3 artifacts
- ☐ Record fingerprints of Maven 3 artifacts
- ☐ Git Publisher
- ☐ E-mail Notification
- ☐ Notify that Maven dependencies have been updated by Maven 3 integration

Figure 6.6: Configuring Project Post-build Options

Archive Maven 3 artifacts
If you activating this option Hudson will archive the artifacts built by the Maven 3 build steps during the build. Selecting the Include generated POMs additional cause the pom.xml files to be archived. With the option Discard old artifacts set, old artifacts will be removed after each successful build. This is a Maven 3 build step specific automation of the option to Archive artifacts without the need to specify the artifacts. The artifacts will be made available on the web interface like other archived artifacts. This option is especially useful in refactorings of the project and the resulting artifacts, since all artifacts are always archived and name or folder changes do not affect the archiving. Since there is no necessity to configure matching patterns, this option is more robust.

Record fingerprints of Maven artifacts
This option activated will trigger the Maven 3 integration of Hudson to record fingerprints of the created Maven artifacts, which will allow Hudson to keep track of when these artifacts are produced and used.

Publish Javadoc
If your build produces Javadoc you can configure Hudson to make it available to users on the project page of Hudson:


☑ Publish Javadoc

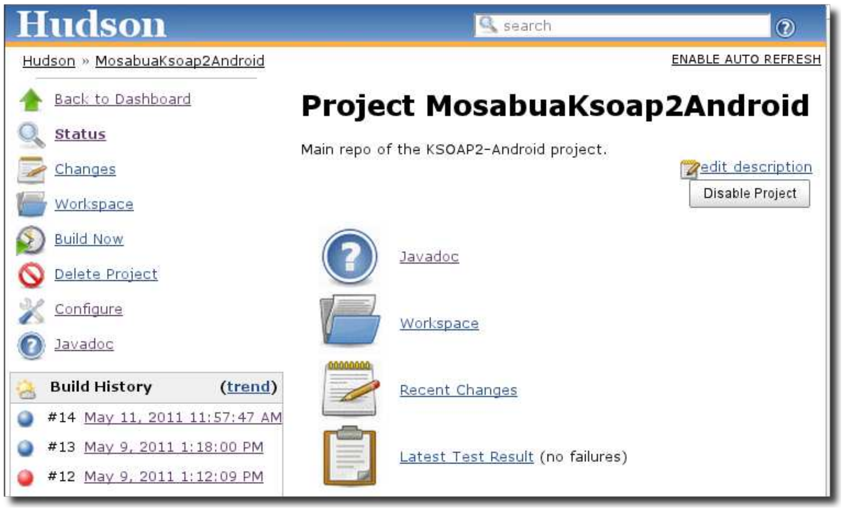
Javadoc directory: target/site/apidocs

No such directory: 'target/site/apidocs'

Directory relative to the root of the workspace, such as 'myproject/build/javadoc'

☐ Retain Javadoc for each successful build

Figure 6.7: Configuring the Javadoc post build action



Project MosabuaKsoap2Android

Main repo of the KSOAP2-Android project.

Latest Test Result (no failures)

Build History (trend)

#14 May 11, 2011 11:57:47 AM
#13 May 9, 2011 1:18:00 PM
#12 May 9, 2011 1:12:09 PM

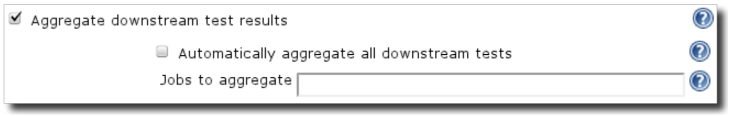
Figure 6.8: Project page providing access to Javadoc and latest test results

To configure this you have to select the check box and provide the relative path to the Javadoc in the Javadoc directory input field. Checking the Retain Javadoc for each successful build will keep the generated Javadoc for all successful builds in the specified folder for older builds instead of overwriting the documentation with each build.

If a warning of No such directory is visible, it means that the current project workspace does not contain the specified path. This is not a problem as long as your build will create the folder. The Console Output of a specific build contains logging information started by Publishing Javadoc that can be used to debug any problems.

Aggregate downstream test results

This feature allows you to pull the test results of this project and any downstream projects together. This is especially useful when long running test are set up as separate downstream projects. You can either let Hudson figure out all downstream projects automatically or supply a specific list of projects in the Jobs to aggregate input box.



☑ Aggregate downstream test results

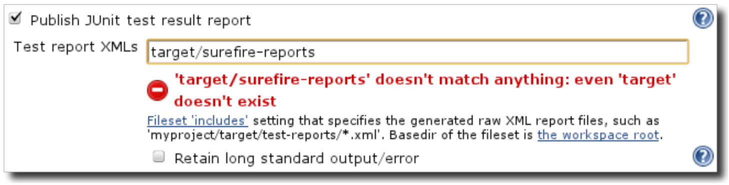
☐ Automatically aggregate all downstream tests

Jobs to aggregate

Figure 6.9: Configuring the test result aggregation post build action

Publish JUnit test result report

Activating this feature allows Hudson to interpret the JUnit test report format, produced by your test runs in the project and produce historic test result trends, a web interface for viewing the reports accessible from the project page as visible in Figure 6.8. The location of the produced xml files has to be specified in the text input box, which allows the use of patterns to find files in multiple sub folders of the project.In addition it is possible to retain the build log output by checking the Retain long standard output/error.



☑ Publish JUnit test result report

Test report XMLs: target/surefire-reports

'target/surefire-reports' doesn't match anything: even 'target' doesn't exist

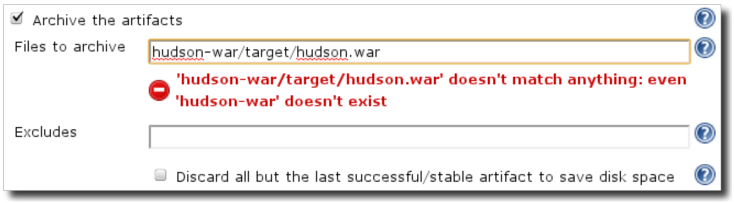
Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/*.xml'. Basedir of the fileset is the workspace root.

☐ Retain long standard output/error

Figure 6.10: Configuring the JUnit test result report post build action

Archive the artifacts

With this feature enabled Hudson will keep the specified artifacts and make them available on the web interface. Using a wild-card syntax in the Files to archive as well as the Excludes input boxes you can specify the artifacts that should be saved after each successful build.The artifacts of the last successful build are available on the project overview page as visible in Figure 6.12.



☑ Archive the artifacts

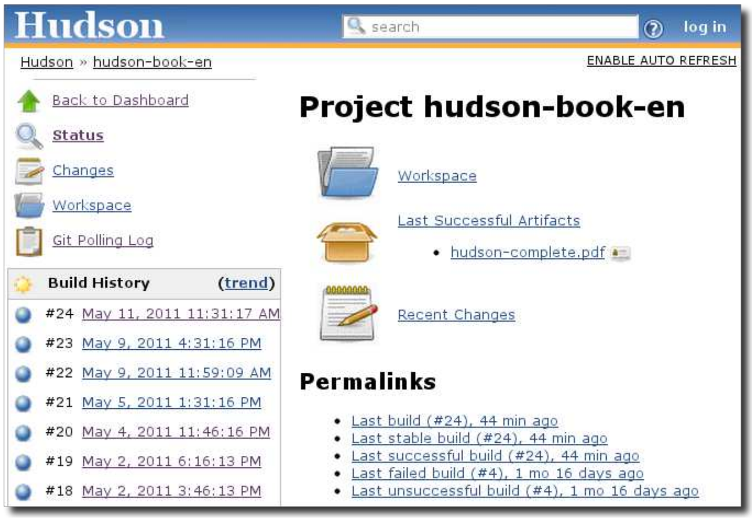
Files to archive: hudson-war/target/hudson.war

'hudson-war/target/hudson.war' doesn't match anything: even 'hudson-war' doesn't exist

Excludes:

☐ Discard all but the last successful/stable artifact to save disk space

Figure 6.11: Configuring the artifacts archival post build action



Hudson search log in

Hudson » hudson-book-en ENABLE AUTO REFRESH

Back to Dashboard
Status
Changes
Workspace
Git Polling Log

Build History (trend)

#24 May 11, 2011 11:31:17 AM
#23 May 9, 2011 4:31:16 PM
#22 May 9, 2011 11:59:09 AM
#21 May 5, 2011 1:31:16 PM
#20 May 4, 2011 11:46:16 PM
#19 May 2, 2011 6:16:13 PM
#18 May 2, 2011 3:46:13 PM

Project hudson-book-en

Workspace

Last Successful Artifacts

- hudson-complete.pdf

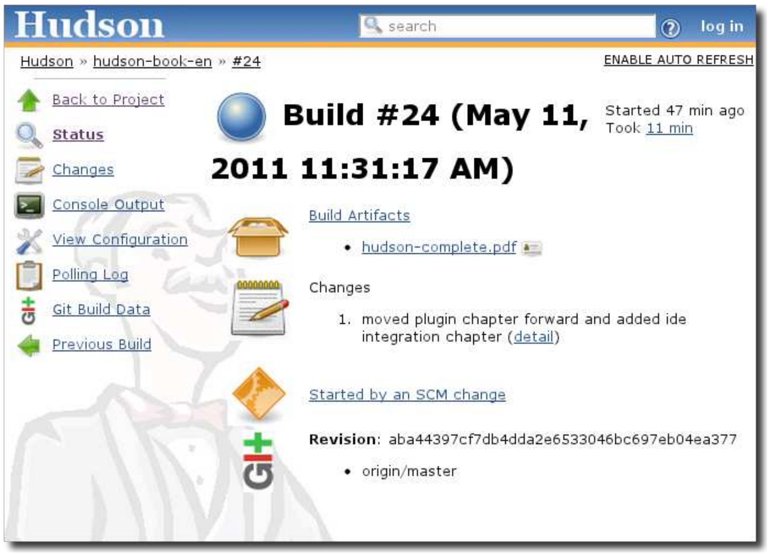
Recent Changes

Permalinks

- Last build (#24), 44 min ago
- Last stable build (#24), 44 min ago
- Last successful build (#24), 44 min ago
- Last failed build (#4), 1 mo 16 days ago
- Last unsuccessful build (#4), 1 mo 16 days ago

Figure 6.12: A project overview page for the Hudson book project with the latest build artifact available for download

In addition Figure 6.13 shows how the artifacts produced by a specific build can be accessed on build specific page.



Hudson search log in

Hudson » hudson-book-en » #24 ENABLE AUTO REFRESH

Back to Project
Status
Changes
Console Output
View Configuration
Polling Log
Git Build Data
Previous Build

Build #24 (May 11, 2011 11:31:17 AM)

Started 47 min ago
Took 11 min

Build Artifacts

- hudson-complete.pdf

Changes

1. moved plugin chapter forward and added ide integration chapter (detail)

Started by an SCM change

Revision: aba44397cf7db4dda2e6533046bc697eb04ea377

- origin/master

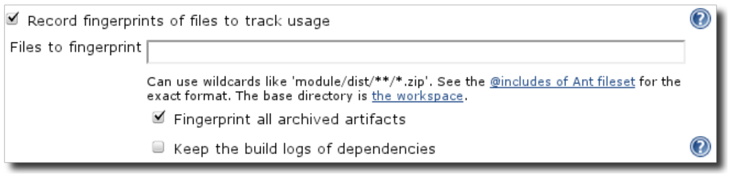
Figure 6.13: A build specific page for the Hudson book build with build artifact available for download

This feature is useful to make release artifacts like war, ear or zip files available for retrieval by other users and for archival purposes. Hudson can be used as the reference storage place for these artifacts that are in turn used e.g. for QA and production deployments or for distribution to customers.

If your artifacts are created by a Maven 3 based build using Maven 3 build steps, it might be a better option to activate the option Archive Maven 3 artifacts for better archiving robustness as documented above in the respective section.

Recording fingerprints of files to track usage

The feature to Record fingerprints of files to track usage can help you track down, where files are used and produced. It will allow you to determine the build number that created an artifact by looking at the fingerprint, which is a unique identifier for the file that Hudson creates and keep track of.

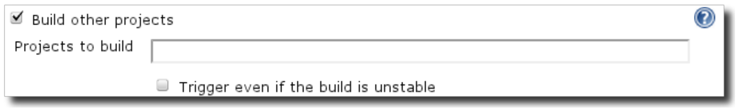


☑ Record fingerprints of files to track usage
Files to fingerprint
Can use wildcards like 'module/dist/**/*.zip'. See the @includes of Ant fileset for the exact format. The base directory is the workspace.
☑ Fingerprint all archived artifacts
☐ Keep the build logs of dependencies

Figure 6.14: Configuring the fingerprinting post build action

Build other projects

Building other projects after completion of the current project build, is one of the key features that allows you to set up chains of project builds. They can then all be small in focus and build time. However in the bigger picture you are able to to run a build for very large and complex systems. The input box Projects to build accepts a comma-separated list of projects to build together with a check box that allows you to trigger the dependent builds even if the current project build failed.



☑ Build other projects
Projects to build
☐ Trigger even if the build is unstable

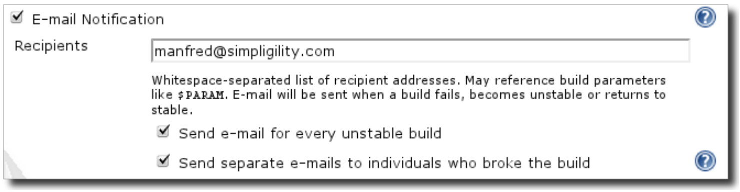
Figure 6.15: Configuring downstream builds post build action

Some examples for the usage of this feature are a main project triggering separate projects that invoke unit and/integration tests, shared libraries invoking server as well as client side application builds or build system plugin builds triggering all projects that use the plugin.

E-mail Notification

A very valuable post-build action for a continuous integration build is the sending of build notification emails.

Hudson can be configured to send out build failure notices to any email address. In most instances it will be best to send the notices to an email list allowing the potential recipients to opt-in and out as well as access archives. This email is one of the primary ways in which developers are notified of build failures. To configure this feature, check the check box next to E-mail Notification and then specify the recipients email addresses in Recipients separated by white-space. Selecting Send e-mail for every unstable build will instruct Hudson to send an email for each build even if a build experience consecutive failures. Selecting Send separate e-mails to individuals who broke the build will send email to all SCM committers that affected a build that broke. The email will be sent with the configuration specified in Section 3.8.


☑ E-mail Notification

Recipients: manfred@simpligility.com

Whitespace-separated list of recipient addresses. May reference build parameters like $PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.

☑ Send e-mail for every unstable build

☑ Send separate e-mails to individuals who broke the build

Figure 6.16: Configuring the email notification post build action

Notify that Maven dependencies have been updated
When activating this option Hudson will notify all projects with a build trigger configured to watch for Maven dependencies that have been updated (see Section 6.2.4). When the additional option Notify even when build is unstable is selected this notification will occur even when an unstable build ran. A side-effect of this could be that only some Maven dependencies have been updated, which could lead to further failures of the dependent projects.

6.2.6 Working with Cascading Projects

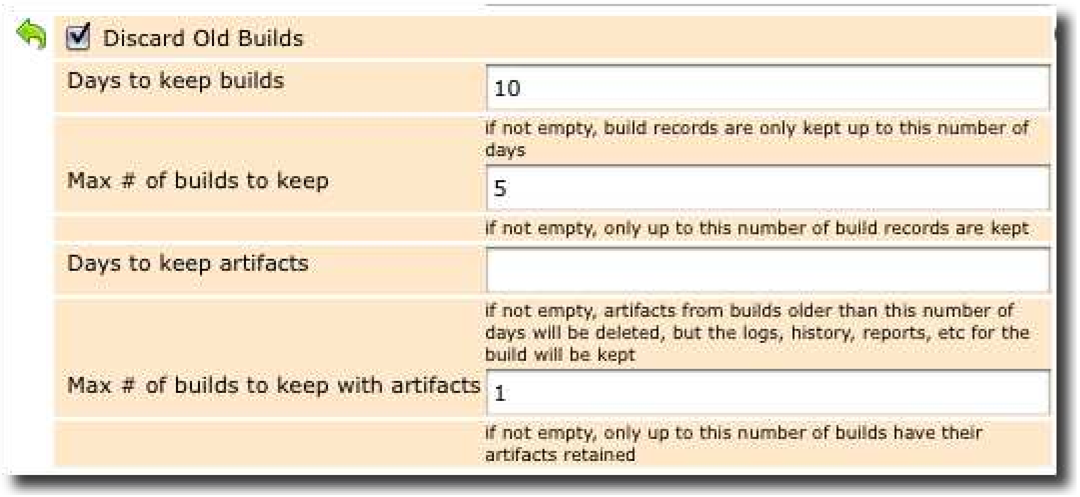
Support for cascading projects is an addition to Hudson that makes managing a large number of projects easier by implementing single inheritance of project setting from one project to one or many others.

Warning
This feature has been introduced with Hudson 2.2.0, so you will have to update to it or a newer version to take advantage of it.

The Cascading Project drop down in Figure 6.3 is populated with all projects the current project can inherit settings from. These have to be of the same type - free-style or multi-configuration and you will be prevented from introducing cyclic inheritance.

Select a project you would like to inherit settings from in the drop down and the complete project configuration will be populated with the configuration values of the parent project.

If you change a configuration for the child project and save the configuration any setting that differs from the parent project setting will be highlighted on the configuration screen and augmented with green arrow on the left side of the input as visible in Figure 6.17. This arrow will revert any changes and reapply the configuration settings of the parent project.



✓ Discard Old Builds	
Days to keep builds	10
	if not empty, build records are only kept up to this number of days
Max # of builds to keep	5
	if not empty, only up to this number of build records are kept
Days to keep artifacts	
	if not empty, artifacts from builds older than this number of days will be deleted, but the logs, history, reports, etc for the build will be kept
Max # of builds to keep with artifacts	1
	if not empty, only up to this number of builds have their artifacts retained

Figure 6.17: A Discard Old Builds configuration in a child project overriding the parent project configuration.

The parent and child projects are linked internal and an user will not be able to delete a parent project as long child projects exist.

The inheritance works for general project parameters as well as builders, publishers, triggers and SCM related settings. When updating a parent project parameters these changed values are propagated to all child projects that do not have the values overridden.

Examples for useful applications of this feature among others are:

- a parent project for a main branch of a project and children for all other branches in the SCM system
- combination of different scheduling and build parameters e.g. for build with unit tests only vs with full integrations tests vs a nightly regression build

6.3 Configuring Free-style Projects

In addition to the common project configurations we discussed prior, a free-style Hudson project has a section for the build definition that allows you to add individual build steps. Press the Add build step button in the Build configuration section to set up one or multiple steps that define your build with the choices displayed in Figure 6.18 and documented below.

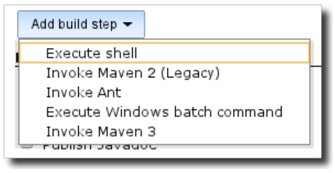


Figure 6.18: Adding a free-style project build step

Execute shell
: Executing a shell script as a build step is configured just like a Windows batch command. The unix convention to use the first line with a #! and the path to an executable allows you to write shell scripts in many available shell and scripting languages.

Invoke Maven 2 (Legacy)
: A Maven 2 based build step can be configured as documented in Section 7.4. We recommend to use the Maven 3 invocation where possible.

Invoke Ant
: Invoking an Ant target can be added just like a shell script task, but with more configuration options. Find out more on how to configure Ant invocations in Section 8.2

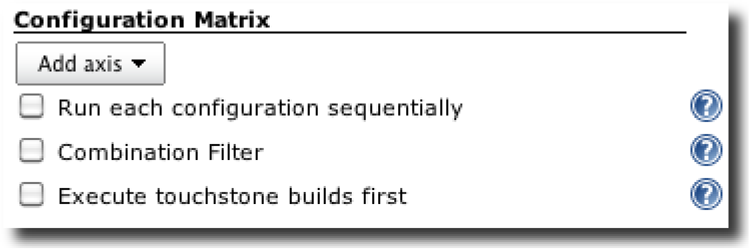
Execute Windows batch command
: The Command input allows you to specify the the name of the batch file to execute. The script has to return an error level value of zero to be recognized as a build success by Hudson. The script will be executed with the current workspace of the project as the directory. A number of environment variables about Hudson and the current project are passed to the execution of the script and can be used from within the script.

Invoke Maven 3
: A Maven 3 based build step can be configured as documented in Section 7.3

6.4 Configuring Multi-Configuration Projects

Multi-configuration projects have to be configured as such when initially creating the project as visible in Figure 6.1. A multi-configuration project is a free-style project with additional parameters called axis defined. These axis values create a matrix of possible combinations. Each of these combinations is a possible build execution with the specific combination of parameters passed to the build.

Figure 6.19 and Figure 6.20 show the general Configuration Matrix section as well as an example of some axes defined. A pre-configured axis available offers the choice of JDK used for the project build. In the example two further axes are defined. For for a database backend used for the build and another for the type of build to execute. Axis can be added by creating the Add axis button and removed with the Delete button.



Configuration Matrix

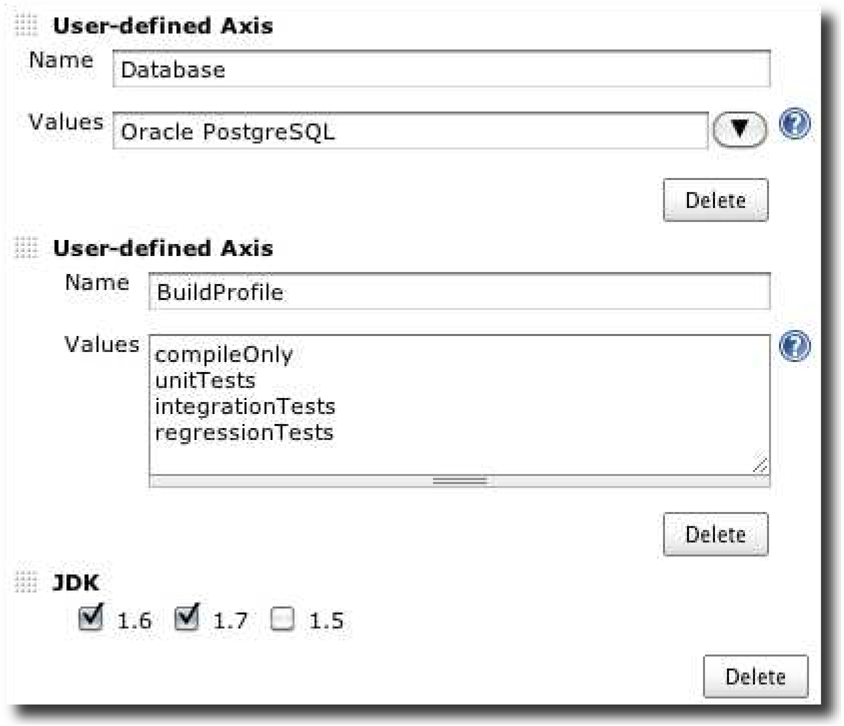
Add axis ▾

☐ Run each configuration sequentially

☐ Combination Filter

☐ Execute touchstone builds first

Figure 6.19: Configuration matrix parameters for multi-configuration project



User-defined Axis

Name: Database

Values: Oracle PostgreSQL

Delete

User-defined Axis

Name: BuildProfile

Values:
compileOnly
unitTests
integrationTests
regressionTests

Delete

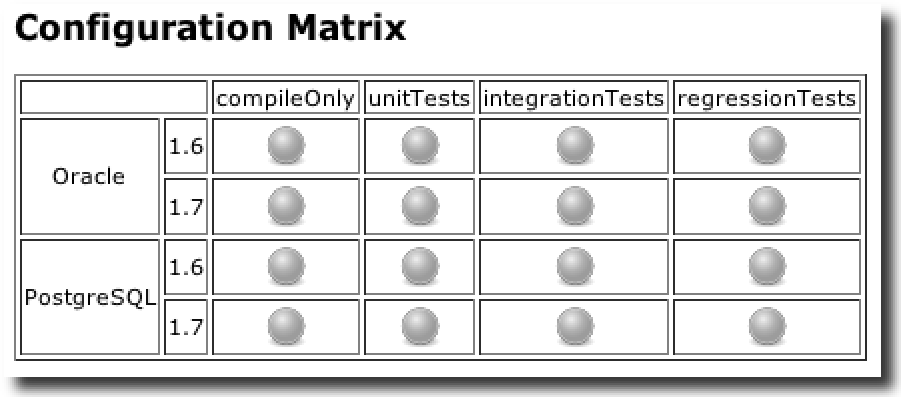
JDK

☑ 1.6 ☑ 1.7 ☐ 1.5

Delete

Figure 6.20: Axis definition for multi-configuration project

All these axis values create a configuration matrix on the project overview page as visible in Figure 6.21.



Configuration Matrix

		compileOnly	unitTests	integrationTests	regressionTests
Oracle	1.6	●	●	●	●
	1.7	●	●	●	●
PostgreSQL	1.6	●	●	●	●
	1.7	●	●	●	●

Figure 6.21: Configuration matrix display for multi-configuration project

The most important configuration steps is to define the axes that you will use for your project. The user-defined axis definition allows you to define a name and multiple values. These values will be passed to your build execution as variables and you have to ensure in your build configuration that they are e.g. used as Maven profile names in the a build step or passed in as properties and picked up to select the correct database connection. The values have to be separated by white-space in the values input field.

In addition you can configure the following settings.

Run each configuration sequentially
: Selecting this option will cause Hudson to build each combination of axis values as sequential build invocations rather than executing them concurrently. This is especially useful if concurrent execution could cause conflicts when accessing resources needed by the build like a central database server.

Combination Filter
: The combination filter input field allows you to define valid axis value combinations with a Groovy expression. Only expressions returning true will be executed.

Execute touchstone builds first
: By defining a filter you can set up a touchstone build. It will be executed first and the required result will have to be met before any further combinations of the matrix configuration are executed.

Chapter 7

Working with Apache Maven Builds

Apache Maven is the most widely used build tool for Java-based applications and beyond. It has excellent support in Hudson and is employed by most Hudson users.

7.1 Installing and Configuring Apache Maven

In order to use Apache Maven for your project build you will have to configure one or more installs, as documented in Section 3.5. Among the many factors that influence your choice of installation method you might want to consider the following:

One or more Maven versions?
Depending on the variety of projects you aim to be building on Hudson, you might need to have more Maven versions for building your projects available. This can be helpful to allow for a staged upgrade with one project at a time to minimize down times due to broken builds with newer versions as well.

Variety of operating systems in build cluster
Your build cluster might be using different operating system versions, which can make some installation methods harder or impossible to use. Reasons for using different operating systems in a cluster can be the need to run tests on them, facilitating pre-existing e.g. desktop workstations at night, the need to build native packages on their own platform or simply the fact that a certain build can only be done on a specific operating system.

Control software via Hudson or something else?
While Hudson has built in support to manage the JDK install as well as Apache Ant and Maven, your builds can require any number of further software to work. If your build requires a high number of these additional software packages installed, it will be increasingly difficult to keep your build cluster setup configured with all required

components. If you or someone else in your company already uses a provisioning software or virtual machine or operating system snapshots for similar purposes and it would make sense to reuse the existing facilities and setup. Otherwise it might be worth getting a provision system suitable to your needs set up.

Available expertise
Your team or yourself might have pre-existing expertise with native package management, provisioning software, virtual machine snapshots or Hudson itself that can be an influencing factor on how to set up your Maven installs.

Control Maven installation
Some installation methods like using a pre-existing Maven install from an operating system package rely on third parties to create these installations at first. While convenient this implies a loss of control that might not be desired and has to be weighed against the additional effort of different install methods.

With assessing these influencing parameters you will be able to derive a strategy for your Maven install(s), that will work across the different machines in your build cluster and for all your projects. This strategy can potentially involve different installation methods. When using the recommended Maven 3 integration no Maven install is necessary, since a Maven 3 install is embedded with the plugin. Otherwise a minimal setup of one Maven install can be used to set up your first project, that is built with Maven on Hudson.

7.2 Selecting Components of your Maven and Hudson Integration

The recommended way to configure a Maven build on Hudson is to create a free-style software project as documented in Chapter 6 and then creating one or more build steps that Invoke Maven 3. More details on the specific configuration options and more can be found in Section 7.3.

If you require Maven 2 for your build for some legacy reason, even though Maven 3 is a higher performing, drop-in replacement you can fall back to using the Invoke Maven 2 (Legacy) build step documented in Section 7.4. Even though this option exists, we recommend moving forward and migrating to Maven 3 and the Maven 3 build step.

Finally your Hudson install may contain a separate project type labelled Build a Maven 2/3 project (Legacy) available when creating a new project. This feature has numerous issues with the use of different Maven versions and use cases, especially Maven 3, and is not supported in Hudson. We recommend not to use this job type. If you do not need this job type anymore after migrating to the Maven 3 build step and want to avoid the accidental creation of projects with this type, you should disable the `Hudson:: Maven(legacy) :: Plugin`. Read more about managing your plugins in Chapter 5.

Both the Maven 2/3 project type as well as the Maven 2 build step are part of Hudson as legacy components and are not the focus of active development and improvements. If your existing jobs use either of these we recommend migrating to Maven 3 build step based free-style projects.

In order to be able to keep existing legacy build around without cluttering to user interface or have inexperienced users create builds with the legacy systems, you can enable the blacklist plugin to disable certain views within Hudson without completely disabling the still required plugins.

7.3 Details of Configuring Maven 3 Build Options

After configuring the general project options as documented in Chapter 6, you can configure one or more build steps. To add a build step, click on the Add build step button as shown in Figure 7.1. To configure a Maven build, select Invoke Maven 3.

The basic and advanced options for configuring a Maven 3 build steps are shown in Figure 7.1 and Figure 7.2 . When adding a new Maven 3 build step all parameters are pre-populated with the default parameters defined for Maven 3 build steps. These are configured in the global Hudson configuration documented in Chapter 3 in Section 3.6.

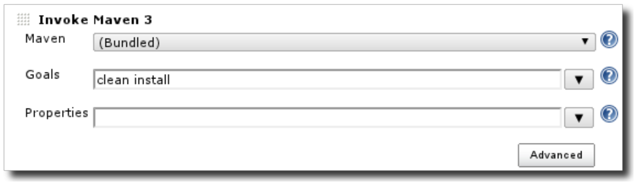
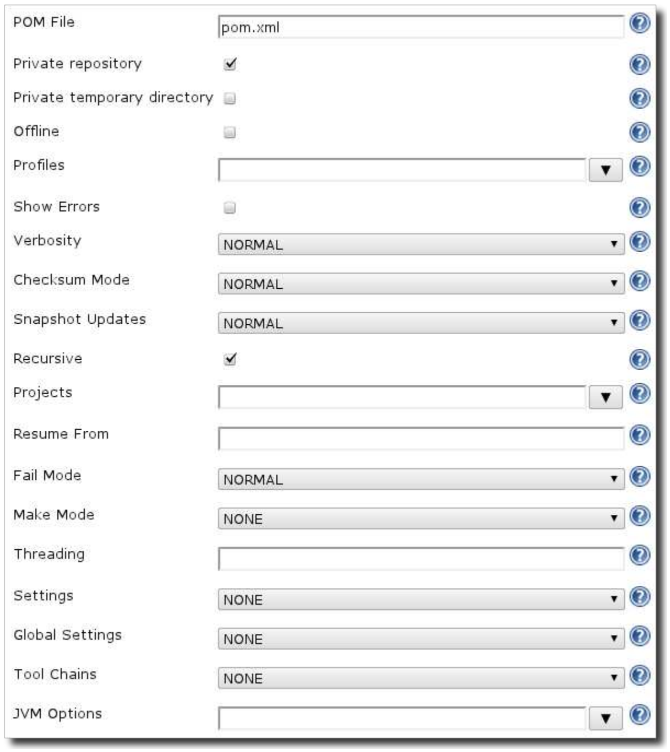

Invoke Maven 3
Maven (Bundled)
Goals clean install
Properties

Figure 7.1: Configuring a Maven 3 Build Step - Default Options



POM File	pom.xml
Private repository	✓
Private temporary directory	☐
Offline	☐
Profiles	
Show Errors	☐
Verbosity	NORMAL
Checksum Mode	NORMAL
Snapshot Updates	NORMAL
Recursive	✓
Projects	
Resume From	
Fail Mode	NORMAL
Make Mode	NONE
Threading	
Settings	NONE
Global Settings	NONE
Tool Chains	NONE
JVM Options	

Figure 7.2: Configuring a Maven 3 Build Step - Advanced Options

Maven Version

Your Hudson installation may have one or more Maven 3 installations configured as part of the global Hudson configuration. This drop-down allows you to specify the version of Maven 3 to use with the current build step. By default the Maven 3 integration will use a bundled version of Maven 3, that is installed as part of the Maven 3 integration itself.

Goals

This field allows you to specify the goals and phases you would like to use for the Maven 3 invocation, separate by spaces and in order of desired invocation just like you would use Maven 3 on the command-line. Typically these would be `clean install`, but depending on your project and build step others may apply. The plugin with automatically show the Maven version and operate in batch mode equivalent to the `-V` and `-B` options. It is also to specify other specific Maven to command-line options like `-P` for profile selection, selection of a specific `settings.xml` file, `-X` for debugging logging and so on. However this is not recommended since there are specific configuration settings for most parameters in the Advanced section. If you specify them in their dedicated option it will be easier to carry out programmatic changes e.g. via scripts in the Hudson console across all projects defined in your Hudson instance.

Properties

You can pass one or more properties to the Maven process. This field accepts a list of properties with lines in a key=value format. These properties will be passed into the Maven build step invocation using the standard way of passing properties of -Dkey1=value1 -Dkey2=value2.

POM File
If your project uses the standard `pom.xml` file-name, there is no need to specify a POM. If your project uses a POM with an alternative name or path other than directly in the project root, you can specify that file name and path here. This setting is equivalent to the command-line option -f pomfilepath or—file pomfilepath. The path specified has to be relative to the project root. It is also necessary to specify the path to the POM file here if you SCM setup causes the POM file to be located in a sub-folder and not in the top level workspace folder.

Private repository
By default a Maven invocation will use a private repository in the project workspace in a `.maven/repo` folder. This ensures that other project builds do not have any side effects. However it can cause considerable usage of storage space, which consequently should be monitored carefully. If this option is disabled the standard Maven repository location in the current users home directory will be used. The user will the operating system user running Hudson and therefore invoking Maven 3 via the plugin.

Private temporary directory
When this option is activated the java environment variable `java.io.tmpdir` will be set to a folder `.maven/tmp` in the project-specific workspace. This is useful when your build accesses the temporary directory for storage of artifacts or any temporary files e.g. used while running tests. When using the option with builds that produce large amounts of data in the temporary folder it is important to monitor the size of the folder and potentially add a clean up routine to the host operating system regular scheduled jobs.

Offline
Activating this option causes Maven to be run with the `-o` offline option enabled and it will therefore not access any remote repositories.

Profiles
Adding a comma or space separated list of profile names causes the Maven 3 integration to pass them to the invocation with the `-P` parameter.

Show Errors
Enabling the Show Errors option is equivalent to use the `-e` command-line parameter, which will cause Maven to output any errors in the console.

Verbosity
Configure the verbosity of the log output to the console by Maven to be at normal, quiet or debug levels. These levels are equivalent to no option and the `-q` and `-X` options for Maven command-line invocation.

Checksum Mode
Configure the strictness of the check-sum validation when downloading artifacts from repositories to be at normal, lax or strict levels. These levels are equivalent to no option and the `-c` and `-C` options for Maven command-line invocation.

Snapshot Updates
The Snapshot Updates option provides control over the way Maven treats `SNAPSHOT` artifacts. The NORMAL activates the standard Maven behaviour, where as FORCE and SUPPRESS will activate the `-U` and `-nsu` options.

Recursive
Just like for normal Maven invocation this option is activated by default, which means that nested modules in a multi-module project will be build. Deactivating this feature is equivalent to the non-recursive command-line option `-N`.

Projects
The Projects option allows you to specify the projects that should be added to the reactor during build. You can either specify them by the relative path in your project workspace or by `artifactId` and optionally `groupId` in a `groupId:artifactId` format. The equivalent command-line option is `-pl`.

Resume From
The Resume From uses the same syntax as the Projects and sets the `-rf` command-line option to resume the build from the specified artifact.

Fail Mode
The Fail Mode option supports the modes NORMAL, FAST, AT_END and NEVER that determine how your Maven build proceeds in case of any failures. The equivalent command-line options are no option, `-ff`, `-fae` and `-fn`. This can have considerable impact on the load on your Hudson server e.g. by not proceeding past failed tests but instead failing the build so that developer can fix it before a long running build needs to be kicked off again.

Make Mode
The Make Mode option can be used to enable Make-like build behaviour of Maven. The options are the default behaviour equivalent to NONE and DEPENDENCIES, DEPENDENTS and BOTH respectively `-am`, `-amd` and `-am -amd`.

Threading
This input takes the value for the `-T` command-line option that enables the experimental support for parallel builds in Maven 3. A value of `4` enables four threads for the build. A value of `2.5C` would enable 2.5 threads per CPU core. When activating this feature, keep the experimental nature of this feature as well as the not yet wide spread support for this feature in the various plugins in mind. Read on the Maven Wiki to find out more about this feature and the current status.

Settings
This option corresponds to the `-s` command-line option for Maven 3 and supplies a Maven 3 build step with a custom settings configuration file. The drop down for this field is populated with the Maven 3 Settings files configured in the Hudson Maven 3 Configuration page as shown in Figure 3.10.

Global Settings
This option corresponds to the `-gs` command-line option for Maven and isn't used as frequently as the `-s` option. This option allows you to reference a custom global configuration file that is an alternative to the global settings file that ships with Apache Maven.

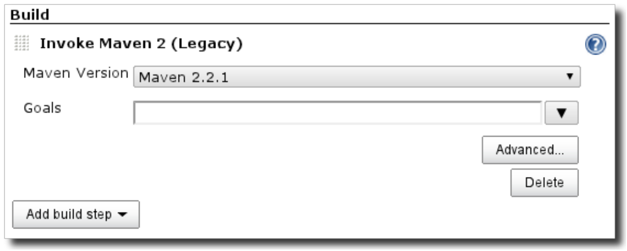
Tool Chains
Tool chains for Maven 3 build steps can be configured globally for your Hudson install as documented in Section 3.10. The drop allows you to select one of these configured tool chains for to be used for the build step. Tool chains are a very useful, but lesser known feature of Maven and more documentation can be found in the mini guide on the Apache web site.

JVM Options
If your build requires specific JVM options, they can be set in this field. The options are passed straight through

as MAVEN_OPTS and use the normal java command-line options syntax. A common configuration for complex builds is to specify a larger memory for the JVM running Maven via -Xmx1024m.

7.4 Details of Configuring Maven 2 (Legacy) Build Options

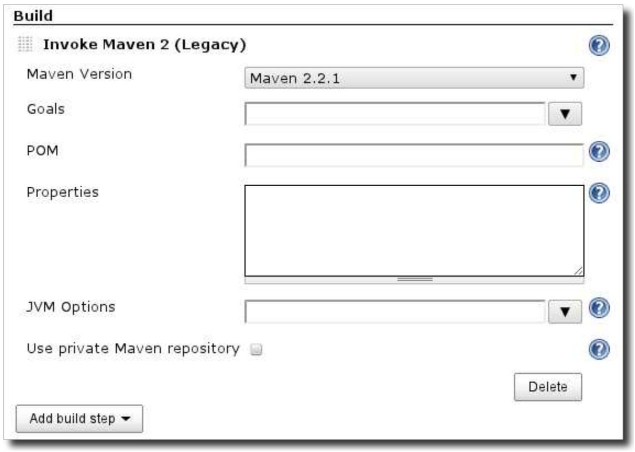
If for some reason you are still using Maven 2 and cannot upgrade to Maven 3 and therefore take advantage of the advanced performance and features of Maven 3 and the Maven 3 integration of Hudson offers you, you should use the Maven 2 (Legacy) integration in a free-style project build step. After configuring the general project options (see Chapter 6), you can configure one or more build steps. To add a build step, click on the Add build step button as shown in Figure 7.3. To configure a Maven build, select Invoke Maven 2 (Legacy).



Maven Version: Maven 2.2.1

Figure 7.3: Configuring Project Build Options

The basic and advanced options for invoking a Maven 2 (Legacy) target are shown in Figure 7.4.



Maven Version: Maven 2.2.1

Figure 7.4: Advanced Configuration of a Maven 2 (Legacy) Build Step

Maven Version
: Your Hudson installation may have one or more Maven installations configured as part of the global Hudson configuration. This drop-down allows you to specify the version of Maven for usage with the current build step.

Goals
: This field allows you to specify the command-line parameters used for the Maven invocation. These are phases, plugin goals as well as specific Maven command-line options like `-P` for profile selection, selection of a specific `settings.xml` file and so on.

POM
: If your project uses the standard `pom.xml` file-name, there is no need to specify a POM. If your project uses a POM with an alternative name or path other than directly in the project root, you can specify that file name and path here. This setting is equivalent to the command-line option -f pomfilepath or—file pomfilepath. The path specified has to be relative to the project root.

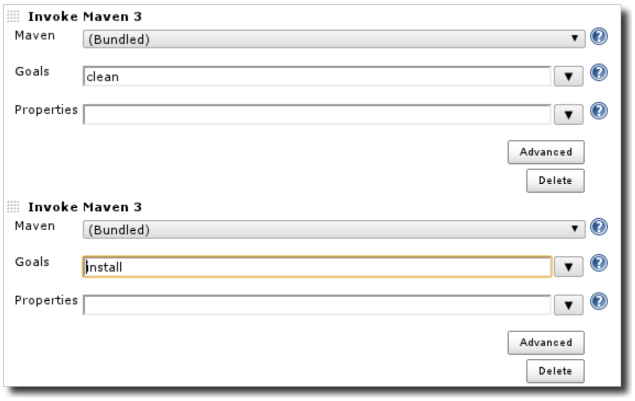
Properties
: You can pass one or more properties to the Maven process. This field accepts a list of properties with lines in a key=value format. These properties will be passed into the Maven build step invocation using the standard way of passing properties of -Dkey1=value1 -Dkey2=value2.

JVM Options
: If your build requires specific JVM options, they can be set in this field. The options are passed straight through as MAVEN_OPTS and use the normal java command-line options syntax. A common configuration for complex builds is to specify a larger memory for the JVM running Maven via -Xmx1024m.

Use private Maven repository
: By default a Maven invocation will use a local repository in the current users home directory taking any further repository configuration done in settings.xml into account. Depending on the necessary separation of the different jobs running on Hudson it can be useful to have a separate Maven repository for each project. Activating this feature will cause the creation of a separate Maven repository in a `.repository` folder in the projects workspace. This can cause considerable usage of storage space, which consequently should be monitored carefully.

The standard way of invoking a Maven build is to run the clean and install life cycle phases in a command-line call like mvn clean install. This can be easily achieved by adding a Maven build step and adding clean install as a Goals parameter. If so desired these two life cycle phases can also be invoked separately by creating two Maven build steps with separate clean and install parameters. This would cause two separate invocations of Maven in sequence equivalent to mvn clean; mvn install Breaking up the invocations allows you to add further build steps in between and build an arbitrarily complex sequence of Maven and other invocations with completely separate parameters and so on.

You could for example slip a plugin goal invocation in between the clean and install invocations, that prepares the execution environment for your build for example by setting up a test environment like an emulator or a specific database and content. Since you can do this in a separate build step you can invoke a shell script or an Ant target for such tasks, in case it is not automated via a Maven plugin or configuration.



Invoke Maven 3

Maven (Bundled)

Goals clean

Properties

Invoke Maven 3

Maven (Bundled)

Goals install

Properties

Figure 7.5: Configuring two build steps as top-level Maven invocations

Chapter 8

Working with Apache Ant Builds

8.1 Installing Apache Ant for Hudson

The general configuration for Apache Ant installs on Hudson is available in Section 3.4. You can read about influencing factors for your Apache Maven install, that apply to your strategy for installing Ant as well in Section 7.1.

A common scenario for Apache Ant installs is the requirement of a build to have access to Ant tasks as provided in Ant-Contrib like For or If or to have some required libraries like JSch for secure copying with scp available on the Ant class-path. Custom developed Ant tasks can be needed in a similar fashion. A convenient way to achieve this, is the creation of a custom archive containing the Ant install as well as additional libraries necessary. This archive can then be used with the `Extract from *.zip/*.tar.gz` option to get a fully working Ant and dependencies installed on all build cluster nodes.

8.2 Configuring Apache Ant Builds

After configuring the general project options as documented in Chapter 6, you can configure one or more build steps. To add a build step, click on the Add build step button and select Invoke Ant.

The basic and advanced options for invoking Ant are shown in Figure 8.1.

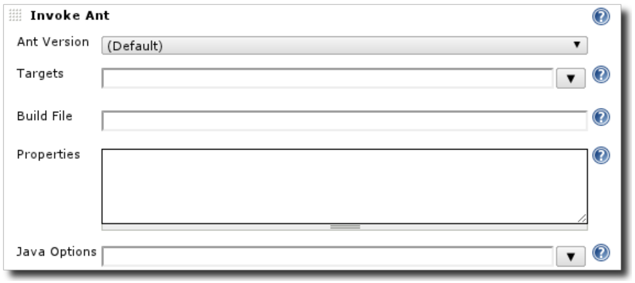


Figure 8.1: Configuring an Ant Build Steps

Ant Version
: Your Hudson installation may have one or more Ant installations configured as part of the global Hudson configuration. This drop-down allows you to specify the version of Ant for usage with the current build step.

Targets
: This field allows you to specify the command-line parameters used for the Ant invocation. These are all options and targets supported by the specified Ant and the current Build File. If nothing is specified in this field, the build step will invoke Ant without parameters. With a default target specified in the build file, this can be sufficient for a full build depending on your build file.

Build File
: If your project uses the standard `build.xml` file-name, there is no need to specify a Build File If your project uses a build file with an alternative name or path other than directly in the project root, you can specify that file name and path here. This setting is equivalent to the command line option -f buildfilepath. The path specified has to be relative to the project root.

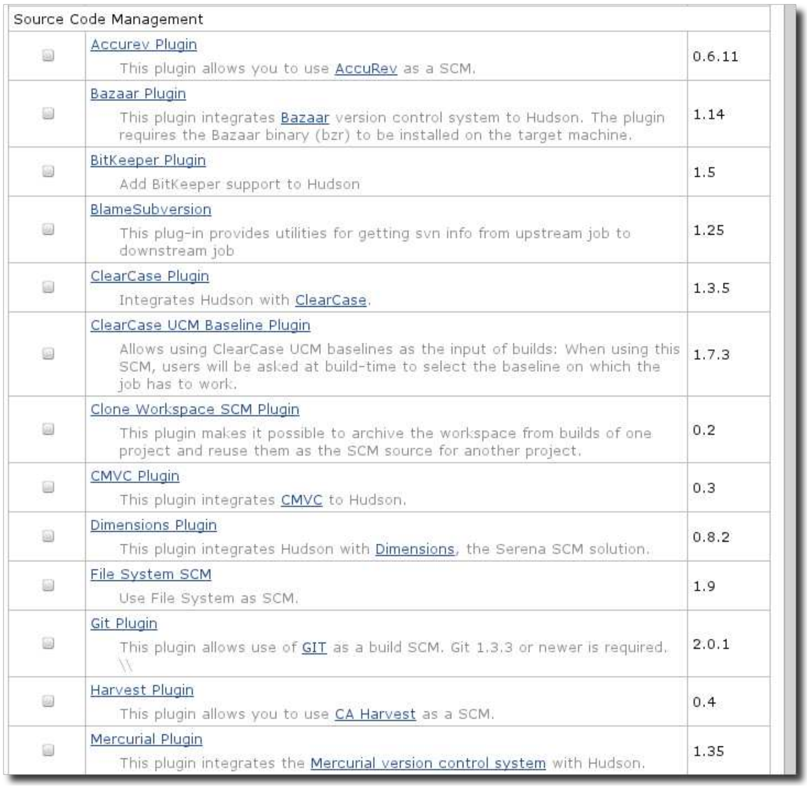
Properties
: You can pass one or more properties to the Ant process. This field accepts a list of properties with lines in a key=value format. These properties will be passed into the Ant build step invocation using the standard way of passing properties of -Dkey1=value1 -Dkey2=value2.

Java Options
: If your build requires specific Java options, they can be set in this field. The options are passed straight through as ANT_OPTS and use the normal java command-line options syntax. A common configuration for complex builds is to specify a larger memory for the JVM running Ant via -Xmx1024m.

Chapter 9

Working with Source Control

One of the most important parts of the Hudson project configuration are the settings that connect your Hudson project to source control. Any software development project should be managed in a source control management system, many of which are open source software and have large user communities. Hudson has support for all common SCM systems as well as many of the less popular ones. Most likely you will find support for your SCM already installed or available as a plugin for installation in the Source Code Management section of available plugins in the plugin management as displayed in Figure 9.1. Read more about available plugins and their management in Section 5.2 in Chapter 5.



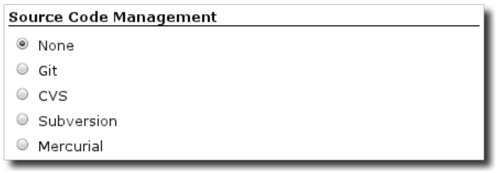
Source Code Management		
☐	Accurev Plugin This plugin allows you to use AccuRev as a SCM.	0.6.11
☐	Bazaar Plugin This plugin integrates Bazaar version control system to Hudson. The plugin requires the Bazaar binary (bzr) to be installed on the target machine.	1.14
☐	BitKeeper Plugin Add BitKeeper support to Hudson	1.5
☐	BlameSubversion This plug-in provides utilities for getting svn info from upstream job to downstream job	1.25
☐	ClearCase Plugin Integrates Hudson with ClearCase.	1.3.5
☐	ClearCase UCM Baseline Plugin Allows using ClearCase UCM baselines as the input of builds: When using this SCM, users will be asked at build-time to select the baseline on which the job has to work.	1.7.3
☐	Clone Workspace SCM Plugin This plugin makes it possible to archive the workspace from builds of one project and reuse them as the SCM source for another project.	0.2
☐	CMVC Plugin This plugin integrates CMVC to Hudson.	0.3
☐	Dimensions Plugin This plugin integrates Hudson with Dimensions, the Serena SCM solution.	0.8.2
☐	File System SCM Use File System as SCM.	1.9
☐	Git Plugin This plugin allows use of GIT as a build SCM. Git 1.3.3 or newer is required. \\	2.0.1
☐	Harvest Plugin This plugin allows you to use CA Harvest as a SCM.	0.4
☐	Mercurial Plugin This plugin integrates the Mercurial version control system with Hudson.	1.35

Figure 9.1: The beginning of the list of Source Code Management plugins

By default Hudson has support for Git, Subversion and CVS pre-installed. This chapter will document usage of the respective Hudson plugins as well as the plugin supporting the popular open source SCM system Mercurial.

Each of these plugins can be configured in the global Hudson configuration setting in its specific section added by the plugin.

After the global settings for the desired source management system are configured, you can configure the project-specific settings. Simply load the project's configuration page and scroll down to the Source Code Management section. In this section, you must then select one of the radio buttons for the source code management system you are using as visible in Figure 9.2


Source Code Management

(●) None
( ) Git
( ) CVS
( ) Subversion
( ) Mercurial

Figure 9.2: Selecting an SCM in the project configuration

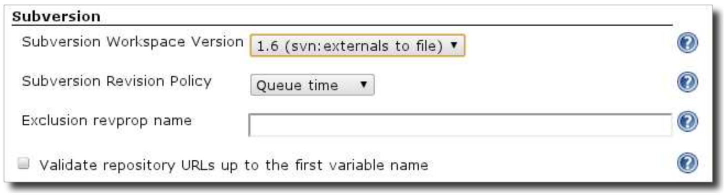
After this selection you will be able to configure the parameters specific to the selected SCM and the current project.

9.1 Configuring Subversion

The Subversion Plugin for Hudson and therefore support for the popular Subversion SCM system is part of the default install of Hudson. It is therefore not necessary to perform any further plugin installation to use Subversion for your project.

9.1.1 Global Subversion Configuration

In order to use Subversion successfully, you need to set up the global configuration in the Hudson Server configuration screen displayed in Figure 9.3.


Subversion

Subversion Workspace Version: 1.6 (svn:externals to file)
Subversion Revision Policy: Queue time
Exclusion revprop name:
[ ] Validate repository URLs up to the first variable name

Figure 9.3: Global Subversion Configuration

The following parameters need to be configured as desired:

Subversion Workspace Version
Subversion uses different formats for storing data in a checked out location. Ideally you should have the same

Subversion version installed on the SCM server as well as on the Hudson server and specify that version here. If your Hudson project tasks only require read access to the Subversion repository it is safe to use a higher version on the Hudson server and specify it here. However if you are automating a release process or any other tasks that will write to the Subversion repository e.g. by creating tags or branches or editing files and checking them in, you should make sure to use the same format on the Hudson server as on the Subversion server since mismatches can produce problems in the Subversion repository and potentially break expected behaviour. An important issue related to this setup is that you can not support different Subversion servers with different versions accessed from one Hudson instance. It is advisable to update the Subversion servers and Hudson installed clients before proceeding.

Subversion Revision Policy
The default revision policy is `Queue time`, which will cause a build to be run off the revision present in the repository when the job is added to the Hudson build queue. The `Build time` policy one the other hand will use the revision in the repository found when the build actually starts . The `Head revision` policy will use the HEAD revision in the repository. Finally these settings are overridden if a revision is specified in the subversion URL or as a revision parameter in a parameterized build.

Exclusion revprop name
This parameter can be used to cause the plugin to exclude revisions with the specified revision property from triggering new builds. This is useful for builds that cause a commit so that this commit done by Hudson will not in turn trigger the execution of another build. The commits carried out by Hudson as part of the build have to be configured to use the same property.

Validate repository URLs up to the first variable name
With this setting activated subversion URLs will only be validated up to the first variable. A variable in a URL would be preceded by a `$` character.

9.1.2 Project-Specific Subversion Configuration

Selecting Subversion under the Source Code Management section will display the configuration options shown in Figure 9.4. Clicking on the Advanced button will reveal the advanced configuration parameters shown in Figure 9.5.

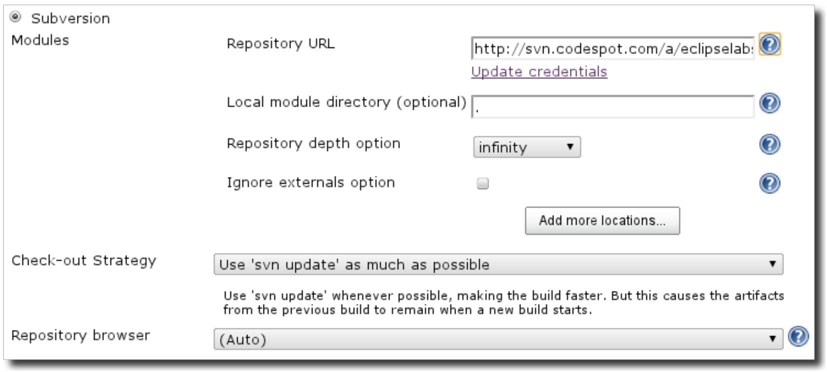
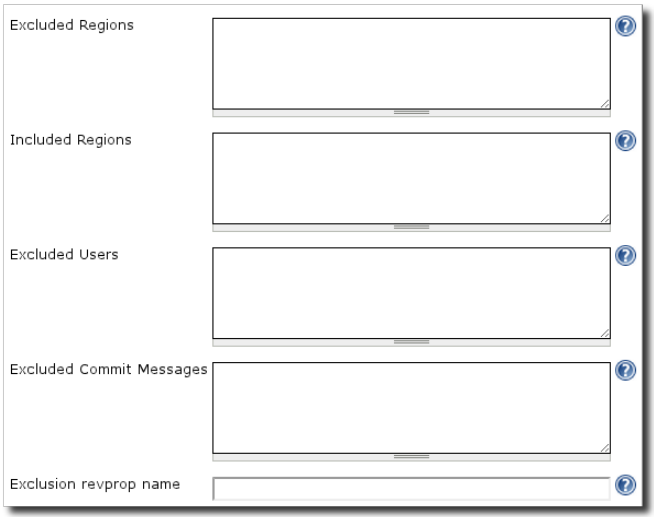

● Subversion
Modules
Repository URL: http://svn.codespot.com/a/eclipselab:
Update credentials
Local module directory (optional): .
Repository depth option: infinity
Ignore externals option: ☐
Add more locations...
Check-out Strategy: Use 'svn update' as much as possible
Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.
Repository browser: (Auto)

Figure 9.4: Configuring project-specific Subversion settings



Excluded Regions	
Included Regions	
Excluded Users	
Excluded Commit Messages	
Exclusion revprop name	

Figure 9.5: Configuring advanced project-specific Subversion settings

The following parameters can be configured:

Modules
: Hudson can check out one or more Subversion modules from a Subversion repository. To configure a repository location, supply a Subversion URL in the Repository URL text field. This field supports Subversion repository URLs including revisions specified with `@number` as well as subversion keywords for revisions and dates. The link Update credentials navigates you to the SVN authentication screen documented below.

If you are checking out more than one Subversion module, you can also supply a Local module directory that Hudson will check out the specified module to. If you leave the Local module directory blank, Hudson will check out the specific module to the root of the project's workspace.

The Repository depth option allows you to specify the depth for the checkout of this module, with the default being+infinity+, which means that all nested directories of the repository will be checked out. This is useful to limit the size and scope of your checkout.

You can add a new module with the Add more locations button and remove it with the Delete button.

Check-out Strategy
: The Check-out Strategy option determines the subversion commands issues prior to starting a build.

The default value of Use *svn update* as much as possible will cause the least load on the Subversion server by only issuing an update command on top of the existing checkout in the project workspace.

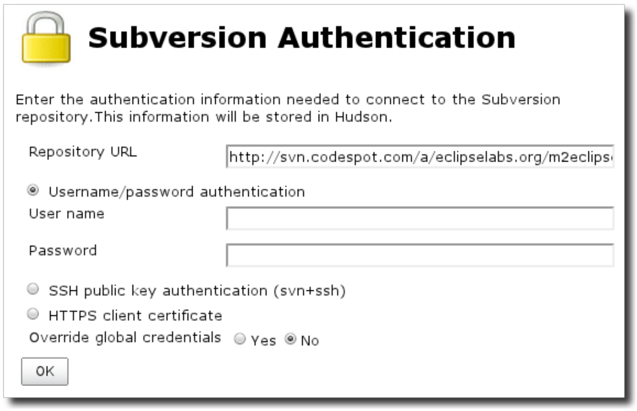
The option Use *svn update* as much as possible, with *svn revert* before update will do minor cleanup of the workspace by reverting any local modifications.

The option Emulate clean checkout by first deleting unversioned/ignored files, then *svn update* provides a good compromise between performance and thoroughness, since it closely resembles a clean checkout without the performance and load implications for the Subversion server as well as Hudson.

Finally the option Always check out a fresh copy will delete the workspace content and perform a fresh checkout for each build. Choosing this option should be considered carefully since it can put a significant load on the Subversion and Hudson servers.

Repository Browser
Hudson supplies valuable information about a build and about the SCM activity and changes that contribute to a specific build. When Hudson displays information about a Subversion commit or a file that has been modified, it can be configured to link to a Repository browser, which provides a rich web view of the source code repository. Hudson supports the repository browsers CollabNet, Sventon 2.x, ViewSVN, FishEye, WebSVN and Sventon 1.x as options in the the drop-down. The (Auto) option will attempt to automatically detect the used browser.

You can navigate to the Subversion Authentication screen by clicking the Update credentials link, which will display as visible in Figure 9.6.


Subversion Authentication

Enter the authentication information needed to connect to the Subversion repository.This information will be stored in Hudson.

Repository URL: http://svn.codespot.com/a/eclipselabs.org/m2eclips

[✓ Username/password authentication]
User name:
Password:

SSH public key authentication (svn+ssh)
HTTPS client certificate
Override global credentials: Yes [✓ No]

Figure 9.6: Configuring subversion authentication Subversion settings

The subversion plugin can be configured to use authentication with:

User name/password authentication
Provide the user name and password in the supplied input fields.

SSH public key authentication (svn+ssh)
The Private Key control allows you to upload a key file for which you can provide the User name in the respective input field. If you key is encrypted with a password you need to add it in the Pass phrase input.

HTTPS client certificate
For HTTPS based authentication you can upload a Public Key Cryptography 12 (PKCS12) file and provide a password in the provided controls.

By default the above configuration will override any global configuration. This behaviour can be deactivated by selecting no in the Override global credentials option.

Beyond these basic configuration options the Subversion plugin supports advanced options to provide more parameters to your Subversion commands used for the build.

Excluded Regions, Included Regions
These fields provide you with the option to either specifically include or exclude files and directories to determine if a build should be triggered. If Included Regions is set, Hudson will only trigger a build if a matching file has been altered. If Excluded Regions is set, Hudson will not trigger a build if a file matching an excluded pattern is matched. These options are useful if you are only interested in a subset of files and directories contained in a Subversion module to trigger a build on Hudson. An example would be if documentation files contained in the repository should not trigger a new build.

Both parameters support usage of regular expression patterns to specify the desired files as well as multiple lines to configure larger sets of files and directories to match.

Excluded Users
If this field is populated and Hudson is configured to poll subversion as a build trigger, Hudson will not trigger builds for commits from the specified users. This can be used to avoid builds to be triggered by commits done by Hudson or other systems that commit changes that should not trigger a build.

Excluded Commit Messages
Similar to the option Exclude users this field contains a regular expression and will cause Hudson not to trigger a build for commits with a matching commit message.

Exclusion revprop name
A Subversion revision can be associated with a property. If Hudson encounters a revision with the specified property, it will not trigger a build from an SCM commit, similar to the behaviour for the options Excluded Users or Excluded Commit Messages

9.1.3 Minimal Basic SVN Configuration

In order to build a project controlled in subversion you only need to configure the repository URL in the project configuration using an URL available for anonymous read access to the repository.

9.1.4 Subversion related environment variables

The subversion plugin exports the following environment variables for your usage in build scripts and others:

SVN_REVISION
The repository revision.

SVN_URL
The URL used to access the repository.

If multiple modules are defined these environment variables get and index appended in to their names and all revisions and URLs of the modules will be exported as `SVN_REVISION_1`, `SVN_REVISION_2`, `SVN_REVISION_n` and `SVN_URL_1`, `SVN_URL_2`, `SVN_URL_n`.

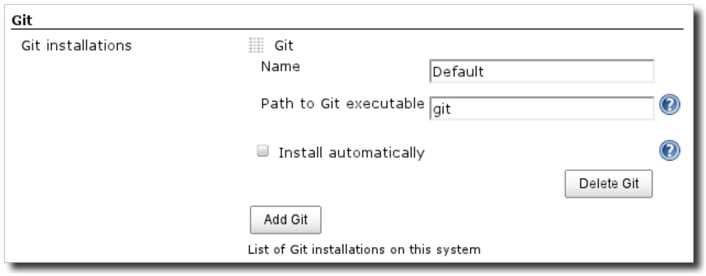
9.2 Configuring Git

Git is the most successful, modern distributed version control system and has gained wide acceptance in the open source community and beyond. and repository hosting services available from multiple suppliers for commercial and open source usage.

The Hudson Git Plugin and therefore support for Git is available in default Hudson installs from version 2.1 onwards. If it is not installed in your Hudson instance, simply find the plugin in the Source Code Management section of the available plugins and install it like any other plugins as documented in Chapter 5.

9.2.1 Global Git Configuration

The global configuration for using Git is set up in the Git section of the Hudson Server configuration screen as displayed in Figure 9.7.



Git

Git installations

Git

Name: Default

Path to Git executable: git

☐ Install automatically

List of Git installations on this system

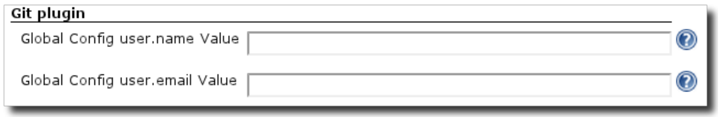
Figure 9.7: Configuring Git Installations

Name
A name for your Git installation can be specified to distinguish between multiple installs.

Path to git executable
If the git command is available on the operating system PATH or the PATH for the user running Hudson, you can simply specify `git`. Otherwise an absolute path can be used as well.

In addition to using an already installed git, the Git plugin facilitates the tool installer from Hudson that allows Run Command and `Extract from *.zip/*.tar.gz` based installs similar to the JDK installs documented in details in Section 3.3. The Ant and the Maven plugins installers described in Section 3.4 and Section 3.5 use the same installers and you can find more hints of its usage there.

A further global configuration for git can be done in the Git plugin section displayed in Figure 9.8. Specifying Global Config user.name Value and Global Config user.email Value values will cause the plugin to issue git config commands setting these options for each project that is configured to use git. The specific project configuration allows you to override these setting for each project individually.



Git plugin

Global Config user.name Value

Global Config user.email Value

Figure 9.8: Configuring Git user name and email values

9.2.2 Project-specific Git Configuration

Once you have configured Git in the global Hudson configuration you can configure project-specific Git settings. Selecting Git under the Source Code Management section of your project configuration will display the configuration options shown in Figure 9.9.

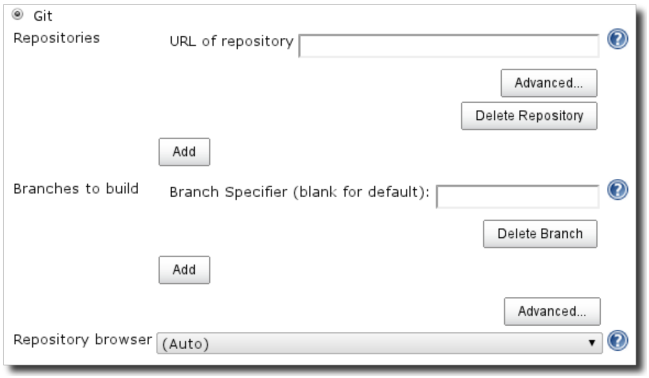


Figure 9.9: Basic Git source control information

The section shown in Figure 9.9 contains the following options for the basic configuration of git in your project configuration:

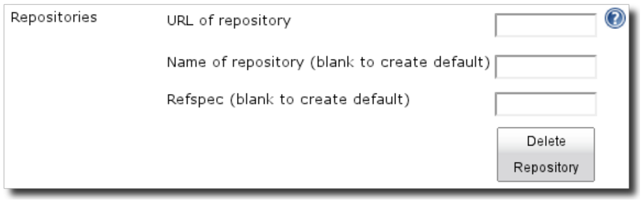
Repositories
Specifying one or more repositories to access for your project can be considered the main configuration of the git details for your project. The minimum configuration is to provide a valid value for URL of repository. The typical protocols `http://`, `ssh://` and `git://` are all supported. Advanced configuration as displayed in Figure 9.10 allows you to provide a name of the repository as well as a refspec. Providing multiple repositories only makes sense when they are clones or instances off the same repository, since they will be used for merging together the content prior to the build.

Branches to build
In this section you can specify one or more branches that should be built. The default of an empty branch specifier causes the git plugin to track all branches and build the latest changed branch.

A common configuration would be to specify the main branch in the repository e.g. `master`. This would ensure that the build is only triggered for changes committed to master. If you want to have other branches built as well it is advisable to create separate Hudson projects for the different branches.

The advanced usage of this feature would be to specify multiple branches. These branches would be used for a merge prior to a build and could be configured to push the merge result back to the remote repository after a successful build.

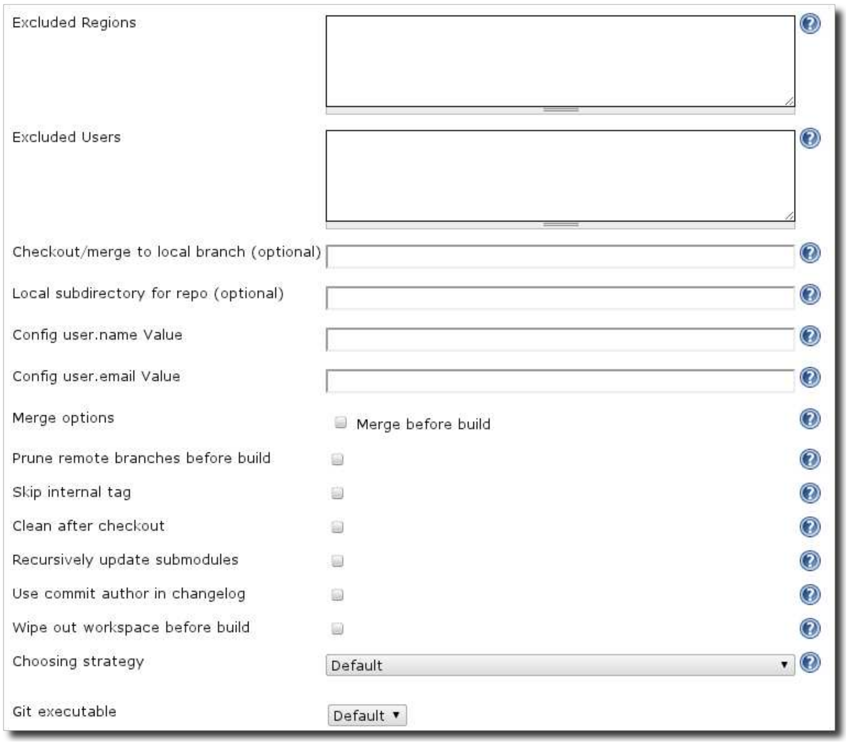
Repository browser
The default Auto option will cause the plugin to attempt to detect a web-based user interface to access the git repository. Selecting one of the supported repository browsers gitweb, redmineweb and githubweb lets you provide a base URL to the repository browser. With the browser URL specified the changes view of each build will have added links to the repository.



Repositories	URL of repository	
	Name of repository (blank to create default)	
	Refspec (blank to create default)	
		Delete Repository

Figure 9.10: Configuring repository specific Git configuration

In many use cases you will be able to configure everything necessary with regards to git using the basic configuration options documented above. For more complex cases the plugin provides various advanced parameters as displayed in Figure 9.11, which become accessible by pressing the Advanced button and documented below.ex



Excluded Regions	
Excluded Users	
Checkout/merge to local branch (optional)	
Local subdirectory for repo (optional)	
Config user.name Value	
Config user.email Value	
Merge options	☐ Merge before build
Prune remote branches before build	☐
Skip internal tag	☐
Clean after checkout	☐
Recursively update submodules	☐
Use commit author in changelog	☐
Wipe out workspace before build	☐
Choosing strategy	Default
Git executable	Default

Figure 9.11: Configuring advanced Git configuration

Excluded Regions
: This configuration allows you to specifically exclude files and directories to determine if a build should be triggered. If set, Hudson will not trigger a build if only files and directories matching the patterns have been

changed. This option is useful if you are only interested in a subset of files and directories contained in a Git repository to trigger a build on Hudson. An example would be if documentation files contained in the repository should not trigger a new build. The configuration supports usage of regular expression patterns to specify the desired files as well as multiple lines to configure larger sets of files and directories to match.

Excluded Users
If this field is populated and Hudson is configured to poll git as a build trigger, Hudson will not trigger builds for commits from the specified users. This can be used to avoid builds to be triggered by commits done by Hudson or other systems that commit changes that should not trigger a build.

Checkout/merge to local branch (optional)
Supplying a value here causes git to create a local branch to checkout to. All the branches specified in the configuration above would be merged into that local branch.

Local subdirectory for repo (optional)
You can specify the name of the subdirectory to checkout a git project to. If you omit this subdirectory, the git repository will be checked out into the workspace directory.

Config user.name Value
This option allows you to cause git to set the `user.name` property prior to checkout and build. It overrides the global git configuration of the same property

Config user.email Value
This option allows you to cause git to set the `user.email` property prior to checkout and build. It overrides the global git configuration of the same property

Merge options
When you activate the option Merge before build the configuration parameters displayed in Figure 9.12 become accessible. This powerful option can be used to specify a repository to merge from in Name of repository . The content of the repository will be merged to the branch specified in Branch to merge to and if the operation succeeds the build will proceed. The merge can then be pushed back to the remote repository by configuring a Git publisher post build action as documented in Section 9.2.3

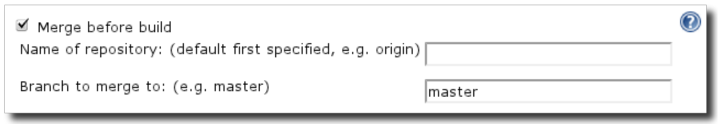

☑ Merge before build
Name of repository: (default first specified, e.g. origin)
Branch to merge to: (e.g. master) master

Figure 9.12: Configuring Git merge options

Prune remote branches before build
Selecting this option will cause the build to run the command git remote prune before each build . This is especially useful if remote branches are created and removed regularly allowing the local checkout in Hudson to stay in sync and only build the branches that also exist remotely.

Skip internal tag
Selecting this option will cause Hudson to omit the tagging of the local repository after each checkout, which performed by default.

Clean after checkout
This option causes git to remove all untracked files after each checkout and prior to the build.

Recursively update submodules
With this option selected and a new enough Git version installed submodules will be updated as part of the update prior to building.

Use commit author in changelog
Selecting this option changes the display of the changelog to show the commits `author`, rather than the default `committer`.

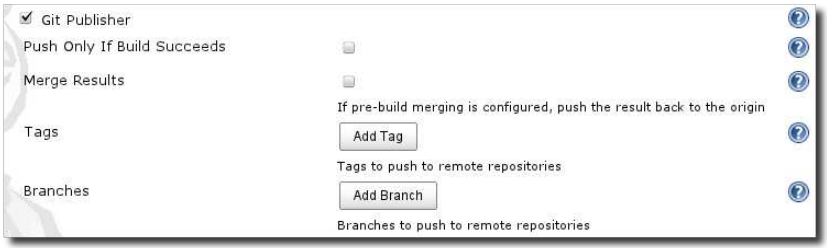
Wipe out workspace before build
This option will cause a complete wipe of the workspace prior to each build. Use caution when activating this option since it can have a significant impact on data transfer and time for the checkout and therefore build. For Maven projects it potentially wipes the local repository as well causing further increases in build time. The other options for keeping the workspace cleaned documented above are more advisable to be used in most cases.

Choosing strategy
This drop down will have only a Default option available with a default Hudson install. It determines which revision of the specified repositories and branches to build. For one branch and one repository HEAD will be built. For multiple branches and repositories a more refined strategy is used selecting revisions that have not yet been built and are on the specified branches. Other plugins can implement a different choosing strategy e.g. the Gerrit Plugin enables a Gerrit change set based strategy effectively allowing verified, pre-tested commits.

Git executable
This drop-down allows you to select a specific git executable used for all operations on this project's build. Configuration of the executables is documented in Section 9.2.1.

9.2.3 Configuring the post-build action Git Publisher

The git plugin adds the post build action Git Publisher as displayed in Figure 9.13 to the project-specific configuration. It can be used to push merges done prior to the build back out to a remote repository after a build.


☑ Git Publisher
Push Only If Build Succeeds ☐
Merge Results ☐
If pre-build merging is configured, push the result back to the origin
Tags

Tags to push to remote repositories
Branches

Branches to push to remote repositories

Figure 9.13: Configuring Git Publisher options

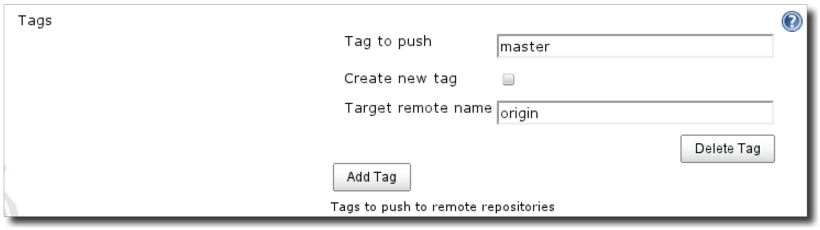
Push Only If Build Succeeds
Selecting this option will cause Hudson to skip publishing any changes via push if the build failed.

Merge Results
With this option activated Hudson will push any merge results done prior to the build back to the remote origin.

Tags
The Add Tag button allows you to configure one or more tags to push to and potentially create.

Branches
The Add Branch button allows you to configure one or more branch and remote combinations to push to.

When configuring the Git Publisher you can specify Tags in fields displayed in Figure 9.14.


Tags
Tag to push: master
Create new tag: ☐
Target remote name: origin

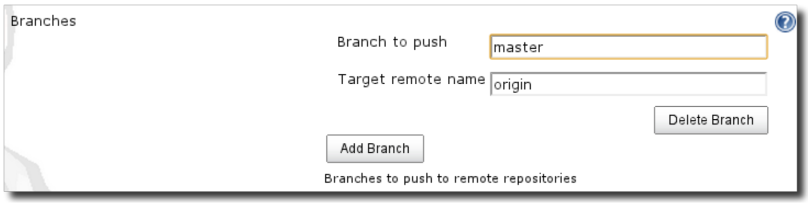
Tags to push to remote repositories

Figure 9.14: Configuring Tags for Git Publisher

Tag to push
This input allows you to provide the name for the tag to use. It supports the expansion of environment variables as part of the tag e.g. Hudson-Build-123. Read Section 3.2 for more information about defining properties and available predefined ones.

Create new tag
This check box determines if the tag to push to as provided above should be created as a new tag or be used as an existing tag.

Target remote name
The name of the remote to push the tag to. The name needs to be configured as a repository in the SCM section for this project. Configuring branches to push to is done in the user interface displayed in Figure 9.15.


Branches
Branch to push: master
Target remote name: origin

Branches to push to remote repositories

Figure 9.15: Configuring Branches for Git Publisher

Branch to push
This parameter specifies the branch the changes get push to after the build completes.

Target remote name
The Target remote name specifies the remote repository to which a push will be done and needs to be one of the names configured as a repository in the SCM section

9.2.4 Minimal Basic Git Configuration

In order to build a project controlled in git you need to have a git installation configured in the global configuration.

In terms of project specific configuration you only need to specify the public URL of the repository. We recommend to specify the branch to build as well since the default setup without a branch specified will examine all branches in the repository and build the latest changed branch. In a normal project this might adversely affect the stability of the build and potentially cause a confusing history for the project. We suggest to set up separate projects for each branch you want to track and build on Hudson.

9.2.5 Multiple branches and automated merging

The recommended basic usage with the git plugin is to configure the project branch for the build e.g. `master`, which will be automatically configured if the field is left blank. Alternative you can specify a different branch name or use `**`. This option causes all branches to be monitored for changes and the branch with the last changes will be built.

This will cause your project build history to be comprised of builds from all the different branches individually in the order of changes received and potentially even omit builds if changes hit multiple branches between builds.

However together with configuring Merge Options it allows for a automated merge from whichever branch to have the latest changes to the target branch e.g. `master` and proceed with the build after the merge.

Now you can activate the Git Publisher post-build action Push Only If Build Succeeds and Merge Results to have the remote repository updated with the successful merge results.

This approach can be configured with specific branches rather than the default empty specification of branches to have better control of the source branches to merge from. It can also be combined with multiple repositories to pull changes in from.

9.3 Configuring Mercurial

Mercurial, also known as hg, is a successful, modern distributed version control system and has gained wide acceptance in the open source community and beyond used for large projects like OpenJDK and Netbeans and repository hosting services available from multiple suppliers for commercial and open source usage.

The Hudson Mercurial Plugin and therefore support for Mercurial needs to be installed on your Hudson instance. Simply find the plugin in the Source Code Management section of the available plugins and install it like any other plugins as documented in Chapter 5.

9.3.1 Global Mercurial Configuration

To configure Mercurial, find the Mercurial section in the Hudson Server configuration screen as displayed in Figure 9.16, press the Add and configure the parameters for your Mercurial install. If Mercurial is already installed on your Hudson server and the hg is available on the path, you do not need to configure a Mercurial installation. The plugin will pick up the installed version.

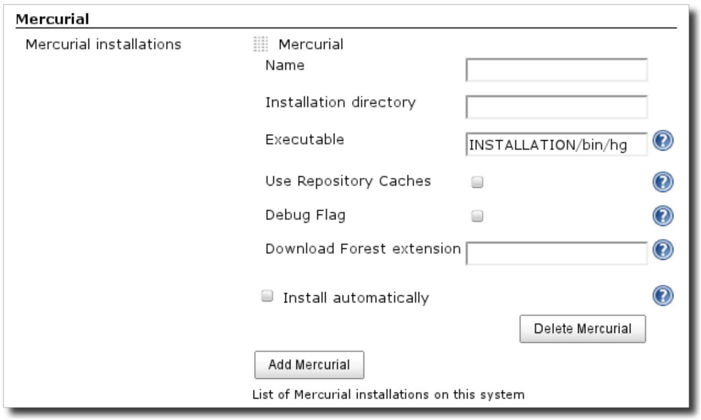

Mercurial
Mercurial installations
Mercurial
Name
Installation directory
Executable INSTALLATION/bin/hg
Use Repository Caches
Debug Flag
Download Forest extension
Install automatically
List of Mercurial installations on this system

Figure 9.16: Configuring Mercurial Installations

Name
: The Name will be displayed in the drop down to select your Mercurial install in the project configuration. Use a name that includes the version to be able to identify the Mercurial install you desire to use in the project configuration, especially when using multiple installs.

Installation directory
: Specify the absolute path to the Mercurial installation .

Executable
: This is the path to the actual `hg` executable. The field is pre-filled with `INSTALLATION/bin/hg`, which is the correct value if you use a manual install of hg. If you are using binary package as provided by your operating system package management system the correct value is likely just `hg`, since the command would be on the `PATH`. Another common option is the absolute path of the executable e.g. `/usr/bin/hg`.

Use Repository Caches
: Enabling this option triggers the Mercurial plugin to establish a repository cache on the Hudson master that will be used by the slave nodes as well. This considerably improves performance and reduces load on the Mercurial server.

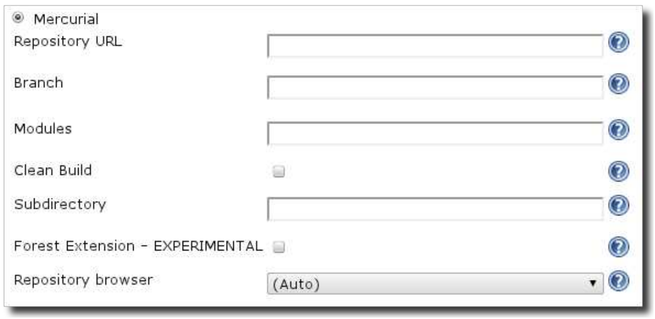
Debug Flag
: As the name suggest activating this option, triggers debug output of any Mercurial command execution. This is especially useful for trouble shooting your configuration.

Download Forest extension
: Using one of the suggested values in the on-line help for this option you can get the forest extension to work with multiple repositories installed automatically. This is especially useful if you can not easily install the extension with the same mechanism you installed Mercurial itself e.g. if no native package for the extension is available.

In addition, the Mercurial plugin facilitates the tool installer from Hudson that allows `Run Command` and `Extract from *.zip/*.tar.gz` based installs similar to the JDK installs documented in details in Section 3.3. The Ant and the Maven plugins installers described in Section 3.4 and Section 3.5 use the same installers and you can find more hints of its usage there.

9.3.2 Project-specific Mercurial Configuration

Selecting Mercurial under the Source Code Management section will display the configuration options shown in Figure 9.17.



● Mercurial	
Repository URL	
Branch	
Modules	
Clean Build	☐
Subdirectory	
Forest Extension - EXPERIMENTAL	☐
Repository browser	(Auto)

Figure 9.17: Configuring Mercurial Source Control Information

The section show in Figure 9.17 contains the following options:

Mercurial Version
This drop down lets you choose from the configured Mercurial installations. In most cases the Default will be fine.

Repository URL
Specify the URL of the project to build.

Branch
By default a branch named `default` will be checked out for the project build. This input allows you to specify and therefore build any other branch.

Advanced - Modules
The advanced setting Modules allows you to specify a folder and its contents, known as module, within the repository to be the exclusive source for changes triggering a build.

Advanced - Clean Build
Activating the Clean Build option causes Mercurial to remove any untracked files prior to the build.

Advanced - Subdirectory
By default the repository is checked out into the workspace. Supplying a value in the Subdirectory option will cause the repository to be checked out into a subdirectory in the workspace.

Advanced - Forest Extension
This experimental features triggers the activation of the Mercurial forest extension, which will treat the workspace as the root of a forest.

Repository Browser
The default Auto option will cause the plugin to attempt to detect a web-based user interface to access the Mercurial repository. Selecting one of the supported repository browsers hgweb, bitbucket, googlecode, kilnhg and fisheye lets you provide a URL to the repository browser. With the URL specified the changes view of each build will have added links to the repository.

9.3.3 Minimal Basic Mercurial Configuration

In order to build a project controlled in Mercurial you need to install the Mercurial plugin and have a hg installation configured in the global configuration.

In terms of project specific configuration you only need to specify the public URL of the repository.

9.4 Configuring CVS

If your Hudson Jobs rely on CVS there is no plugin configuration necessary, as the Hudson CVS Plugin and support for the CVS system is included as a part of the default Hudson distribution.

9.4.1 Global CVS Configuration

To configure CVS, find the CVS section in the Hudson Server configuration screen as displayed in Figure 9.18. From the Hudson start page, click on Manage Hudson, then click on Configure System, and scroll down to the CVS section of this configuration form. If CVS is already installed on your Hudson server and the cvs is available on the path, you do not need to configure a CVS installation. The plugin will pick up the installed version.

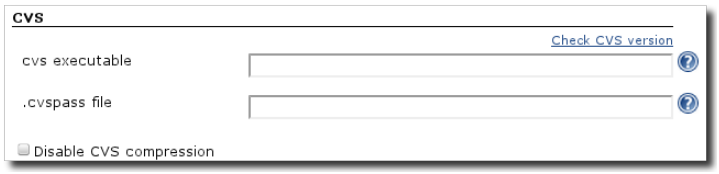


Figure 9.18: Global CVS Configuration

The CVS configuration section allows you to configure the following properties:

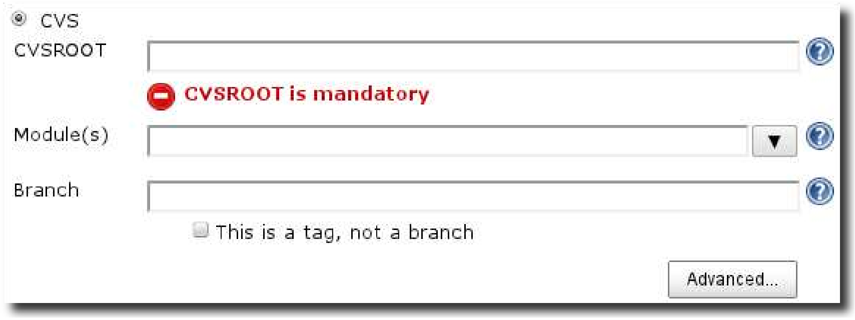
cvs executable
If the cvs executable is not available on the PATH, you can configure an absolute path to it in this input field.

cvspass file
Specify the path to the .cvspass file that contains the user credentials.

Disable CVS compression
Activate this option to disable CVS compression.

9.4.2 Project-specific CVS Configuration

Selecting CVS under the Source Code Management section will display the configuration options shown in Figure 9.19.



● CVS

CVSROOT

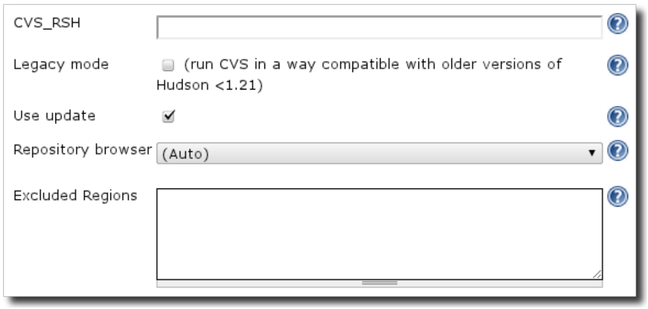
CVSROOT is mandatory

Module(s)

Branch

This is a tag, not a branch

Figure 9.19: Configuring CVS Source Control Information



CVS_RSH

Legacy mode (run CVS in a way compatible with older versions of Hudson <1.21)

Use update ✓

Repository browser (Auto)

Excluded Regions

Figure 9.20: Configuring CVS Source Control Information

The section show in Figure 9.19 and Figure 9.20 contain the following options:

CVSROOT
: This is the CVSROOT parameter for your source. You would enter in the same information here that you would use if you were checking out source code use the CVS client. The format for the URL is

```
<protocol>:<user>:<password>@<servername>:<serverpath>
```

Each component in the CVSROOT parameter

protocol
: The protocol defines the way the CVS client communicates to the repository. Supported protocols are `:local:` for local or net file system level connection, `:pserver:` as the standard unsecured protocol, `:ext:`, `:ssh:` and `:extssh:` for secure shell based connections and `:sspi:` for Windows based access.

Note
Due to concurrency and additional load on the repository, we recommend avoiding the local protocol for CI builds. Using the local protocol in Hudson often results in problems if too many clients are trying to access a CVS repository at the same time.

Warning
Please also keep in mind that the pserver protocol is insecure. If you storing sensitive data or code in a CVS repository, avoid the pserver protocol and use ssh as an alternative.

user
Supply the user name for the client-server protocol used.

password
Specify the password when using the `:pserver:` or `:sspi:` protocol.

servername
The server name of the repository server as reachable via the network. It can be a fully qualified server and domain name, a server name only or an IP number. The character `@` is required at the beginning of the server name.

serverpath
The path to the repository on the server pre-pended by `:`. The path itself can be either unix style like `/opt/data/cvsr` or Windows style like `C:\cvs\repository`.

An example for a valid URL with a module is shown below.

```
:pserver:anonymous@tortoisecvs.cvs.sourceforge.net:/cvsroot/tortoisecvs
```

Module(s) and Branch
Here you can provide specific modules and specific branches to be checked out by Hudson. Multiple module can be specified using a space separated list and with a parameterized build job parameters can be used to specify branch or module.

CVS_RSH
If you are using CVS over SSH, you can specify options and parameters in this variable.

Use update
Hudson can be configured to do a full checkout on each build or to use cvs update. Selecting this option will tell Hudson to use update instead of performing a clean checkout. For most reasonably sized projects you should activate this option.

Repository browser

The default Auto option will cause the plugin to attempt to detect a web-based user interface to access the CVS repository. Selecting one of the supported repository browsers ViewCVS and FishEye lets you provide a URL to the repository browser.

Excluded Regions

If the module you are checking out contains some files that you need to exclude from triggering a build when changed, you can exclude regions by populating this field with the respective patterns.

Chapter 10

Tools Integration

The default way of interacting with Hudson is the web-based user interface. In addition Hudson ships with web service interfaces, that enable integration with others tools such as integrated development environments. The following section will detail some of these integrations, with links to further tools available on the Hudson wiki.

10.1 Eclipse Integration

As a top level Eclipse project it is only fitting that Hudson has great integration with the Eclipse development environment. Historically there are various independently developed integrations that are currently available. Most notably Tasktop's Eclipse Mylyn includes Hudson integration as of release 3.6 and Sonatype provides a Hudson integration as well. Other integrations have mostly stalled and we recommend to use either of these two. Both Sonatype and Tasktop are project members of the Eclipse Hudson project and going forward a merge of the two integrations is planned.

10.1.1 Sonatype Hudson Integration

Installation of the Hudson integration is currently with the update site URL https://repository.sonatype.org/content/-repositories/forge-sites/m2eclipse-hudson/0.13.0/S/0.13.0.20111015-0033/ following the usual install process. Source code for the integration can be found on github.

To open up the Hudson Jobs view in Eclipse, go to Window → Show View → Other... . Selecting this menu item will display the dialogue shown in Figure 10.1.

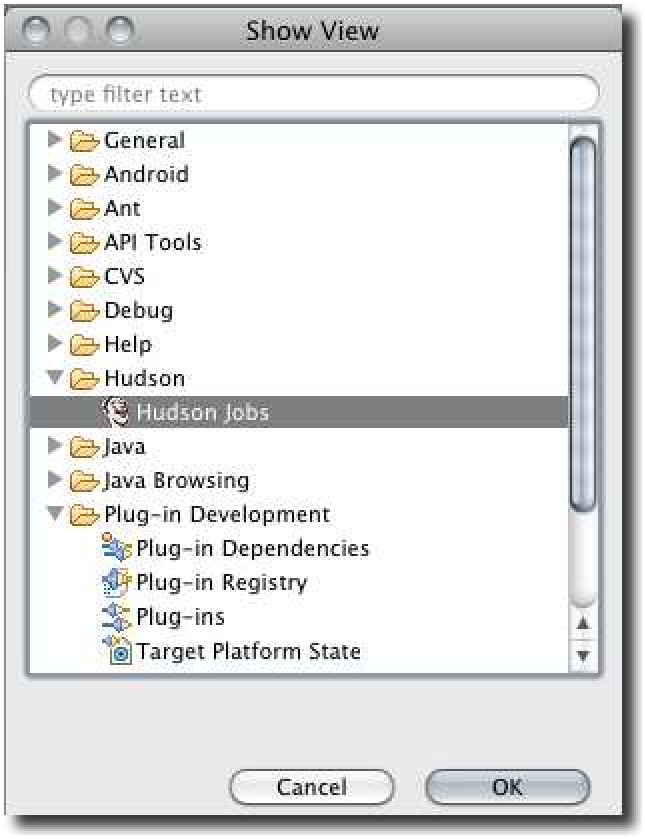
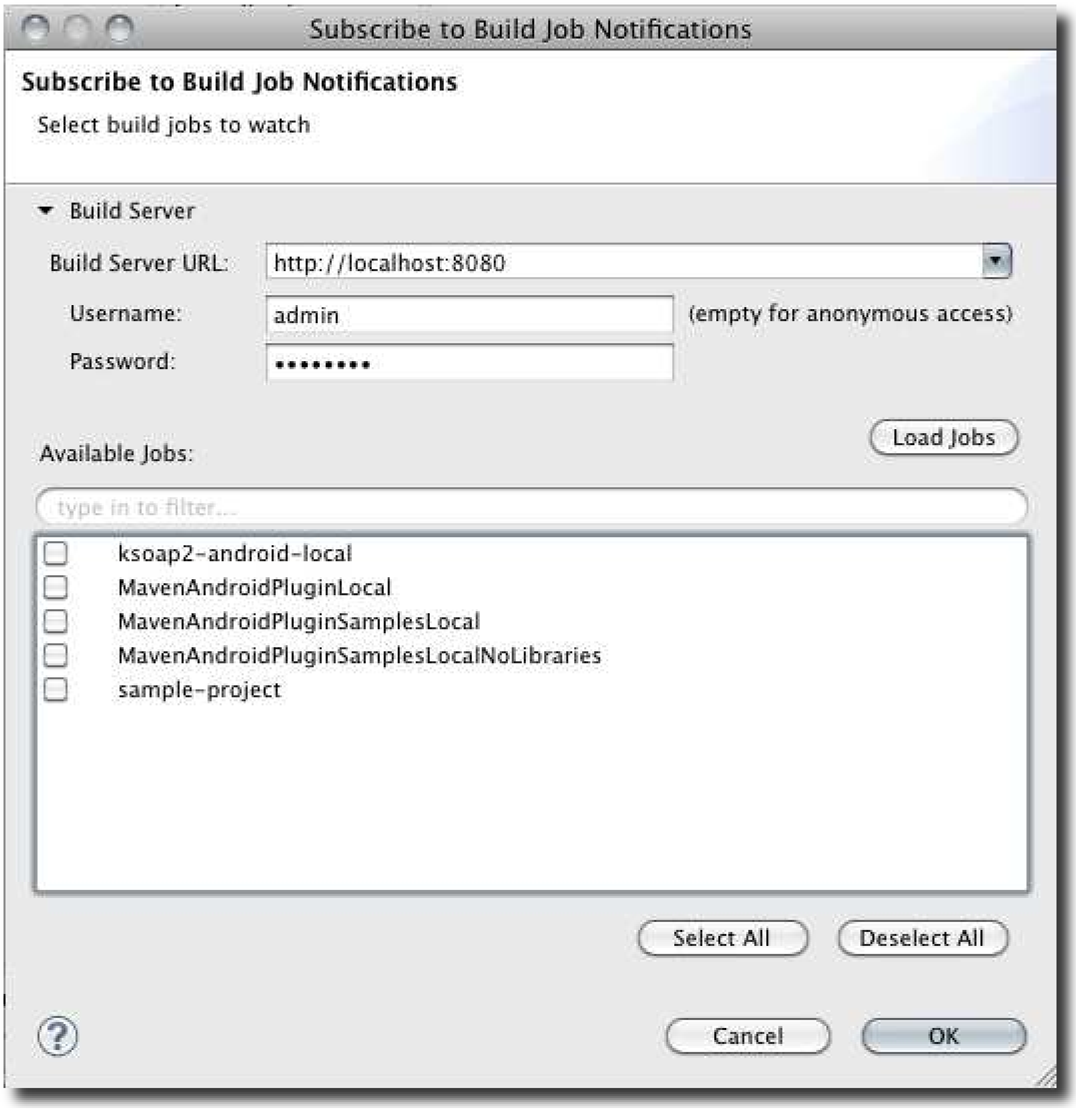


Figure 10.1: Opening the Hudson Jobs View in Eclipse

Once you have selected this Hudson Jobs view, you have to configure the Build Server URL in the Subscribe to Build Notifications dialogue displayed in Figure 10.2 that can be reached via the blue sphere icon on the top left of the Hudson Jobs view in Figure 10.3.



Subscribe to Build Job Notifications

Subscribe to Build Job Notifications

Select build jobs to watch

Build Server

Build Server URL: http://localhost:8080

Username: admin (empty for anonymous access)

Password: ••••••••

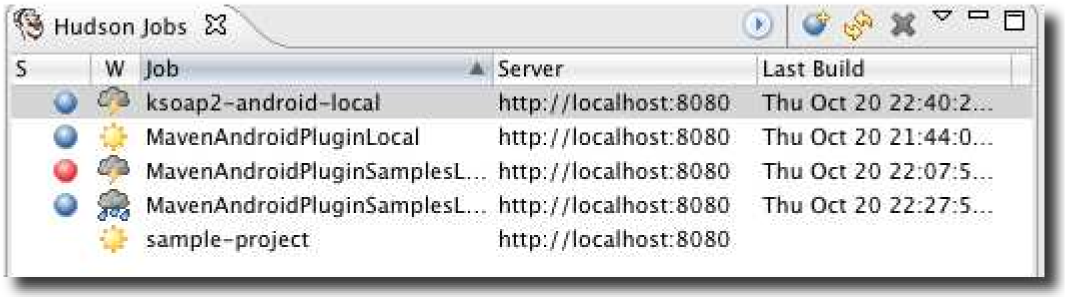
Available Jobs:

- [ ] ksoap2-android-local
- [ ] MavenAndroidPluginLocal
- [ ] MavenAndroidPluginSamplesLocal
- [ ] MavenAndroidPluginSamplesLocalNoLibraries
- [ ] sample-project

Figure 10.2: Subscribe to Build Notifications Dialogue

Optionally provide user name and password and then press Load Jobs to have the list of jobs on the bottom of the dialogue populated.

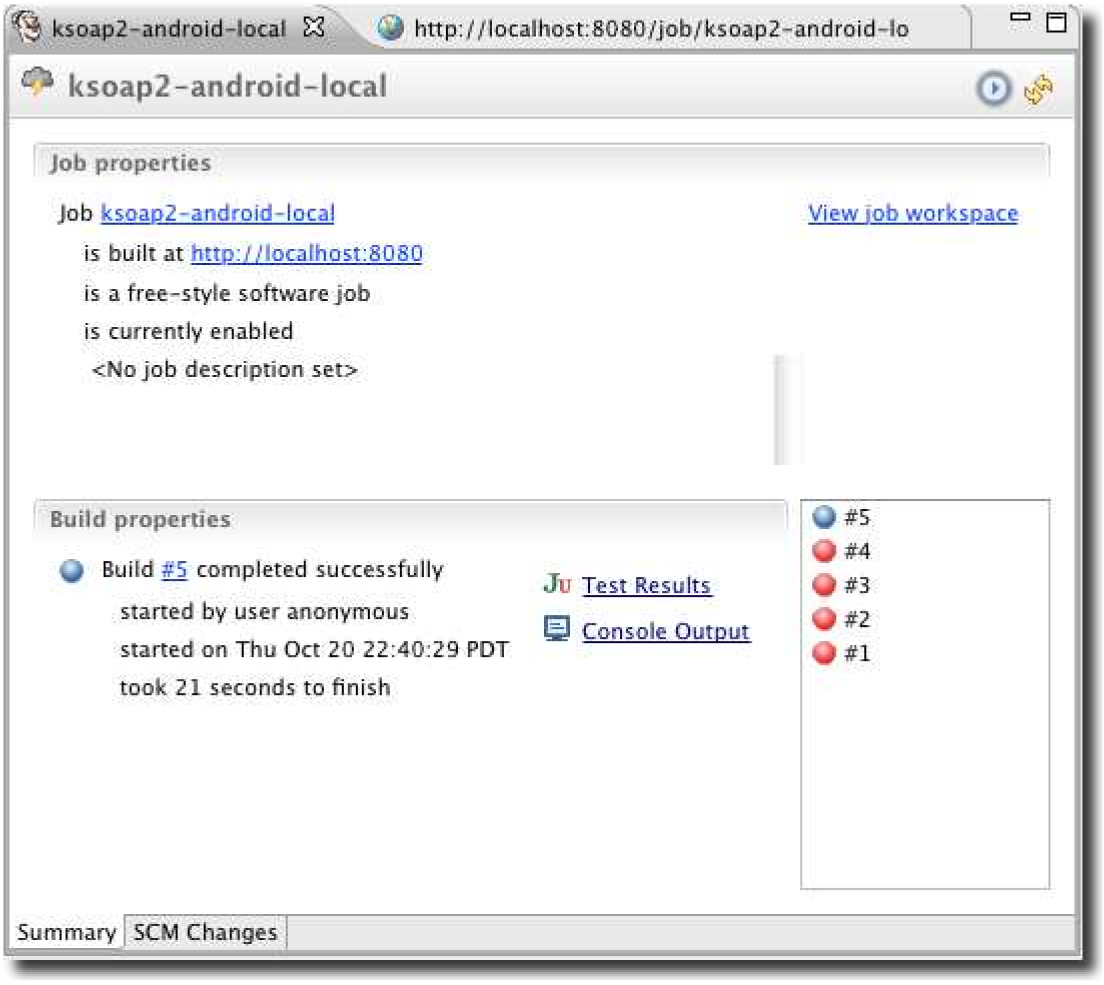
After you select the desired jobs and exit the dialog with OK, you will see the list of jobs in the Hudson Jobs view in Figure 10.3. This list is retrieved directly from Hudson using Hudson REST services. To refresh the list of jobs, click on the refresh icon.



S	W	Job	Server	Last Build
●		ksoap2-android-local	http://localhost:8080	Thu Oct 20 22:40:2...
●		MavenAndroidPluginLocal	http://localhost:8080	Thu Oct 20 21:44:0...
●		MavenAndroidPluginSamplesL...	http://localhost:8080	Thu Oct 20 22:07:5...
●		MavenAndroidPluginSamplesL...	http://localhost:8080	Thu Oct 20 22:27:5...
		sample-project	http://localhost:8080	

Figure 10.3: Hudson Jobs View in Eclipse

Once you've loaded a list of Hudson jobs in Eclipse, you can click on one of these Build Jobs to view a detailed snapshot of the project status and any associated builds like in Figure 10.4. In this dialog you can see that the job detail window contains general information about the Job in Job Properties, information about specific Job Builds in Build Properties and also links to alternative configuration for Build Jobs. If you need to get more information about the Job's workspace in order to trouble shoot a build, you can also click on View job workspace. Clicking on this link will load the project's workspace in a browser window as another Eclipse tab.


ksoap2-android-local

http://localhost:8080/job/ksoap2-android-lo

ksoap2-android-local

Job properties

Job ksoap2-android-local

is built at http://localhost:8080

is a free-style software job

is currently enabled

<No job description set>

View job workspace

Build properties

Build #5 completed successfully

started by user anonymous

started on Thu Oct 20 22:40:29 PDT

took 21 seconds to finish

Test Results

Console Output

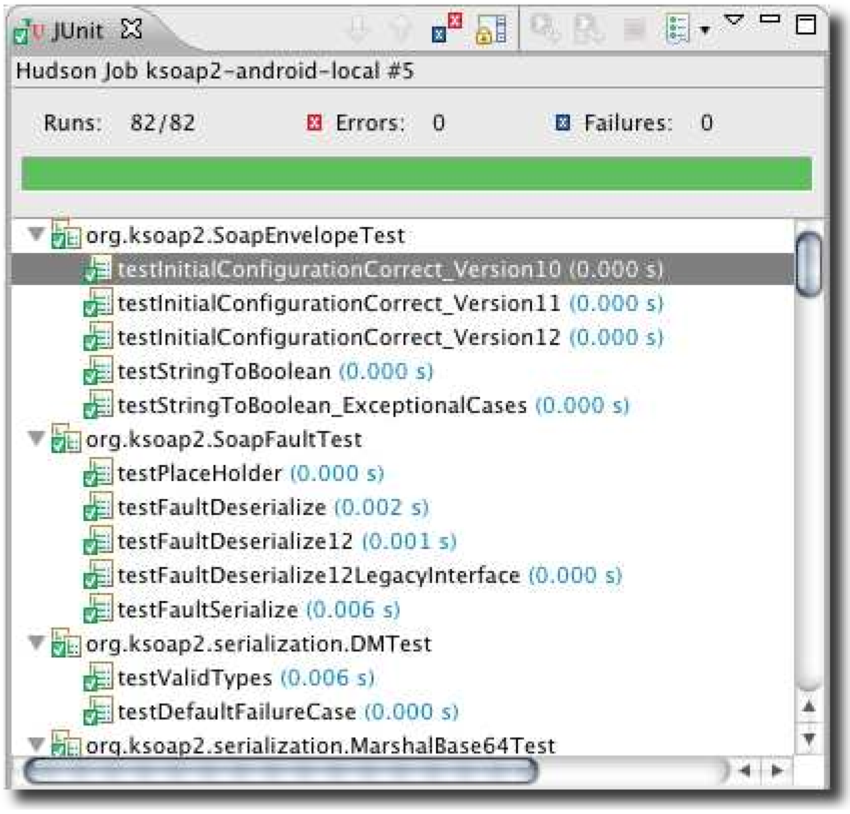
#5
#4
#3
#2
#1

Summary | SCM Changes

Figure 10.4: Hudson Jobs Summary

There are some very useful ways in which you can view the results of a build. They are viewing a build's JUnit test results, viewing the SCM changes associated with a specific build, and viewing a build's console output. These views of a particular job are the main ways in which you can understand and diagnose issues with continuous integration builds.

To view a build job's unit test results, load the job detail page and click on the Test Results link shown in Figure 10.4. Clicking on this link will load the view shown in Figure 10.5. This level of visibility into the continuous integration machine gives you insight into code-level issues happening on a remote build machine without requiring you to exit out of Eclipse and fire up a web browser.



```
org.ksoap2.SoapEnvelopeTest
    testInitialConfigurationCorrect_Version10 (0.000 s)
    testInitialConfigurationCorrect_Version11 (0.000 s)
    testInitialConfigurationCorrect_Version12 (0.000 s)
    testStringToBoolean (0.000 s)
    testStringToBoolean_ExceptionalCases (0.000 s)
org.ksoap2.SoapFaultTest
    testPlaceHolder (0.000 s)
    testFaultDeserialize (0.002 s)
    testFaultDeserialize12 (0.001 s)
    testFaultDeserialize12LegacyInterface (0.000 s)
    testFaultSerialize (0.006 s)
org.ksoap2.serialization.DMTest
    testValidTypes (0.006 s)
    testDefaultFailureCase (0.000 s)
org.ksoap2.serialization.MarshalBase64Test
```

Figure 10.5: Viewing JUnit Test Reports from Eclipse

To view a specific build's SCM changes. Load the project detail view and click on the SCM Changes tab shown in Figure 10.4. This particular view of a project combines the changelog of every single build into an easily navigable interface. From that view you can see what code changes trigger individual builds and you can get a sense for what activity and which committer are responsible for build successes and failures.

Viewing a specific build's console output is often the quickest way to get to the bottom of a build failure. To open a build's console output, select the project from the Hudson jobs view, select an individual project build, and then click on the Console Output link shown in Figure 10.4 and you will see the the raw console output from the Hudson build.

The test results, console output and other features can also be accessed by right clicking on a job in the jobs list view and selecting from the context menu as visible in Figure 10.6.

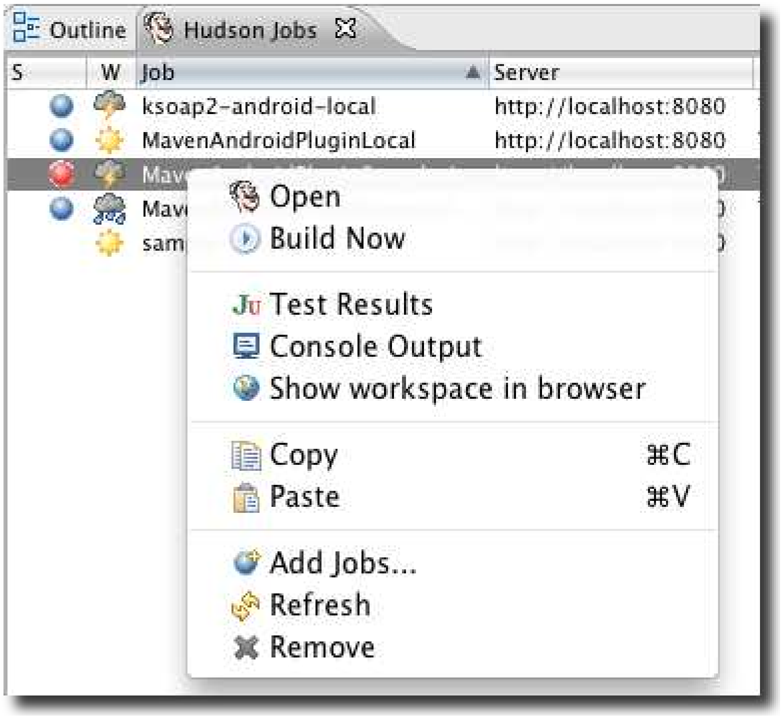
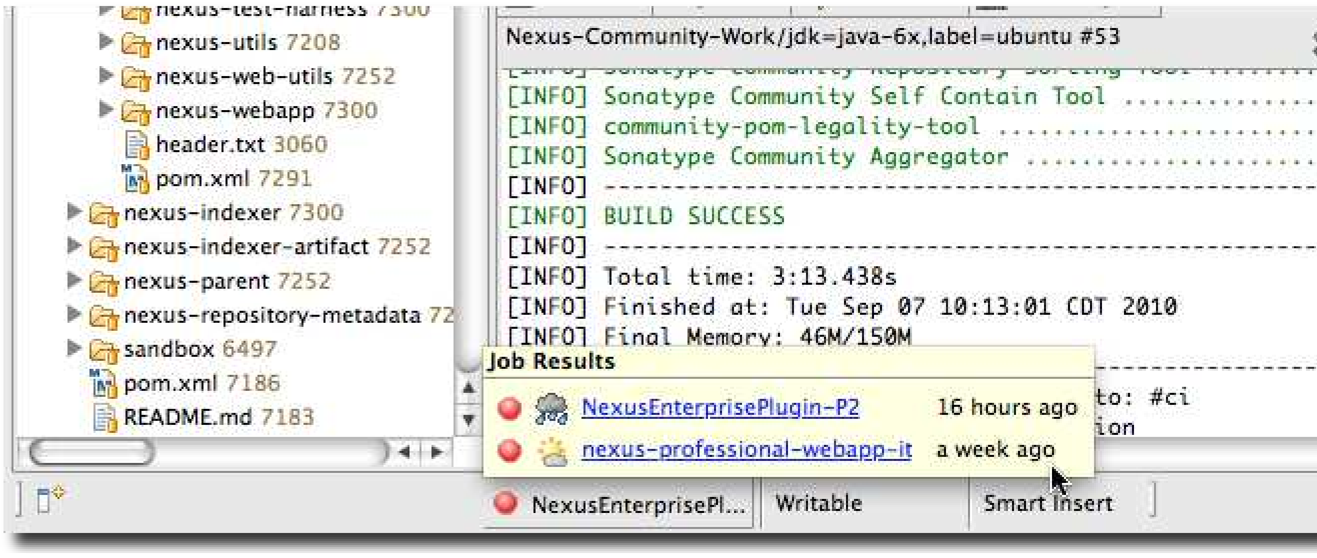


Figure 10.6: Contex Menu for a Specific Hudson Job

Once you've connected Eclipse to Hudson, you can also configure the tool to notify you of build failure events. Figure 10.7 shows an Eclipse installation which has been configured to receive notifications of specific build failures. Eclipse will periodically poll Hudson to check for build failures.



```
Nexus-Community-Work/jdk=java-6x,label=ubuntu #53
[INFO] Sonatype Community Self Contain Tool .............
[INFO] community-pom-legality-tool ......................
[INFO] Sonatype Community Aggregator ....................
[INFO] ------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------
[INFO] Total time: 3:13.438s
[INFO] Finished at: Tue Sep 07 10:13:01 CDT 2010
[INFO] Final Memory: 46M/150M
```

Figure 10.7: Hudson Notifications in Eclipse

10.1.2 Tasktop Mylyn Builds Connector for Hudson

The Mylyn Builds Connector can be found in the standard update sites that are pre-configured with any Eclipse install.

To open up and configure the Builds view in Eclipse, go to Window → Show View → Other…, find the Builds view in the Mylyn section visible in Figure 10.8 and click the Create a build server link or the blue New Build Server Location icon.

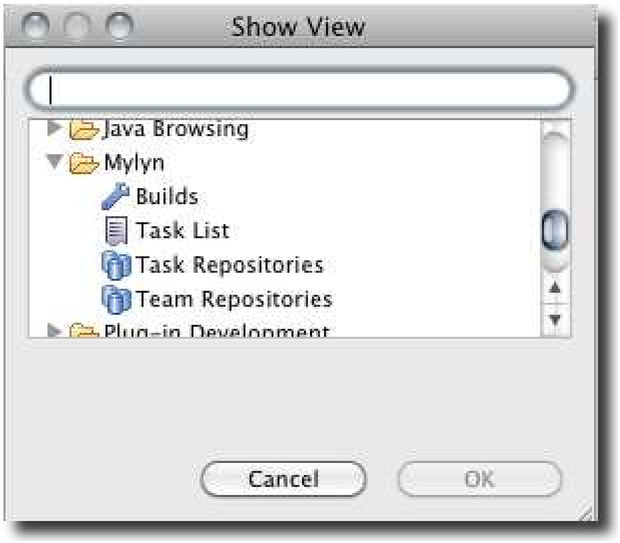
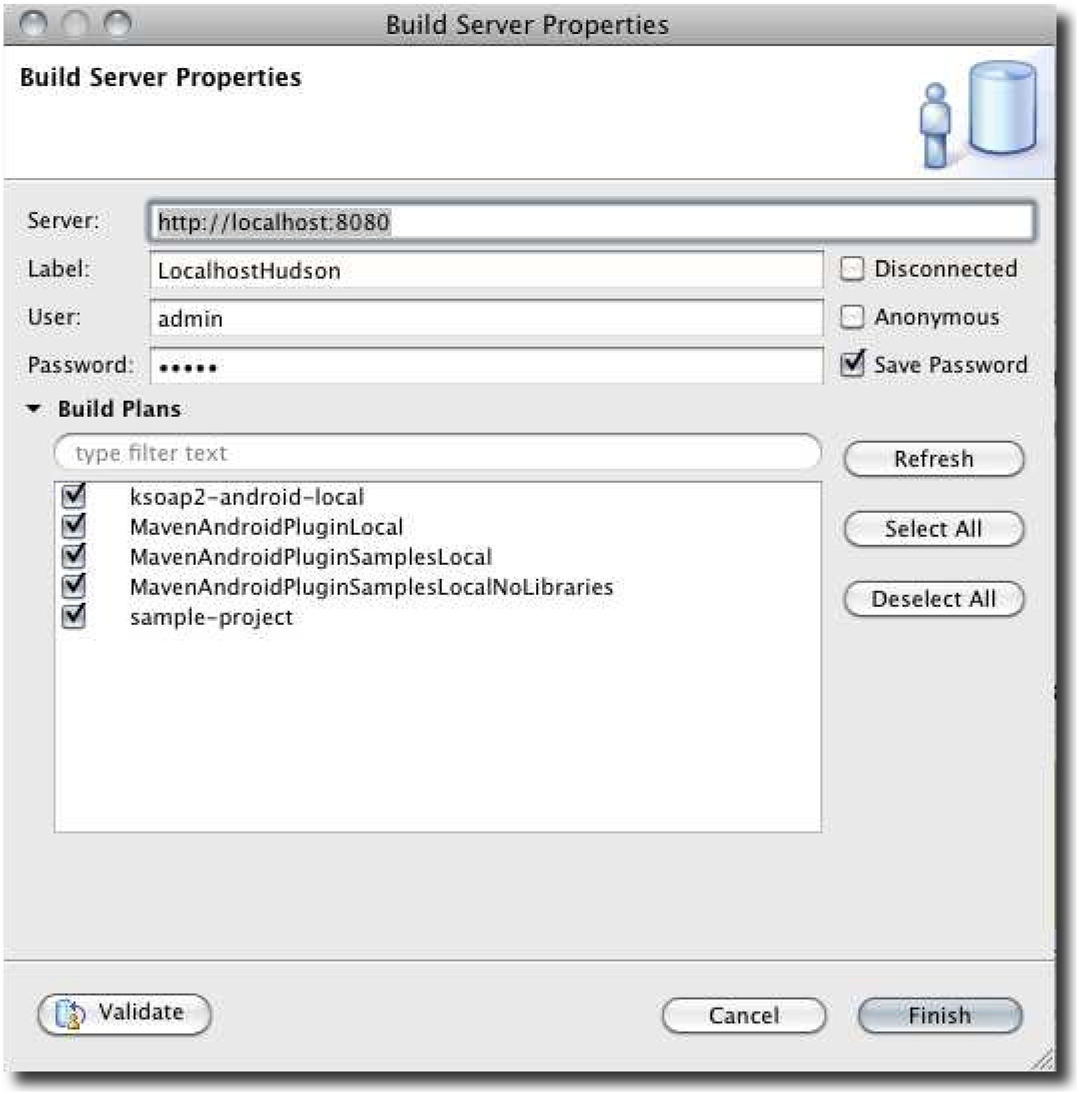


Figure 10.8: Selecting the Builds View in the Show View Dialog

After selecting Hudson in the wizard you will be able to configure the server in the Hudson Server Properties dialog in Figure 10.9.



Build Server Properties

Build Server Properties

Server: http://localhost:8080

Label: LocalhostHudson

User: admin

Password: •••••

[ ] Disconnected

[ ] Anonymous

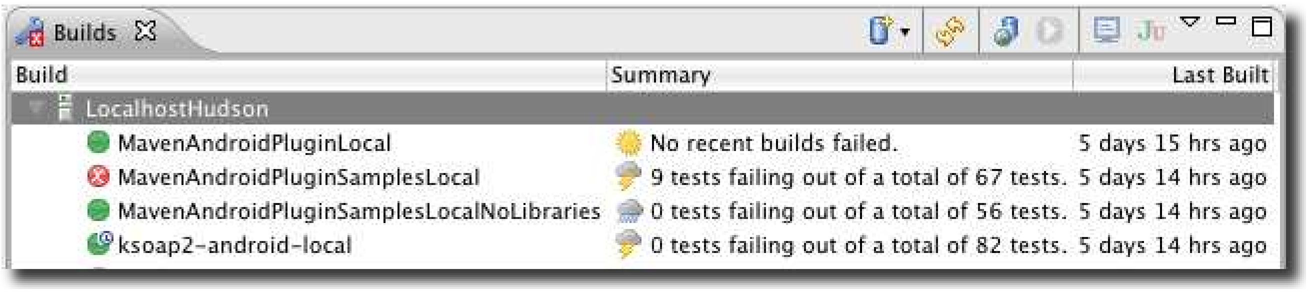
[✓] Save Password

Build Plans

- [✓] ksoap2-android-local
- [✓] MavenAndroidPluginLocal
- [✓] MavenAndroidPluginSamplesLocal
- [✓] MavenAndroidPluginSamplesLocalNoLibraries
- [✓] sample-project

Figure 10.9: Hudson Server Properties Configuration

Provide the Server URL, a label and optionally user and password and press Refresh to see a list of available Build Plans. Selecting plans and pressing Finish will close the dialog and show the build plans in the Builds view as visible in Figure 10.10.



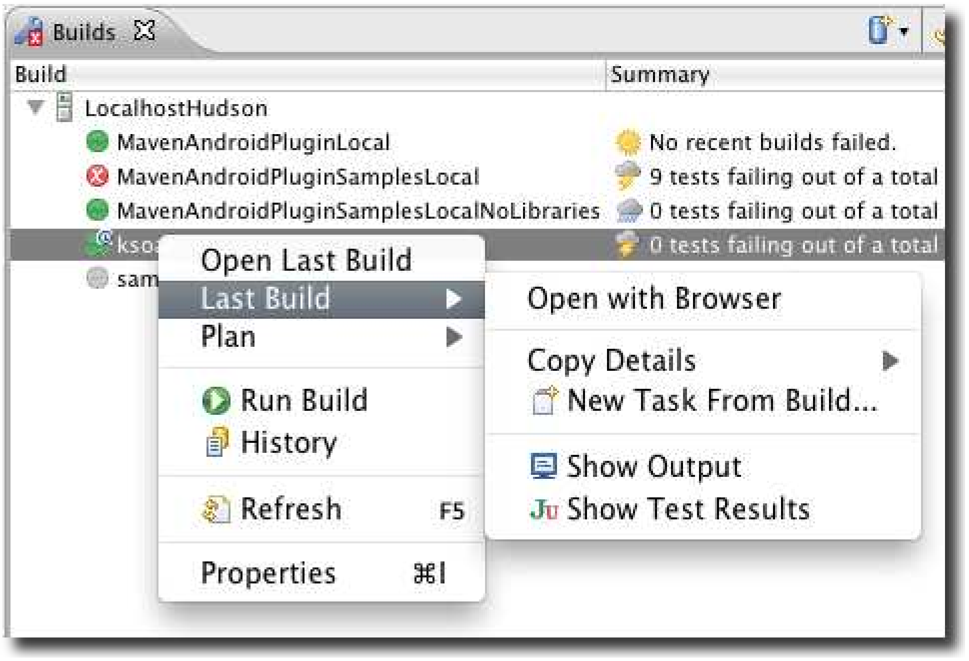
Build	Summary	Last Built
LocalhostHudson		
MavenAndroidPluginLocal	No recent builds failed.	5 days 15 hrs ago
MavenAndroidPluginSamplesLocal	9 tests failing out of a total of 67 tests.	5 days 14 hrs ago
MavenAndroidPluginSamplesLocalNoLibraries	0 tests failing out of a total of 56 tests.	5 days 14 hrs ago
ksoap2-android-local	0 tests failing out of a total of 82 tests.	5 days 14 hrs ago

Figure 10.10: Builds List View

With the same process you can add further Hudson servers to monitor by pressing on the New Build Server Location button.

If you select a specific plan the icons in the top left corner of the builds view give you access to the job page directly on Hudson in a browser window in Eclipse and the console output and the test results of the last build in an Eclipse windows. You can start another build with the Run Build button.

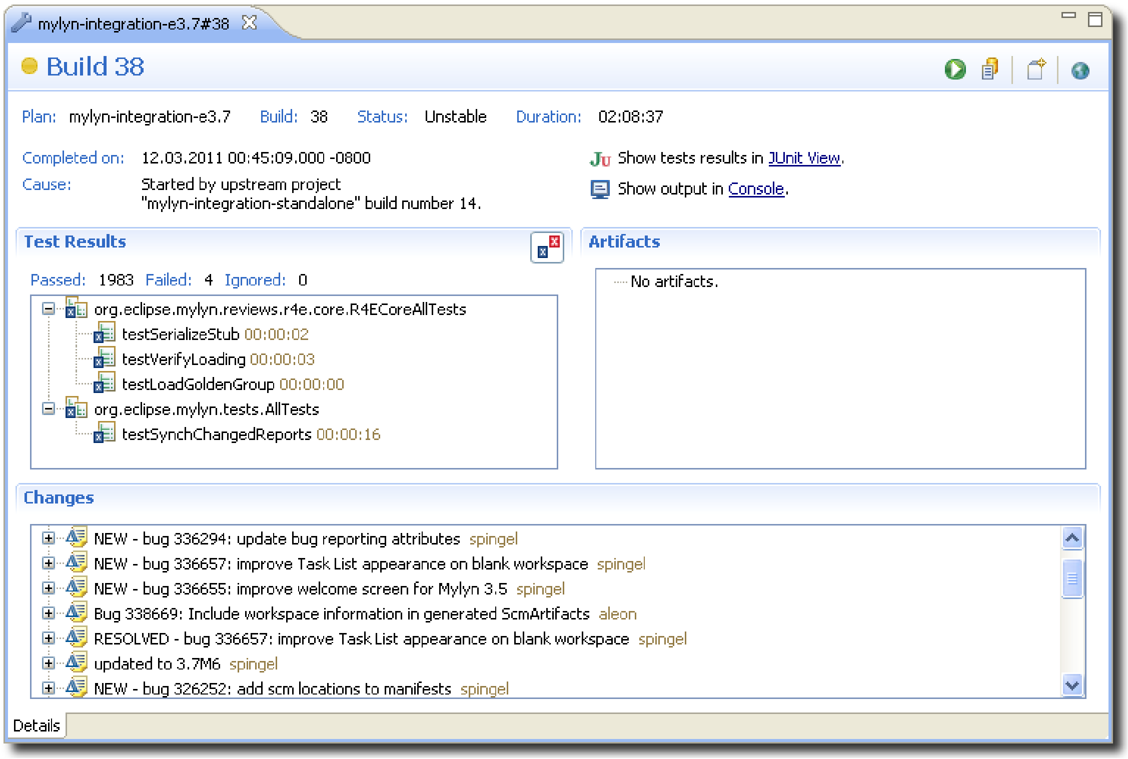
A context menu for each build plan like displayed in Figure 10.11 provides access to the build history and integration into the Mylyn task management.



Build	Summary
LocalhostHudson	
MavenAndroidPluginLocal	No recent builds failed.
MavenAndroidPluginSamplesLocal	9 tests failing out of a total
MavenAndroidPluginSamplesLocalNoLibraries	0 tests failing out of a total
kso	0 tests failing out of a total
sam	

Figure 10.11: Context Menu for an Individual Build Plan

The individual builds overview page visible in Figure 10.12 displays various details about the build and provides access to test results, artifacts, changes and console output with further integration to the task management.



Plan: mylyn-integration-e3.7 Build: 38 Status: Unstable Duration: 02:08:37

Completed on: 12.03.2011 00:45:09.000 -0800
Cause: Started by upstream project "mylyn-integration-standalone" build number 14.

Show tests results in JUnit View.
Show output in Console.

Test Results

Passed: 1983 Failed: 4 Ignored: 0

```
org.eclipse.mylyn.reviews.r4e.core.R4ECoreAllTests
    testSerializeStub 00:00:02
    testVerifyLoading 00:00:03
    testLoadGoldenGroup 00:00:00
org.eclipse.mylyn.tests.AllTests
    testSynchChangedReports 00:00:16
```

Artifacts

No artifacts.

Changes

```
NEW - bug 336294: update bug reporting attributes spingel
NEW - bug 336657: improve Task List appearance on blank workspace spingel
NEW - bug 336655: improve welcome screen for Mylyn 3.5 spingel
Bug 338669: Include workspace information in generated ScmArtifacts aleon
RESOLVED - bug 336657: improve Task List appearance on blank workspace spingel
updated to 3.7M6 spingel
NEW - bug 326252: add scm locations to manifests spingel
```

Figure 10.12: Individual Build Overview Display

10.2 Oracle JDeveloper Team Productivity Center

The Oracle JDeveloper Team Productivity Center optionally includes the Hudson Plugin for integration with Hudson in the IDE. The plugin pushed build and test results from Hudson via the Team Productivity Center server to the JDeveloper IDE.

10.3 Netbeans

The Netbeans IDE can use the Hudson support plugin to provide integration of Hudson in the IDE.

10.4 Jetbrains IntelliJ IDEA

IntelliJ IDEA users can install the Hudson Build Monitor plugin for integration with Hudson in the IDE.

10.5 Hudson Integration for Android

The Hudson Monitor for Android application or Hudson2Go can be used to access build details on your Android based mobile phone or tablet.

10.6 Firefox Add-on Build Monitor

The Firefox Add-on Build Monitor provides access to Hudson as a browser plugin.

Appendix A

Creative Commons License

```
<div xmlns:cc="http://creativecommons.org/ns#"
about="http://creativecommons.org/license/results-one?q_1=2&q_1=1\
&field_commercial=n&field_derivatives=n&field_jurisdiction=us\
&field_format=StillImage&field_worktitle=Repository%3A+\Management\
&field_attribute_to_name=Eclipse%2C+Hudson\
&field_attribute_to_url=http%3A%2F%2Fwww.eclipse.org%2Fhudson\
&field_sourceurl=http%3A%2F%2Fwww.eclipse.org%2Fhudson%2Fhudson-book\
&lang=en_US&language=en_US&n_questions=3">
<a rel="cc:attributionURL" property="cc:attributionName"
href="http://www.eclipse.org/hudson">Eclipse Hudson</a> /
<a rel="license"
href="http://creativecommons.org/licenses/by-nc-nd/3.0/us/">
CC BY-NC-ND 3.0</a>
</div>
```

A.1 Creative Commons BY-NC-ND 3.0 US License

A.2 Creative Commons Notice