Luntbuild - Quick Start Guide

Luntbuild - Quick Start Guide

Table of Contents

Chapter 1. Introduction

This document serves as "Quick Start" guide for Luntbuild, a build automation and management tool. We will try to explain basic use of Luntbuild by building the Luntbuild itself. You can start your newly installed Luntbuild, point your browser to it, and follow the guide along.

So what is really that Luntbuild thing?

Luntbuild is a build automation and management tool based on the popular *Apache Ant* [http://ant.apache.org]. With Luntbuild, daily builds and continuous integration builds can be set easily. Refer to the following articles for benefits of daily builds and continuous integration builds, if you are not familiar with them:

- *Continuous Integration* [http://www.martinfowler.com/articles/continuousIntegration.html]
- *Daily Builds Are Your Friend* [http://www.joelonsoftware.com/articles/fog0000000023.html]

You can explore Luntbuild's functionality by viewing *tutorial* [http://luntbuild.javaforge.com/luntbuild-demo.html] movie. You can also check for Luntbuild FAQ [../faq/index.html] to learn more about Luntbuild.

Basic unit of work in Luntbuild is a *build*. Build execution is triggered either by a schedule or it can be started manually. A build in Luntbuild performs following steps:

1. Checks out source code from the Version Control System(s) (VCS).
2. Labels the current source code based on the current build version.
3. Runs an Ant/Maven/Command/Rake build script in the source tree.
4. Runs an Ant/Maven/Command/Rake post build script in the source tree.
5. Publishes the build log and other build artifacts.

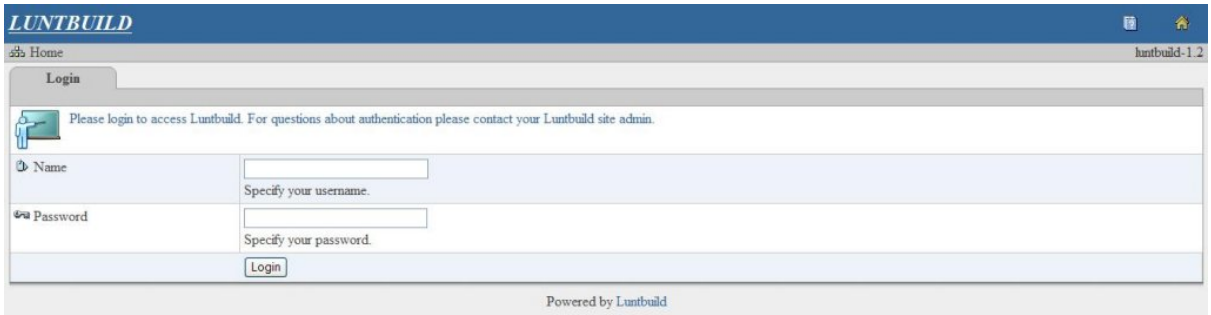
Build configuration, monitoring, and access to the build artifacts are all done using an intuitive web interface. Your development and testing team will have a central area to access the build information.

Please consult Luntbuild User's Guide [http://luntbuild.javaforge.com/manual/guide/manual.html#installation] or Installation Guide [http://luntbuild.javaforge.com/docs/installguide/installguide.html] about installing and configuring Luntbuild.

Now lets login to Luntbuild and create a project that will allow us to build Luntbuild 8-). Next two chapters explain Luntbuild's login and home page.

Chapter 2. Login to Luntbuild

And now is the exciting time to login for the first time into Luntbuild. The Luntbuild login page:



asks you to enter a *Name* and *Password*. Use luntbuild/luntbuild for name/password, or if you have modified the security configuration, use the password you have specified in applicationContext.xml configuration file (please see *Luntbuild Security* for details).

Login page contains two icons in upper right corner of the screen. Those two icons are present on all Luntbuild pages. The icon is a link to this Luntbuild User's Guide. The icon is a link to the Luntbuild web site.

Enter the name and the password and click the Login button (or press Enter) to login to Luntbuild.

You can login to Luntbuild as:

1. *System Administrator* - with default name/password luntbuild/luntbuild
2. *Registered user* - created by system administrator
3. *Anonymous user* - does not need to be registered

Registered users may optionally create projects. They can be granted privileges to view and modify projects, builds and schedules, and execute builds.

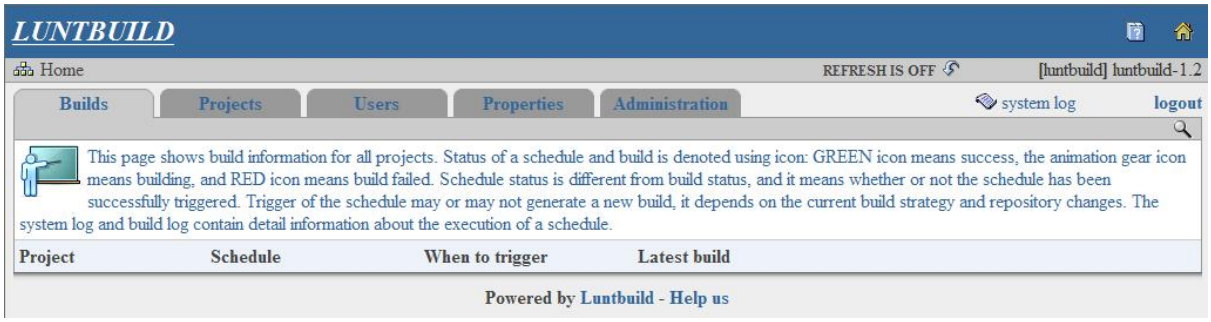
Anonymous users may can only view projects, builds and schedules.

You can login as anonymous user by using one of the following methods:

1. *http://<server>:8080/luntbuild/luntbuild-login.jsp* - click on "Login as Anonymous" link
2. *http://<server>:8080/luntbuild/luntbuild-login-anonymous.jsp* - automatically redirects to anonymous login
3. *http://<server>:8080/luntbuild/j_acegi_security_check.do?j_username=anonymous&j_password=anonymous* - does not need to be registered

Chapter 3. Luntbuild Home Page

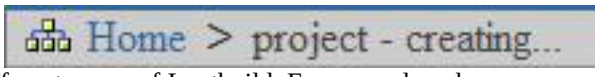
After you login, Luntbuild Home page will display:



There are five tabs on Home page:

1. *Builds* - shows all Luntbuild builds
2. *Projects* - shows all Luntbuild projects
3. *Users* - shows all Luntbuild users
4. *Properties* - shows general Luntbuild properties
5. *Administration* - shows Luntbuild administration tasks like import/export

Just click on the tab and the appropriate tab page will display.

Top area of the page contains navigation area 

that will help you to navigate quickly throughout the different pages of Luntbuild. For example, when you are creating a new project (by clicking New icon on Project tab page), you can jump quickly to Home page by clicking Home link in the navigation area.

If you run into problems while running Luntbuild, click on *system log* link in the upper right corner of each page. The Luntbuild's system log will display, that contains Luntbuild's and application server logging information. See chapter *Debugging Build Problems* for details about debugging Luntbuild problems.

The upper right corner of each page contains refresh icon , that toggles automatic page refresh on and off. It is good idea to switch refresh on, if you are tracking the status of the currently running build.

To logout from Luntbuild, just click on *logout* link in the upper right corner.

Chapter 4. About to create a project

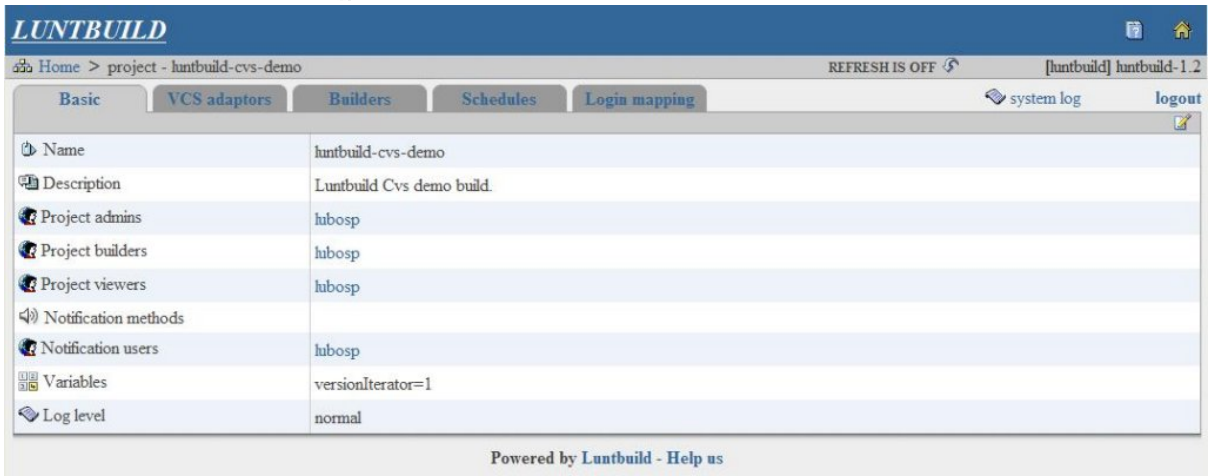
OK, we managed to login to Luntbuild and we are ready to create our first project. The next chapter explains how to create a new project and specify basic project properties.

Chapter 5. Creating a Project

Click on Project Tab.

The project page shows all projects configured in the current Luntbuild instance. A project is a buildable unit configured with information such as Version Control System, project builders and schedules.

Click on the New Project icon in the upper right corner of the tab's frame.



Name	Provide a name to identify this project. The name will be used to identify this project, and cannot be changed later. Keep in mind that the name of the project will be used as a name of the sub-directory in Luntbuild's work and publish directories.
Description	Specify project description.
Project admins	Select the users who should be assigned the role of 'project admin'.
Project builders	Select the users who should be assigned the role of 'project builders'.
Project viewers	Select the users who should be assigned the role of 'project viewers'.
Notification methods	Select the notification methods for the builds of this project.
Notification users	Select the users who will get notified, when the builds of this project finish.
Variables	Define variables for this project with one variable definition per line, for example: a=1 b=2 Values of these variables can be referenced or assigned in an OGNL expressions, for example when constructing "next build version" property of the schedule. Numeric variables can even be increased or decreased, for example, if you have two schedules with the name "nightly" and "release" respectively, and you want the build of these two schedules to increase a global build version. You can define the following variables: versionPart=foo-1.0.0 iterationPart=1 And then set "next build version" of both schedules to be:

	${project.var["versionPart"]} (${project.var["iterationPart"].increaseAsInt()})
	This way, build version of both schedules will consist of two parts: the first part takes the value of the variable "versionPart", and the second part takes the value of the variable "iterationPart" and this part will increase with every build. Thus the build version of the consequent builds will look like:
	foo-1.0.0 (build 1) foo-1.0.0 (build 2) foo-1.0.0 (build 3) ...
	You can define many other types of versioning strategies, refer to *next build version* property of a schedule for details.
Log level	Select the log level for this project.

Chapter 6. Creating project to build Luntbuild

Now we know how to create a project, so lets create project that will build Luntbuild. Following are basic properties for our project:

Basic properties for Luntbuild project

Name: luntbuild
Description: Luntbuild build
Notification methods: Email
Notification users: <users who checked in code recently>

Now we need to create CVS Adaptor, to let Lundbuild know how to access project buildable artifacts (source files etc.). The next chapter explains how to create CVS Adaptor and CVS module. Then we will create CVS Adaptor and module for our Luntbuild build project.

Chapter 7. Create CVS Adaptor

Setting Cvs connection information.

In order to use this adaptor, install appropriate Cvs client based on your platform from http://www.cvshome.org or http://www.cvsnt.org if you are using Windows platform.

Note

Please keep time of the build server machine in sync with the Cvs server machine to allow build server to detect repository changes in Cvs server more accurately. Please make sure that times recorded in the Cvs revision log are in UTC time format instead of local time format.

Here is the list of properties for this adaptor:

Cvs root	The Cvs root for this project, for example, :pserver:administrator@localhost:d:/cvs_repository. If you are using ssh, the :ext: protocol will need to be specified, and proper ssh environment needs to be set outside of Luntbuild. Please refer to your Cvs User's Guide for details.
Cvs password	The Cvs password for above Cvs root if connecting using pserver protocol.
Is cygwin cvs?	This property indicates whether or not the cvs executable being used is a cygwin one. The possible values are "yes" or "no". When omitted, the "no" value is assumed.
Disable "-S" option?	This property indicates whether or not the "-S" option for the log command should be disabled. The possible values are "yes" or "no". When omitted, the "no" value is assumed. The -S option used in the log command can speed up modification detection, however some earlier versions of Cvs do not support this option. In this case you should enter "yes" value to disable it.
Disable history command?	This property indicates whether or not to disable the history command when performing modification detection. The possible values are "yes" or "no". When omitted, the "no" value is assumed. Using the history command in conjunction with the log command can speed up modification detection, however some Cvs repositories may not hold history information of commits. In this case you should enter "yes" value to disable it.
Cvs executable path	The directory path, where your cvs executable file resides in. It should be specified here, if it does not exist in the system path.
Quiet period	Number of seconds the current VCS should be quiet (without checkins) before Luntbuild decides to check out the code of this VCS for a build. This is used to avoid checking out code in the middle of some other checkins. This property is optional. When left empty, quiet period will not be used before checking out code to build.

Setting Cvs module information.

Source path	Specify a path to retrieve from the Cvs repository, for example: testcvs/src.
Branch	Specify the branch for the above source path. This property is optional. When left empty, main branch is assumed.

Label	Specify the label for the above source path. This property is optional. If specified, it will take preference over branch. When left empty, latest version of the specified branch will be retrieved.

"Source path" represents a module path in the cvs repository, for example "/testcvs", "/testcvs/web", or "testcvs", but you can not define a "source path" using "/" or "\". "Branch" stands for a Cvs branch and "Label" stands for a Cvs tag. Only one of these properties will take effect for a particular module. If both of them are not empty, label will take preference over branch. If both of them are empty, Luntbuild will get the latest code from main branch for a particular module.

Chapter 8. Creating CVS adaptor and module for Luntbuild project

Now we know how to create a CVS adaptor and module, so lets create CVS adaptor and module that will access Luntbuild CVS repository at Sourceforge [http://sourceforge.net/projects/luntbuild/]. Following are CVS adaptor properties for our project:

CVS adaptor properties for Luntbuild project

Version Control System: Cvs
Cvs root: :pserver:anonymous@cvs.sourceforge.net:/cvsroot/luntbuild
Cvs password:
Is cygwin cvs?: yes(Windows)/no(Unix)

And here are the CVS module properties for our project:

CVS module properties for Luntbuild project

Source path: luntbuild

Now we need to create Ant builder, to be able to build Luntbuild 8-). The next chapter explains how to create an Ant builder. Then we will create Ant builder for our project.

Chapter 9. Create Ant Builder

Configuring Ant Builder.

Name	Provide a name to identify this builder, this name can be changed later.
Command to run Ant	Specify the command to run Ant (normally path to ant.bat or ant shell script). For example: /path/to/ant. String enclosed by ${...} will be interpreted as OGNL expression, and it will be evaluated before execution. Root object used for OGNL expression evaluation here is current Builder [../javadoc/com/luntsys/luntbuild/builders/Builder.html] object. **Note** A single argument that includes spaces should be quoted in order not to be interpreted as multiple arguments. **Note** From available Ant command line options, you should not specify the option "-buildfile" and "-logfile", which will be used by Luntbuild. Other options are allowed. You can modify the command to add Ant command line options and properties, for example -Ddebug=_debug.
Build script path	The path of the Ant build script. If this path is not an absolute path, it is assumed, that it is relative to the project work directory.
Build targets	Specify the target(s) to build. Use space to separate different targets (target name containing spaces should be quoted in order not to be interpreted as multiple targets). If not specified, the default target in the above Ant build file will be build. You can also use OGNL expressions (${...}) to pass variables as the target name. For example you can use ${build.schedule.name} to use different targets for different schedules. Root object used for OGNL expression evaluation here is current Builder [../javadoc/com/luntsys/luntbuild/builders/Builder.html] object.
Build properties	Define build properties here to pass into the ant build script. For example: buildVersion=${build.version} scheduleName=${build.schedule.name} You should set one variable per line. OGNL expression can be used to form the value provided it is enclosed by ${...}. Root object used for OGNL expression evaluation here is current Builder [../javadoc/com/luntsys/luntbuild/builders/Builder.html] object.
Environment variables	Environment variables to set before running this builder. For example: MYAPP_HOME=${build.schedule.workingDir} SCHEDULE_NAME=${build.schedule.name} You should specify one variable per line. OGNL expression can be inserted to form the value, provided they are enclosed by ${...}. Root object used for OGNL expression evaluation here is current Builder [../javadoc/com/luntsys/luntbuild/builders/Builder.html] object.
Build success condition	The build success condition is an OGNL expression used to determine, if the build of the current project was successful (root object used for OGNL

expression evaluation here is current Builder [../javadoc/com/luntsys/luntbuild/builders/Builder.html] object). If left empty, the *result==0 and logContainsLine("BUILD SUCCESSFUL")* value is assumed. When this expression evaluates to true, the build is considered successful. Here are some examples to demonstrate format of this OGNL expression:

result==0, here "result" represents return code of ant execution of the build file.
logContainsLine("^ERROR.")*, the expression will be true if the build's build log contains a line that matches the regular expression pattern "^ERROR.*". Please see http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html [http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html] for the format of the regular expressions.
The above expressions can be prefixed with a '!' character to inverse the value. For example, !logContainsLine("^ERROR.*") will be true if the build log does not contain a line that matches the specified pattern.
The above expressions can be joined into expression with "and", and "or". For example, the expression *result==0 and !logContainsLine("^ERROR.*")* will be true if Ant execution of the build returns 0, and the build log does not contain any line starting with "ERROR".

Chapter 10. Ant Builder for Luntbuild project

Now we know how to create Ant builder, so lets create one for our project:

Ant Builder properties for Luntbuild project

builder type: Ant
Name: luntbuild
Command to run Ant: <your ant installation location>
Build script path: luntbuild/build/build.xml
Build targets: clean installer

In order to make the builder run, we have to create a schedule that will tell Luntbuild how and when to build our project. The next chapter explains how to create a schedule, and then we will create a schedule for our Luntbuild project.

Chapter 11. Create Schedule

Edit Schedule.

Schedules are used to initiate/trigger builds either non-interactively or manually.

Each build needs a work directory to checkout the artifacts from VCS repository. Following are the rules that Luntbuild uses to construct work directory:

1. Main Luntbuild *work directory* is used as a root of all Luntbuild projects.
2. Each schedule allows you to define its work directory. By default, this directory is a subdirectory named using the project name under Luntbuild's top level work directory.
3. VCS modules contain source path that is appended after the schedule work directory.

For example if Luntbuild's work directory is */luntbuild-install-dir/work*, project name is *myproject*, schedule subdirectory is *myscheduleworkdir*, and VCS source path is *source*, then absolute path of the build's work directory for given schedule is */luntbuild-install-dir/work/myproject/myscheduleworkdir/source*.

Why is this important? Because of following reasons:

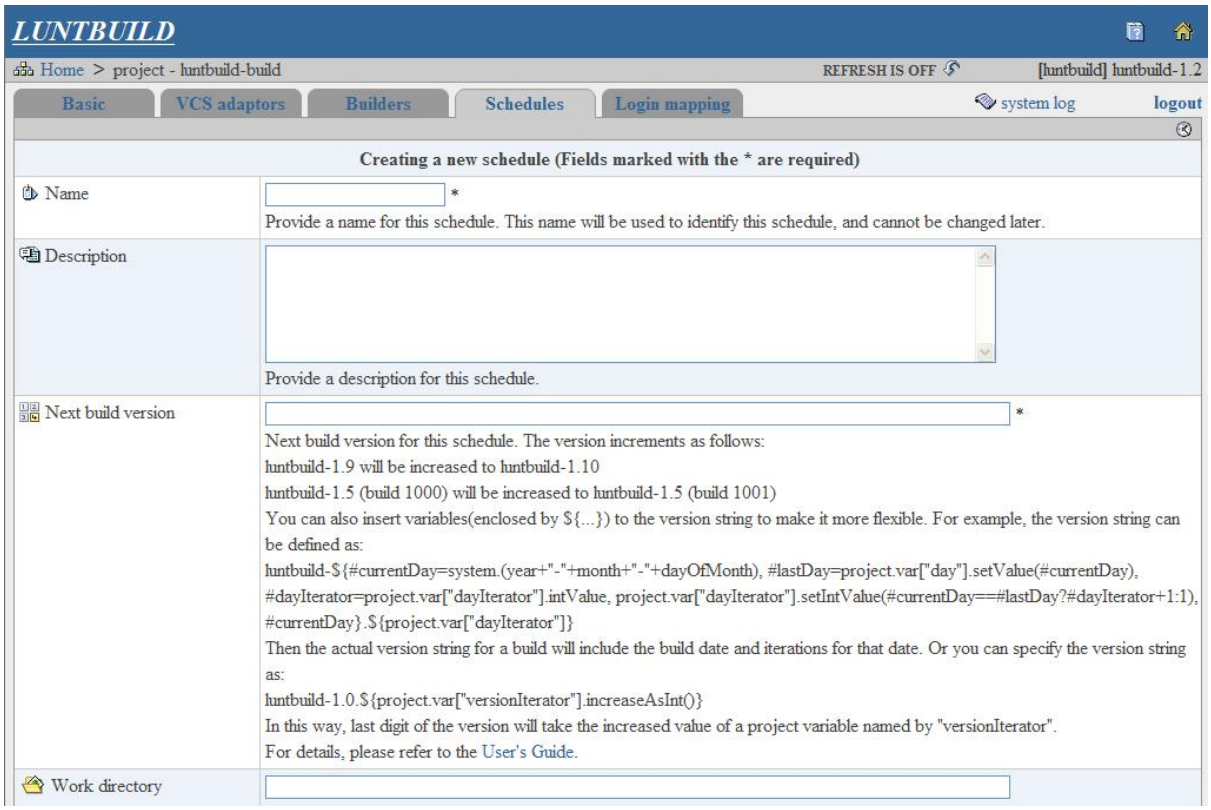
The build's work directory can be shared between multiple schedules of the same project. In this case the builds of those schedules use the same work directory, thus saving the disk space. Luntbuild guarantees that builds that share the same work directory cannot be executed at the same time. If first build using the shared work directory starts, all additional builds that share the same work directory are entered to the pending build queue, and they are executed only after currently executing build finishes.
If the build's work directory is not shared with other schedules of the same project, contents of the VCS modules for the given project is checked multiple times (to multiple work directories), thus consuming more disk space and possibly taking more time to checkout the contents of the VCS modules. Advantage of this approach is, that builds using different work directories (for the same project) can be executed in parallel.

Each build also uses its publish directory to store the build artifacts like build log and revision log. Following are the rules that Luntbuild uses to construct publish directory:

1. Main Luntbuild *publish directory* is used as a root of all Luntbuild projects.
2. Project name is used to define subdirectory in the main *publish directory*.
3. Schedule name is used to define subdirectory in the project subdirectory.
4. Build version string is used to create subdirectory in the schedule subdirectory. This subdirectory contains build log *build_log.txt*, *build_log.html*, *build_log.xml* and revision log *revision_log.txt*, *revision_log.html*, *revision_log.xml*, and two subdirectories *artifacts* and *junit_html_report*. Subdirectory *artifacts* can be used by you to store any other additional artifacts, subdirectories *junit_html_report* is used to store results of JUnit testing.

For example if Luntbuild's publish directory is */luntbuild-install-dir/publish*, project name is *myproject*, schedule name is *myschedule*, and current build version is *myapp-1.2.0*, then absolute path of the build's publish directory for given schedule is */luntbuild-install-dir/publish/myproject/myschedule/myapp-1.2.0*.

To create a schedule, click on Schedules Tab, and click on New Schedule icon in the upper right corner of the tab's frame.


LUNTBUILD
Home > project - luntbuild-build
REFRESH IS OFF
[luntbuild] luntbuild-1.2
Basic
VCS adaptors
Builders
Schedules
Login mapping
system log
logout
Creating a new schedule (Fields marked with the * are required)
Name
*
Provide a name for this schedule. This name will be used to identify this schedule, and cannot be changed later.
Description
Provide a description for this schedule.
Next build version
*
Next build version for this schedule. The version increments as follows:
luntbuild-1.9 will be increased to luntbuild-1.10
luntbuild-1.5 (build 1000) will be increased to luntbuild-1.5 (build 1001)
You can also insert variables(enclosed by ${...}) to the version string to make it more flexible. For example, the version string can be defined as:
luntbuild-${#currentDay=system.(year+"-"+month+"-"+dayOfMonth), #lastDay=project.var["day"].setValue(#currentDay), #dayIterator=project.var["dayIterator"].intValue, project.var["dayIterator"].setIntValue(#currentDay==#lastDay?#dayIterator+1:1), #currentDay}.${project.var["dayIterator"]}
Then the actual version string for a build will include the build date and iterations for that date. Or you can specify the version string as:
luntbuild-1.0.${project.var["versionIterator"].increaseAsInt()}
In this way, last digit of the version will take the increased value of a project variable named by "versionIterator".
For details, please refer to the User's Guide.
Work directory

Chapter 12. Schedule for Luntbuild project

Now we know how to create a schedule, so lets create one for the Luntbuild project:

Schedule properties for Luntbuild project

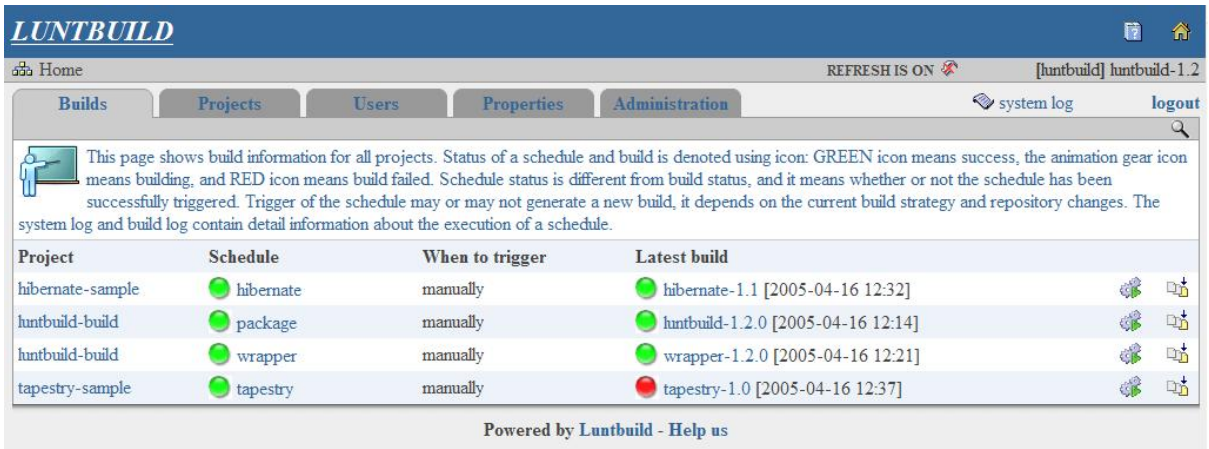
Name: luntbuild
Next build version: luntbuild-0
Trigger type: manual
Build necessary condition: always
Associated builders: luntbuild
Build type: clean
Label strategy: do not label

Warning

It is very IMPORTANT to set *Label strategy* to *do not label*. This is because we use anonymous access to Sourceforge Luntbuild repository, which is read only.

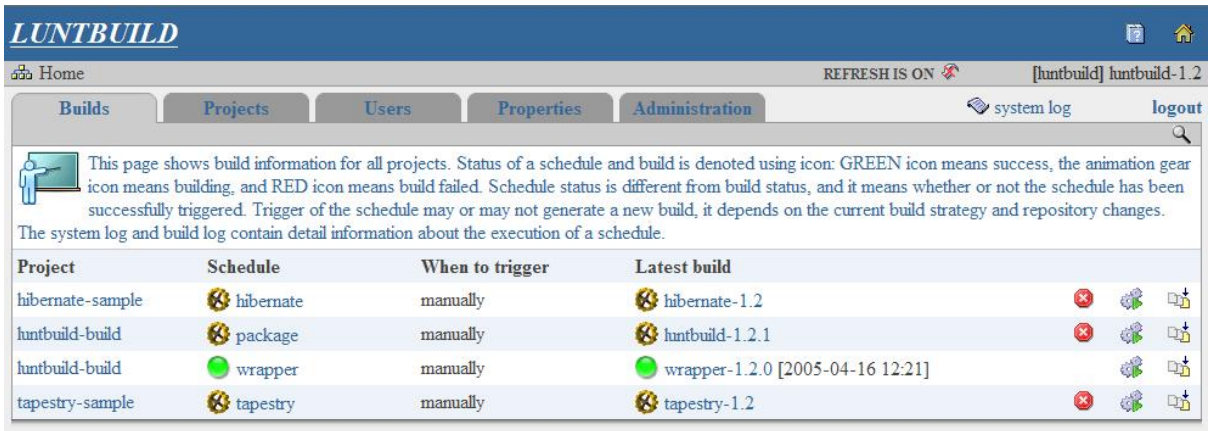
Now we are finally ready to start the build. The next chapter explains how to view all available builds. Lets click on Home link and then on Builds tab. You will see *luntbuild* schedule, which was not build yet. So lets click on  icon on the right side, and *Save* button at the bottom of the next page. Make sure *REFRESH IS ON* and click on *luntbuild-0* link. Then you can explore build log by clicking on *build log* link.

Chapter 13. Snapshot of all Build schedules



This page shows all build schedules configured in Luntbuild. The "Project" field identifies a project this build schedule belongs to. The "Schedule" field specifies a schedule this build uses. The "When to trigger" field specifies the condition that causes the build schedule to start execution. The "Latest build" field specifies the most recent build instance for this build schedule. The last field contains two icons. The rightmost icon gives access to all history build instances for this build schedule. Icon just left to the "history builds" icon is "run manually icon . You can start the build manually by clicking on this icon. When the "manually" started build is running a "stop" icon appears. You can stop the "manually" started build, by clicking on this icon.

Following is an example of all build schedules with some of the builds currently running:



There is a search link icon on the right top side of this page. You can follow this link to find particular builds, and you can perform operations on the found builds, such as you can delete the listed builds.

Icon to the left of the schedule indicates the execution status of the schedule. The schedule execution status is different from the build status. It indicates whether or not the schedule has been successfully triggered. Trigger of the schedule may or may not generate a new build, it depends on the current build strategy and repository changes. The schedule execution status may "fail" while the build succeeds, for example, due to an error while sending the notification mail. On the other side the schedule execution status may be "successful" although the build failed, because the schedule has been successfully triggered, and the build itself failed. Details about execution of a schedule can be found in the system log, which can be accessed using the "system log" link at the top of every page.

There are two types of build, *clean* build and *incremental* build. To perform a *clean* build, Luntbuild first purges

the project work directory, and then performs a full checkout of the project's VCS modules. To perform an *incremental* build, Luntbuild only updates source code checked out by previous build(s). The intermediate build files (e.g. .class files) are not purged before the new build. An incremental build is fast, but it might be less reliable. For example, if someone have deleted a file from Version Control System, this may not get reflected in an incremental build.

There are four build strategies, *build when necessary*, *build always if failed*, *build always*, and *do not build*.

build when necessary

Performs build only when there are any changes detected since the last build for this schedule. Changes since the last build exist if the following conditions are met:

1. Current build is the first build for the current schedule.
2. The VCS setting has changed since last build.
3. If the VCS adaptor is Clearcase UCM adaptor and "what to build" property is set to a value other than "latest", changes exist if related baselines have changed. For example, if "what to build" is set to "recommended baselines" and the Clearcase admin has recommended different set of baselines since the last build, changes exist, causing the execution of the next build.
4. Head revisions of the project files (or directories) have changed in the repository, and the current project VCS setting uses HEAD revisions.

build always if failed

Always performs the build, if the last build has failed. However, if the last build is successful, the next build will only be performed, when there are any changes detected since the last build of this schedule.

build always

Always performs the build at the specified schedule trigger time regardless of the status of the last build, or changes in the repository.

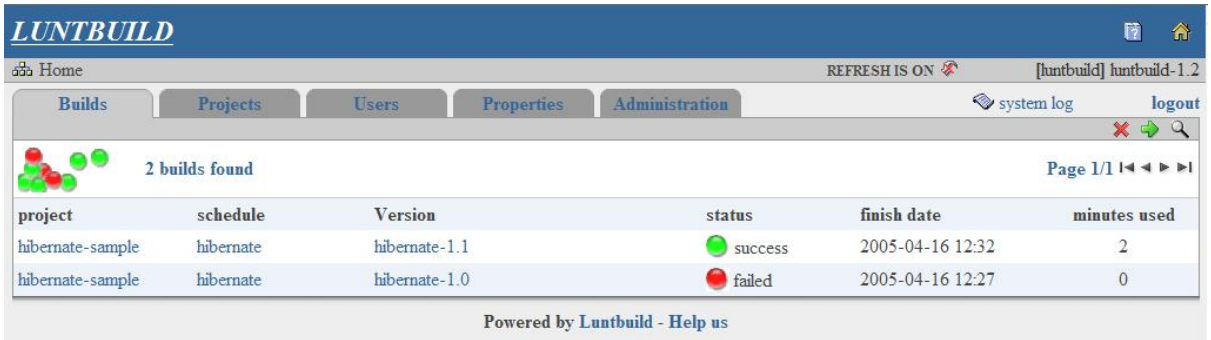
do not build

Does not perform the build in any circumstances. This strategy can be used to stop the schedule.

Note

The build strategy is only used when the trigger type of the schedule is not "manually".

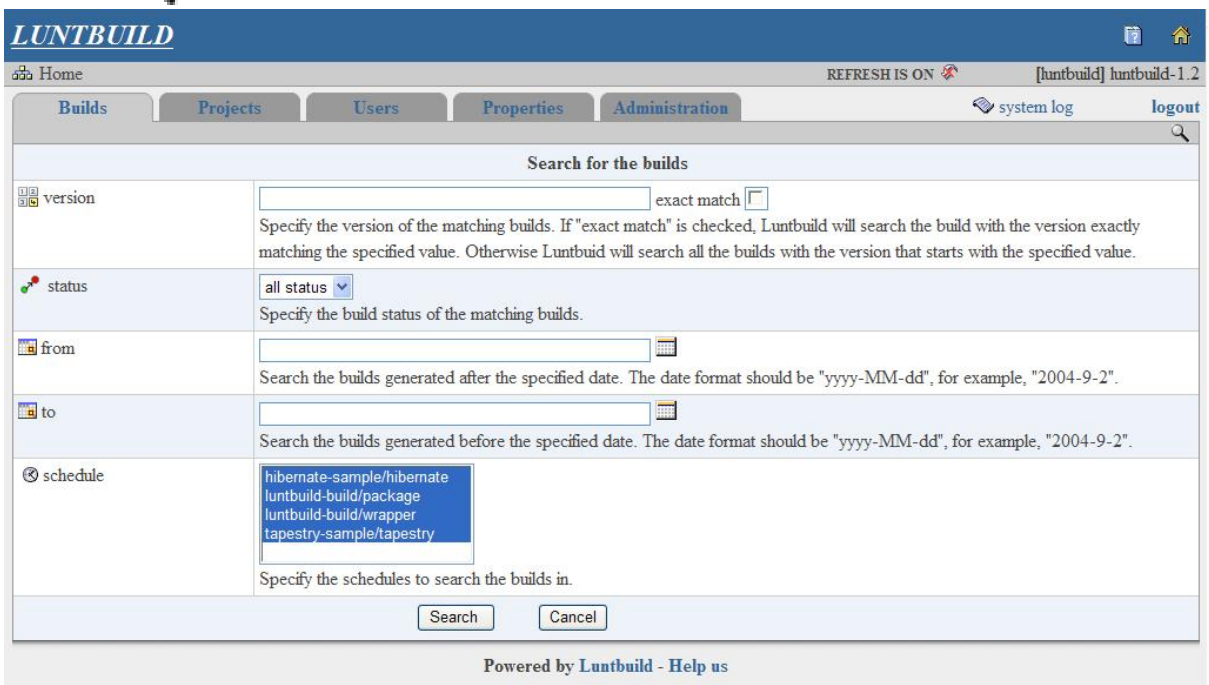
History builds for each schedule can be accessed by clicking the icon "history builds" on the right side of the schedule row. The list of all builds for the given schedule will display. You can access detailed information about a particular build by following the version hyperlink for that build. This will display a page:



In the "Build artifacts" area of this page, you can download artifacts for this particular build. You can also create a new directory as well as upload new artifacts. This can be useful for example if you want to supply patches for the specific build. You can also access the build log for this build. This log file can help you to diagnose any problems in case the build failed. The revision log records file or directory changes in the repository between previous build and this build. If you select to "label build" when generating this build, the "rebuild" icon with a link at the top area of this page will display. If you follow this link, you will be able to rebuild this build later. The rebuild process will use exactly the same VCS setting as when the build has been initially built. The

exact rebuild VCS setting will be written into the build log when you perform a rebuild. You can return Build Schedules page by clicking on the "Builds" tab.

You can search "history builds" form the Build Schedules or History Builds page by clicking on the "search build icon" . The following page will display



that will allow you to specify following search criteria:

Version — Specify the version of the matching builds. If "exact match" is checked, Luntbuild will search the build with the version exactly matching the specified value. Otherwise Luntbuild will search all the builds with the version that starts with the specified value.

Status — Specify the build status of the matching builds. One of the options is available:

all status
successful
failed
running

From — Search the builds generated after the specified date. The date format should be "yyyy-MM-dd", for example, "2004-9-2".

To — Search the builds generated before the specified date. The date format should be "yyyy-MM-dd", for example, "2004-9-2".

Schedule — Specify the schedules to search the builds in.

The page containing a list of the "history builds" matching the specified criteria will display.

You can delete the displayed list of builds by clicking in on icon .

You can move (or promote) the displayed list of builds by clicking in on icon , which will display "Move builds" page. On this page you can select a "Destination schedule" to move the displayed builds to. Specify the destination schedule for these builds. The move function enables you:

1. To save the builds before deleting a schedule or project.

2. To promote important builds. For example, we can promote a particular build from the "nightly" schedule

to the "release" schedule, to mark it as an external release.