Luntbuild - Build Automation and Management User's Guide

Luntbuild - Build Automation and Management User's Guide

Table of Contents

List of Tables

Chapter 1. Introduction

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]

While Luntbuild team would appreciate if you choose Luntbuild as your tool of choice for your continuous integration process, we realize, there are more tools available, and it is up to you to make the decision, which tool to use. Very nice comparison of continuous integration tools is available in *Continuous Integration Server Feature Matrix.* [http://damagecontrol.codehaus.org/Continuous+Integration+Server+Feature+Matrix]

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.

Chapter 2. Installing Luntbuild

Installation using Luntbuild installer is the easiest way to install Luntbuild. We recommend to use the installer for all Luntbuild standard installations including upgrades. See installer section and upgrade section for details.

In case you need to modify for some reason Luntbuild configuration files (web.xml, applicationContext.xml) with attributes/values beyond the attributes/values modified by Luntbuild installer, you might choose installation using Luntbuild distribution zip file. See zip section for details.

In case you need to modify and/or extend the source code implementation of Luntbuild, you might to choose to build Luntbuild distribution from the source code. See source section for details.

If you are upgrading from previous releases, please refer to upgrade section.

Using Luntbuild installer (with GUI)

Note

This is clean installation of Luntbuild. For upgrades from previous versions of Luntbuild please refer to upgrade section.

1. Make sure you have jdk1.4 or jdk1.5 installed, and add the directory which contains the java and jar executable into your system path **- IMPORTANT!**. Go to http://java.sun.com [http://java.sun.com/] for JDK download, if you didn't install it yet.

2. Make sure you get one of Luntbuild supported servlet containers or application servers installed (Servlet2.3 and JSP1.2 support are required), and make sure it has been stopped. Alternatively you can run Luntbuild in standalone mode.

3. Download Luntbuild installer from Luntbuild Sourceforge site [http://sourceforge.net/projects/luntbuild/]. This file is normally named as *luntbuild-xxx-installer.jar*, where xxx denotes current version.

4. Run command *java -jar luntbuild-xxx-installer.jar*. A GUI will display to guide you through the installation, and Luntbuild will install into the selected directory, let's say */opt/luntbuild*.

5. Deploy *luntbuild.war* (located in */opt/luntbuild* directory) into your servlet container or application server. Note, that if you selected the deployment directory of your servlet container or application server during installation, the installer will deploy *luntbuild.war* for you. If you plan to run Luntbuild in standalone mode (without servlet container), just start Luntbuild as described in standalone section.

6. Access the Luntbuild web application and you should be able to start your Luntbuild adventure, :D

Installation using zip distribution (without GUI)

Note

This is clean installation of Luntbuild. For upgrades from previous versions, please refer to upgrade section.

1. Download the Luntbuild zip distribution from Luntbuild Sourceforge site [http://sourceforge.net/projects/luntbuild/]. This file is normally named *luntbuild-xxx.zip*, where xxx denotes the current version.

2. Extract the zip file into the directory where you want to install Luntbuild, say */opt/luntbuild*. Edit the following files with your text editor:

   Edit file */opt/luntbuild/web/WEB-INF/web.xml*:
   Replace *$INSTALL_PATH* with your Luntbuild installation path (/opt/luntbuild here).

Replace *${sessionTimeout}* with your desired session timeout value (normally 30).
Edit file */opt/luntbuild/web/WEB-INF/applicationContext.xml*:
Replace *${luntbuildPassword}* with your desired site administrator password.

3. If you plan to run Luntbuild in standalone mode (without servlet container), just start Luntbuild as described in standalone section. Else copy all the contents under */opt/luntbuild/web* directory, and deploy it as a web application to your application server, or servlet container. For example, if you are using Tomcat servlet container:

   Make sure Tomcat has been stopped
   Change to Tomcat install dir: > *cd <tomcat install dir>/webapps*
   Make luntbuild directory: > *mkdir luntbuild*
   Copy luntbuild/web to webapps: > *cp -r /opt/luntbuild/web/* <tomcat install dir>/webapps/luntbuild*
   Start Tomcat

   Note. Do not create *luntbuild.war* file, just copy the contents under */opt/luntbuild/web* directory to the *luntbuild* directory in the appropriate web application directory of your application server, or servlet container.

4. Access the Luntbuild web application and you should be able to start your Luntbuild adventure, :D

Building Luntbuild from source distribution

1. Make sure you have jdk1.4 or jdk1.5 installed, and add the directory which contains the java and jar executable into your system path **- IMPORTANT!**. Go to http://java.sun.com [http://java.sun.com/] for JDK download, if you didn't install it yet.

2. Make sure you have Apache ant 1.6.1 (or higher) installed. Goto http://ant.apache.org for Ant download.

3. Download the source distribution from Luntbuild Sourceforge site [http://sourceforge.net/projects/luntbuild/]. This file is normally named *luntbuild-xxx-src.zip*, where xxx denotes the current version.

4. Extract the source distribution into a directory, let's say */yourhome/luntbuild-src*. Change to the directory */yourhome/luntbuild-src/build*, and run command *ant clean installer zip*. Then Luntbuild installer and zip distribution will both be generated into directory */yourhome/luntbuild-src/distribute*.

5. Follow the installation procedures in sections installer section, zip section, or upgrade section to install Luntbuild.

Upgrading from previous versions of Luntbuild

1. Assuming you've installed Luntbuild under */opt/luntbuild*. Backup the directory */opt/luntbuild/db* which contains your db files to another location. Alternatively, if you used external database to store Luntbuild data, backup the database. Backup all important build artifacts.

2. If you are upgrading from Luntbuild version 1.1.1, upgrade 1.1.1 web application with war file from here

. If you are upgrading from version 1.2 or higher just follow next steps.

Note. This only upgrades web application, and should not change anything under */opt/luntbuild*.

3. Access Luntbuild web application again, Select *Administration* tab, and export data into your specified file, let's say */yourhome/luntbuild-data.xml*. This file will be stored on the machine that runs your servlet container or application server hosting your Luntbuild application.

4. Follow the instructions in installer section, zip section, or source section of this file to install new release of Luntbuild into your previous Luntbuild installation directory, that is */opt/luntbuild* in this case.

5. Access the Luntbuild web application, click on *Administration* tab, and import from previously exported data file */yourhome/luntbuild-data.xml*. That will migrate data of previous Luntbuild installation to latest version.

   However, if your previous version is 1.1.1, some settings needs to be re-configured:

   The property *environment file* has been removed in 1.2. If you are using this property in 1.1.1, you need to extract contents of the environment file, and enter them as *Environment variables* property.
   All build success condition has been reset for 1.2 format and above.
   All build necessary condition has been reset for 1.2 format and above.
   Format of property "Next build version" has been changed, please verify your version string.
   Build properties passed to Ant build script has been changed, thus your Ant build script need to be changed to use new build properties.
   Ant builder command has been reset to default value. You may need to change it based on your Ant installation directory.

Running Luntbuild in standalone mode

You do not need servlet container or application server to run Luntbuild. Luntbuild comes with *build-in* servlet conatainer based on Jetty [http://jetty.mortbay.org/jetty/], a 100% Java HTTP Server and Servlet Container. Standalone Luntbuild launcher accepts three arguments:

hostname - name of the Luntbuild host machine
port - port number for servlet container
stop port - port number to use to stop launcher (optional, defaults to port+1)

Standalone Luntbuild stopper accepts two arguments:

hostname - name of the Luntbuild host machine
stop port - port number to use to stop launcher (optional, defaults to port+1)

You can run standalone Luntbuild launcher from the command line:

```
> cd <luntbuild-install-dir>
> bin/luntbuild.sh localhost 8888
```

or alternatively

```
> cd <luntbuild-install-dir>
> java -jar luntbuild-standalone.jar localhost 8888
```

To stop the standalone Luntbuild, you can use:

```
> cd <luntbuild-install-dir>
> bin/stop-luntbuild.sh localhost 8889
```

or alternatively

```
> cd <luntbuild-install-dir>
> java -cp luntbuild-standalone.jar com.luntsys.luntbuild.StandaloneStopper localhost 8889
```

While we sure you have several options, we suggest the excellent Java Service Wrapper [http://wrapper.tanukisoftware.org/doc/english/introduction.html] to run standalone Luntbuild. You'll get a clean, cross-platform way to start/stop/restart your services. On Unix, you'll get an init-style script, and on Windows you'll be able to integrate your app as a system service if you like.

Backing up Luntbuild data

It is very good idea to backup Luntbuild data so they can be restored in case they get corrupted. The backup strategy might depend on the type of the database you are using.

The database independent way to backup Luntbuild data is to use Export function in Administation tab. Unfortunatelly there is currently not an automated way to create a backup, it has to be created by login as Luntbuild administrator and perform Export.

In case your Luntbuild installation uses HSQLDB database (default) the database data is stored in <luntbuild install>/db directory. It is highly recommended to backup this directory on regular basis, if you do not want to use Export method mentioned above.

If you are using an external database (MySql, PostgreSql), please use the database backup/restore procedure, if you do not want to use Export method above.

Chapter 3. Configuring Luntbuild to use database

Luntbuild uses a database to make important Luntbuild data persistent. It uses ORM package Hibernate [http://hibernate.org] to access the persistent data and as a framework to access a database.

HSQL database

Luntbuild uses HSQL DB [http://hsqldb.org/] as default database running inside Luntbuild (in process mode). Additional in process (embedded) database supported by Luntbuild is H2 [http://www.h2database.com/html/frame.html]. Luntbuild also supports following external (client/server mode) databases:

MySql
ProgreSql
SqlServer
Oracle
Derby
H2
For external (client/server mode) databases, you will have create initial Luntbuild database by executing SQL scripts in appropriate db/... subdirectories.

Database definitions and data are located in the db directory of the Luntbuild installation.

HSQL DB [http://hsqldb.org/] uses db/luntbuild.script to define DB layout. Following files are also used, see HSQL DB documentation [http://hsqldb.org/web/hsqlDocsFrame.html]:

db/luntbuild.data
db/luntbuild.bakup

H2 [http://www.h2database.com/html/frame.html] embedded database uses luntbuild-h2-data* files located in db directory.

Sometimes, it is useful to look at the data in the DB to find out issues. Following are the steps to run a SQL client to examine data in the HSQL DB:

Download HSQLDB 1.7.3.3 from www.hsqldb.org [http://hsqldb.org/]. (Latest version 1.8.0 should also work)
Extract the downloaded package.
Change into the demo folder, edit the file "runServer.bat", replace the original contents with this line:

```
@java -classpath ../lib/hsqldb.jar org.hsqldb.Server -database <db path>
```

<db path> should be replaced with your path to Luntbuild DB. For example, if you are installing Luntbuild in D:\luntbuild, you should specify <db path> as D:\luntbuild\db\luntbuild.

Run "runserver.bat" to start up the database server.
Run "runManager.bat", from the popup dialog, select "HSQL Database Engine Server" as "Type" option, then you'll able to run SQL commands to examine contents of the DB.

If you are using Eclipse [http://eclipse.org] you can use DB plugin Quantum DB [http://quantum.sourceforge.net/] and the set the connection:

Table 3.1. QuantumDB HSQL DB Eclipse Configuration

Parameter	Value
Jdbc Url	jdbc:hsqldb:file:D:\luntbuild-13alpha\db\luntbuild
User	sa

Parameter	Value
Password	
Driver Path	<app-server>/webapps/luntbuild/WEB-INF/lib/hsqldb.jar
Driver Class	org.hsqldb.jdbcDriver
Driver Type	HSQL (Hypersonic)

Note

Many Luntbuild tables contain binary data which can not be displayed through the client, for example, if you run command "select * from LB_BUILD", nothing will be displayed. You should only specify non-binary columns here, for example, "select LB_ID, LB_NAME from LB_BUILD", etc.

MySql database

MySql [http://www.mysql.com/] database can be used as persistent storage for Luntbuild. The best way to configure Luntbuild to use MySql is to use Luntbuild installer, select MySql from Database install page, and fill appropriate database attributes. Alternatively you can configure <app-server>/webapps/WEB-INF/jdbc.properties for MySql (see jdbc.mysql.properties in the same directory):

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc.url=jdbc:mysql://localhost:3306/luntbuild
jdbc.username=luntbuild
jdbc.password=luntbuild
hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

Then you have to create Luntbuild database as follows:

```
> cd <luntbuild-install>
mysql> grant all on luntbuild.* to sa@localhost.localdomain identified by 'mypassword';
mysql> drop database luntbuild;
mysql> create database luntbuild;
mysql> use luntbuild;
mysql> source db/mysql/luntbuild.sql;
```

To manage MySql consider using free version of SQLyog [http://www.webyog.com/sqlyog/download_sqlyogfree.html]. If you are using Eclipse [http://eclipse.org] you can use DB plugin Quantum DB [http://quantum.sourceforge.net/] and set the connection:

Table 3.2. QuantumDB MySql Eclipse Configuration

Parameter	Value
Jdbc Url	jdbc:mysql://localhost:3306/luntbuild
User	
Password	
Driver Path	<app-server>/webapps/luntbuild/WEB-INF/lib/mysql-connector-java-3.1.7-bin.jar
Driver Class	com.mysql.jdbc.Driver
Driver Type	MySQL

PostgreSQL database

PostgreSQL [http://www.postgresql.org/] database can be used as persistent storage for Luntbuild. The best way to configure Luntbuild to use PostgreSQL is to use Luntbuild installer, select PostgreSQL from Database install page, and fill appropriate database attributes. Alternatively you can configure <app-server>/webapps/WEB-INF/jdbc.properties for PostgreSQL (see jdbc.postgresql.properties in the same directory):

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://localhost:5432/luntbuild
jdbc.username=luntbuild
jdbc.password=luntbuild
hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Then you have to create Luntbuild database as follows:

```
> psql  -h localhost -p 5432 postgres "admin"
postgres=# CREATE USER luntbuild;
postgres=# ALTER USER luntbuild PASSWORD 'luntbuild';
postgres=# CREATE DATABASE luntbuild WITH OWNER luntbuild;
postgres=# source luntbuild/db/postgresql/luntbuild.sql;
```

I use free pgAdminIII that comes with Postgres installation to manage Postgres database. If you are using Eclipse [http://eclipse.org] you can use DB plugin Quantum DB [http://quantum.sourceforge.net/] and set the connection:

Table 3.3. QuantumDB PostgreSQL Eclipse Configuration

Parameter	Value
Jdbc Url	jdbc:postgresql://localhost:5432/luntbuild
User	
Password	
Driver Path	<app-server>/webapps/luntbuild/WEB-INF/lib/postgresql-8.1-404.jdbc3.jar
Driver Class	org.postgresql.Driver
Driver Type	Postgres

SqlServer

SqlServer including Express [http://msdn.microsoft.com/vstudio/express/sql/] (free) edition is now supported. I use free SQL Server Management Studio Express for database management.

The best way to configure Luntbuild to use SqlServer is to use Luntbuild installer, select SqlServer from Database install page, and fill appropriate database attributes. Alternatively you can configure <app-server>/webapps/WEB-INF/jdbc.properties for SqlServer (see jdbc.sqlserver.properties in the same directory):

```
jdbc.driverClassName=net.sourceforge.jtds.jdbc.Driver
jdbc.url=jdbc:jtds:sqlserver://localhost:1525/luntbuild
hibernate.dialect=org.hibernate.dialect.SQLServerDialect
jdbc.username=sa
password=your password
```

If you are using Eclipse [http://eclipse.org] you can use DB plugin Quantum DB [http://quantum.sourceforge.net/] and set the connection:

Table 3.4. QuantumDB SqlServer Eclipse Configuration

Parameter	Value
Jdbc Url	jdbc:jtds:sqlserver://localhost:1525/luntbuild
User	sa
Password	your password
Driver Path	D:\luntbuild\lib\jtds-1.2.jar
Driver Class	net.sourceforge.jtds.jdbc.Driver

Parameter	Value
Driver Type	Microsoft SQL Server

Oracle

Luntbuild has been tested with Oracle 10g [http://www.oracle.com/technology/products/database/oracle10g/index.html]. You can use Oracle Enterprise Manager to manage the database. Enterprise manager is available http://<oracle-machine>:1158/em/console/logon/logon.

The best way to configure Luntbuild to use Oracle is to use Luntbuild installer, select Oracle from Database install page, and fill appropriate database attributes. Alternatively you can configure <app-server>/webapps/WEB-INF/jdbc.properties for Oracle (see jdbc.oracle.properties in the same directory):

```
jdbc.driverClassName=oracle.jdbc.driver.OracleDriver
jdbc.url=jdbc:oracle:thin:@localhost:1521:luntbuild
hibernate.dialect=org.hibernate.dialect.OracleDialect
jdbc.username=sa
jdbc.password=your password
```

If you are using Eclipse [http://eclipse.org] you can use DB plugin Quantum DB [http://quantum.sourceforge.net/] and set the connection:

Table 3.5. QuantumDB Oracle Eclipse Configuration

Parameter	Value
Jdbc Url	jdbc:oracle:thin:@localhost:1521:orcl
User	luntbuild
Password	luntbuild
Driver Path	D:\luntbuild\lib\ojdbc14.jar
Driver Class	oracle.jdbc.driver.OracleDriver
Driver Type	Oracle

Derby

Hibernate does not officially supports Derby. Embedded version of Derby [http://db.apache.org/derby/] fails to work with Hibernate and Luntbuild (caused by incompatibilities with BLOB data type). Client/Server version of Derby [http://db.apache.org/derby/] was tested with Hibernate/Luntbuild and it is working as expected.

If you are using Eclipse [http://eclipse.org] you can download and install Apache Derby plugin [http://db.apache.org/derby/integrate/plugin_howto.html].

The best way to configure Luntbuild to use Derby is to use Luntbuild installer, select Derby Client from Database install page, and fill appropriate database attributes. Alternatively you can configure <app-server>/webapps/WEB-INF/jdbc.properties for Derby (see jdbc.derby-cs.properties in the same directory):

```
jdbc.driverClassName=org.apache.derby.jdbc.ClientDriver
jdbc.url=jdbc:derby://localhost:1527/luntbuild
hibernate.dialect=org.hibernate.dialect.DerbyDialect
jdbc.username=luntbuild
jdbc.password=luntbuild
```

If you are using Eclipse [http://eclipse.org] you can use DB plugin Quantum DB [http://quantum.sourceforge.net/] and set the connection:

Table 3.6. QuantumDB Derby Eclipse Configuration

Parameter	Value
Jdbc Url	jdbc:derby://localhost:1527/luntbuild
User	luntbuild
Password	luntbuild
Driver Path	D:\luntbuild\lib\derbyclient.jar
Driver Class	org.apache.derby.jdbc.ClientDriver
Driver Type	Apache Derby

H2

H2 [http://www.h2database.com/html/frame.html] is database written in Java, available in both embedded and client/server mode. Based on performance comparison [http://www.h2database.com/html/performance.html] H2 is very fast and performs well with Luntbuild in both embedded and client/server mode. H2 comes with web client management console [http://www.h2database.com/html/quickstartText.html].

The best way to configure Luntbuild to use H2 is to use Luntbuild installer, select H2 Embedded or H2 Client from Database install page, and fill appropriate database attributes. Alternatively you can configure <app-server>/webapps/WEB-INF/jdbc.properties for H2 (see jdbc.h2-embedded.properties or jdbc.h2-cs.properties in the same directory):

```
jdbc.driverClassName=org.h2.Driver
jdbc.url=jdbc:h2:tcp://localhost:9092/luntbuild or ${h2Url}
hibernate.dialect=org.hibernate.dialect.HSQLDialect
jdbc.username=sa
jdbc.password=
```

If you are using Eclipse [http://eclipse.org] you can use DB plugin Quantum DB [http://quantum.sourceforge.net/] and set the connection:

Table 3.7. QuantumDB H2 Eclipse Configuration

Parameter	Value
Jdbc Url	jdbc:h2:tcp://localhost:9092/luntbuild or jdbc:h2:file:D:/luntbuild/db/luntbuild-h2-data
User	sa
Password	
Driver Path	D:\luntbuild\lib\h2.jar
Driver Class	org.h2.Driver
Driver Type	HSQL

Updating database during Luntbuild upgrade

The only supported way of updating Luntbuild database is to:

1. Export Luntbuild data from currently installed Luntuild.
2. Backup the Luntbuild database.
3. Delete the Luntbuild database.
4. Follow appropriate database installation to create database.
5. Follow appropriate Luntbuild installation.

6. Import previously exported data.

Chapter 4. Configuring Luntbuild to use LDAP

Luntbuild offers basic LDAP support for users authentication. LDAP can be configured in the installer, or, if you do not use installer to install Luntbuild, LDAP can be configured in the file <servlet-webapps>/luntbuild/WEB-INF/applicationContext.xml. The bean *luntbuildAuthenticationDAO* needs to be edited, for example:

```
<!-- luntbuild internal RDBMS based authentication DAO -->
<bean id="luntbuildAuthenticationDAO"
    class="com.luntsys.luntbuild.security.ApplicationInternalDAO">
       <property name="ldapHost" value="dis-corp.com"/>
       <property name="ldapPort" value="389"/>
       <property name="ldapUserDn" value="OU=All DIS Domain Users,DC=dis-corp,DC=com"/>
       <property name="ldapAuthentication" value="simple"/>
       <property name="ldapUserId" value="sAMAccountName"/>
       <property name="ldapUseLuntbuildOnFail" value="true"/>
       <property name="ldapCreateLuntbuildUser" value="true"/>
       <property name="ldapCanCreateProject" value="true"/>
       <property name="ldapCanViewProject" value="true"/>
       <property name="ldapCanBuildProject" value="true"/>
       <property name="ldapEmailAttrName" value="mail"/>
       <property name="ldapUrl" value="ldap://dis-corp.com"/>
       <property name="ldapPrefix" value=""/>
       <property name="ldapSuffix" value="@dis-corp.com"/>
</bean>
```

Following table expains the meaning of the LDAP configuration properties:

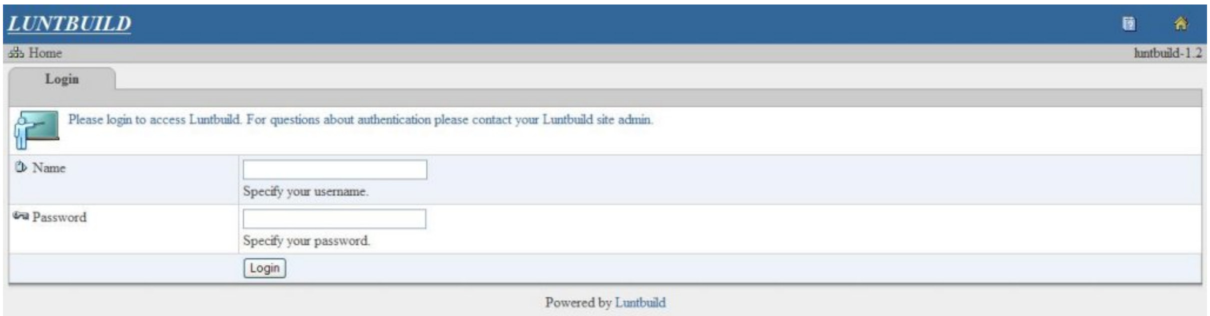
Table 4.1. LDAP configuration properties

Property	Meaning
ldapHost	LDAP server host
ldapPort	LDAP port (default is *389*)
ldapUrl	LDAP Url (defaults to "ldap://[host]:[port]") - this is used as the URL for making the connection, you can use LDAPS by setting the ldapUrl to "ldaps://my.ldap.server"
ldapPrefix	LDAP Prefix (defaults to "[userId]=") - this is prepended to the username for making the connection
ldapSuffix	LDAP Suffix (defaults to ",[userDn]") - this is appended to the username for making the connection
ldapUserDn	User DN address to access users database
ldapAuthentication	Authentication type (default is *simple*)
ldapUserId	User id LDAP identifier (default is *uid*)
ldapUseLuntbuildOnFail	*boolean*; if true uses Luntbuild authentication if LDAP fails
ldapCreateLuntbuildUser	*boolean*; if LDAP user is not a Luntbuild user, create Luntbuild user with the same name and password
ldapCanCreateProject	*boolean*; if true and if Luntbuild user is created, the user can create project and can administer existing projects
ldapCanBuildProject	*boolean*; if true and if Luntbuild user is created, the user can build existing projects
ldapCanViewProject	*boolean*; if true and if Luntbuild user is created, the user can view existing projects

Property	**Meaning**
ldapEmailAttrName	User email LDAP identifier (default is *mail*)

Chapter 5. Login to Luntbuild

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



LUNTBUILD

Home

luntbuild-1.2

Login

Please login to access Luntbuild. For questions about authentication please contact your Luntbuild site admin.

Name

Specify your username.

Password

Specify your password.

Login

Powered by Luntbuild

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.

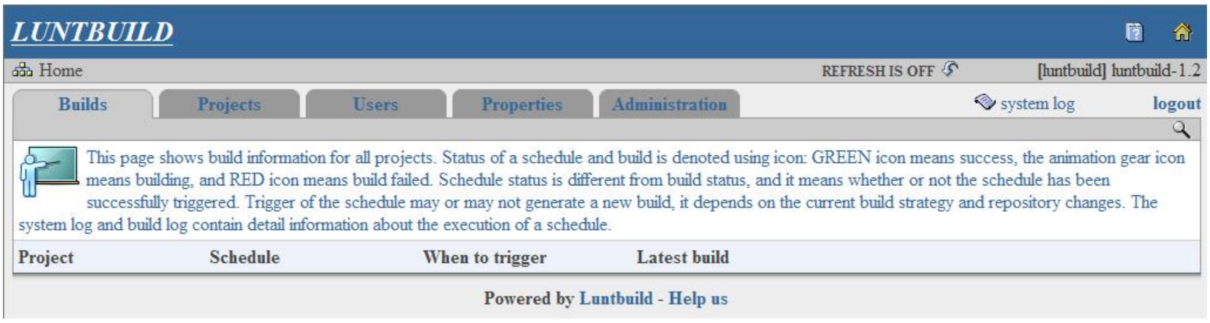
Anonymous users may can only view projects, builds and schedules.

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 6. Luntbuild Home Page

After you login, Luntbuild Home page will display:



This page shows build information for all projects. Status of a schedule and build is denoted using icon: GREEN icon means success, the animation gear icon means building, and RED icon means build failed. Schedule status is different from build status, and it means 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 system log and build log contain detail information about the execution of a schedule.

Project	Schedule	When to trigger	Latest build

Powered by Luntbuild - Help us

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 7. Setting General Luntbuild Properties

Properties listed in this tab are applied to all projects in Luntbuild. General Luntbuild Properties are explained in detail here:

Url for luntbuild servlet

The servlet url should be specified correctly here so it can be used in email notification and for Hessian remoting (for example used in Luntclipse). Normally this value should be http://<server>:<port>/luntbuild/app.do, where <server> is your build server name or ip address, and <port> is the port number you use to access Luntbuild. If this property is left empty, Luntbuild will use default value http://<server_ip>:8080/luntbuild/app.do, where <server_ip> is the actual ip address of the build server.

Warning

It is imperative to set the servlet Url properly. Otherwise Email notification and remote connection to Luntbuild will not work properly.

Work directory

You can optionally specify Luntbuild work directory. Work directory is the directory, where Luntbuild checks out artifacts from the version control system and performs a build. If not specified, Luntbuild uses the *work* sub-directory of Luntbuild installation directory. When a particular build schedule for given project is executed it creates a sub-directory in the Luntbuild *work* directory in the form *work*/<project-name>.

Publish directory

You can optionally specify Luntbuild publish directory. Publish directory is the directory, where Luntbuild publishes the output of the build process like the build log, and other build artifacts. If not specified, Luntbuild uses the *publish* sub-directory of Luntbuild installation directory. When a particular build schedule for given project is executed it creates a subdirectory in the Luntbuild *publish* directory in the form /*publish*<project-name>/<schedule-name>/<build-version>. For example, you top level publish directory might be a directory under *htdocs* directory of your Apache web server.

Page refresh interval

You can optionally specify a page refresh interval in seconds. If left empty, the default value will be 15 seconds.

SMTP host

You can optionally specify the SMTP mail host Luntbuild can use to send email notification. If this property is not specified, Luntbuild will use localhost as the default value.

SMTP user

This property is optional. If the SMTP host needs authentication, you should provide the user name here.

SMTP password

This property is optional. If the SMTP host needs authentication, you should provide the password here.

Luntbuild Jabber account

Set the Jabber related properties here if you want to notify user by Jabber. Luntbuild need a Jabber account in order to send out build notification messages.

Note

Connecting through proxy is not currently supported.

Note

This account needs to be different from your user's Jabber accounts. Keep this account logged off, and let your user login to his Jabber account using Gaim or any other IM client supporting Jabber.

Jabber server

You can optionally specify the Jabber host used by Luntbuild to send Jabber messages. If this property is not specified, Luntbuild will use localhost as the default value.

Jabber server type

You can optionally specify the type of Jabber server Luntbuild will communicate with. Valid entries are NORMAL, SSL, and GOOGLE.

Jabber server port

The Jabber server port to connect to; default is 5222.

Jabber user

The Jabber account name to be used for Luntbuild to login and send messages.

Jabber password

The Jabber account password to login.

Luntbuild MSN account

Set MSN Messenger related properties here if you want to notify user through MSN Messenger. Luntbuild needs a MSN Messenger account in order to send out build notification messages. For example *luntbuild@hotmail.com*.

Note

This account needs to be different from your user's MSN accounts. You also have to add the Luntbuild MSN account to your user's account(s) in order the user to get MSN messages. If Luntbuild MSN Account is *luntbuild@hotmail.com*, and your user account is *your-user@msn.com*, you have to add *luntbuild@hotmail.com* to the contacts of *your-user@msn.com*. Then keep the account *luntbuild@hotmail.com* logged off, and let your user login to *your-user@msn.com* using MSN Messanger, Gaim or any other IM client supporting MSN.

Note

Connecting through proxy is not currently supported.

Luntbuild MSN password

Password for the above MSN account.

Sametime Server

You can optionally specify the Sametime host used by Luntbuild to send Sametime messages. If this property is not specified, Luntbuild will use localhost as the default value.

Sametime User

The Sametime account name to be used for Luntbuild to login and send messages.

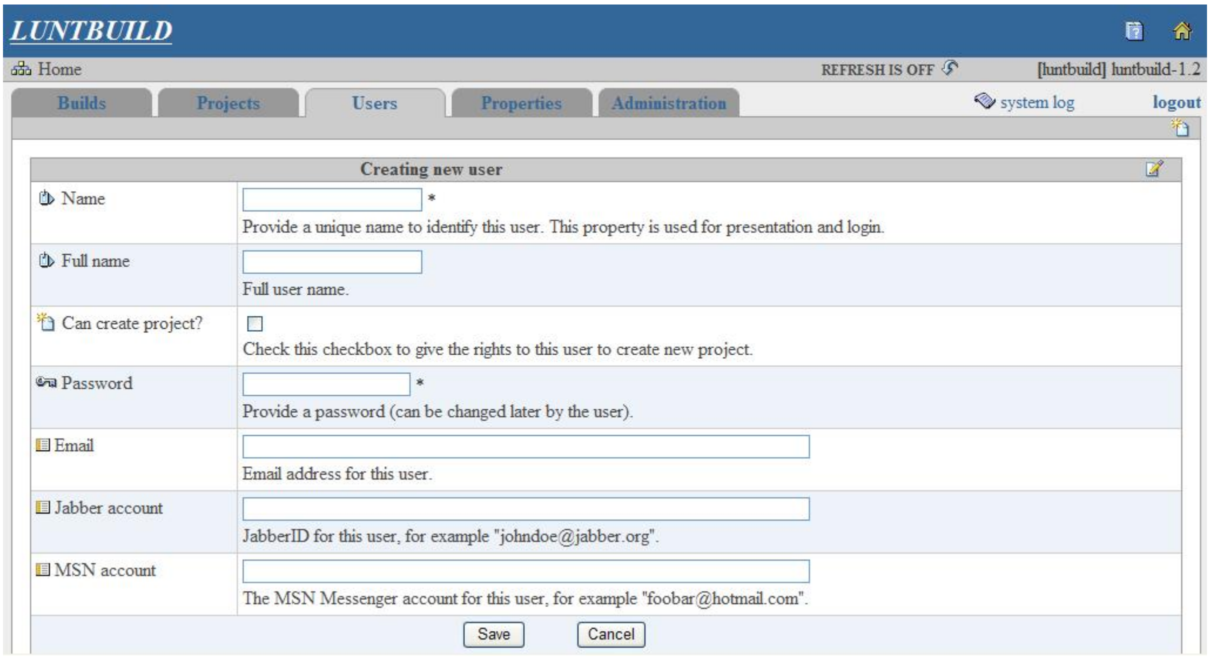
Sametime Password

The Sametime account password to login.

Chapter 8. Adding Luntbuild Users

Before you start using Luntbuild in a team with multiple members, or even if you are the only one using Luntbuild, it is good idea to create Luntbuild user(s) and give them appropriate privileges. Users will be notified of a build status and they will be authorized to access different parts of Luntbuild.

To create a Luntbuild user, click on the Users tab, and on the "new" icon, and following page will display:



Creating new user

Name	* Provide a unique name to identify this user. This property is used for presentation and login.
Full name	Full user name.
Can create project?	□ Check this checkbox to give the rights to this user to create new project.
Password	* Provide a password (can be changed later by the user).
Email	Email address for this user.
Jabber account	JabberID for this user, for example "johndoe@jabber.org".
MSN account	The MSN Messenger account for this user, for example "foobar@hotmail.com".

Fill the following information:

Name
Provides a unique name to identify this user. This property is used for presentation and login.

Full name
Full user name.

Can create project?
Check this checkbox to give the rights to this user to create new project.

Password
Provide an initial password (can be changed later by the user).

Jabber account
JabberID for this user, for example "johndoe@jabber.org". See *jabber.org* [http://www.jabber.org/about/overview.shtml] for details about Jabber.

Email
Email address for this user.

MSN account
The MSN Messenger account for this user, for example "foobar@hotmail.com".

Sametime account
The Sametime account for this user, for example "foobar".

Blog Type
Specify Blog Type. Supported blog types are: blogger or livejournal or metaweblog.

Blog URL
Specify full URL for your blog. For example http://www.blogger.com/api for blogger type, http://jroller.com/xmlrpc for metaweblog type, and http://www.livejournal.com/interface/xmlrpc for livejournal type.

Blog User
Blog User to use to access your blog.

Blog Password
Blog Password to use to access your blog.

Blog ID	Specify ID for your blog. Only used for blogger type.
Blog Category	Blog Category to use to send the blog (comma separated list). Only used for metaweblog type.

All Luntbuild notifiers use templates based on Velocity [http://jakarta.apache.org/velocity/] for configuration of the notification messages. There are two types of notification messages in Luntbuild. Schedule notification message, is schedule error/failure message, when schedule fails to execute. Build notification message is sent at the end of the build execution. The templates are located in following directories in the Luntbuild installation directory:

templates/blog
templates/email
templates/jabber
templates/msn
templates/sametime

Each directory contains file properties file *set-template.txt* that contains the templates to be used for build notifier and schedule notifier, for example:

```
buildTemplate=simple-build.vm
scheduleTemplate=simple-schedule.vm
```

Build templates support following variables:

luntbuild_webroot - Luntbuild root url
build_project - name of the project
build_project_desc - description of the project
build_schedule - schedule name
build_schedule_desc - schedule description
build_schedule_url - schedule url
build_schedule_status - schedule status (failed, success)
build_schedule_status_date - schedule status date
build_url - build url
build_version - build version
build_status - build status (failed, success)
build_isSuccess - true if build status is success
build_isFailure - true if build status is failed
build_start - build start date
build_end - build end date
build_duration - build duration in seconds
build_artifactsdir - build artifacts directory
build_publishdir - build publish directory
build_revisionlog_url - revision log url (Html)
build_revisionlog_text - revision log text (contents)
build_buildlog_url - build log url (Html)
build_buildlog_text - build log text (contents)
luntbuild_systemlog_url - system log url (Html)
build_junit_reportdir - JUnit report directory
build_type - build type (clean, increment)
build_labelstrategy - build label strategy
build_randomquote - random quote from quotes.txt file

Schedule templates support following variables:

luntbuild_webroot - Luntbuild root url
schedule_project - name of the project
schedule_project_desc - description of the project
schedule_name - name of the schedule
schedule_desc - description of the schedule
schedule_url - schedule url
schedule_status - schedule status (failed, success)

schedule_status_date - schedule status date
schedule_publishdir - schedule publish directory
luntbuild_systemlog_url - system log url (Html)
schedule_type - build type (clean, increment)
schedule_labelstrategy - build label strategy

Both templates also support OGNL expressions for Build or Schedule class, for example "version" for build template or "buildNecessaryCondition" for schedule template.

Note

Msn and Sametime notifiers do not support Html format, only text format.

Note

If you have problem with html format notification for Blog, Email, or Jabber notifier, you can use text only format. Just use files **-text.vm* templates in *set-template.txt* properties file.

New template for The Visual Studio is available in template *vs-build.vm*. The Visual Studio output scraper may be used to provide more detailed information regarding a Visual Studio build. To enable the scraper, make sure the command line used to trigger the build is visible in the build log. The scraper looks for a regular expression pattern containing the name of a .sln file followed by the “\build config#? parameter. If a match is found, the following variables are available to Velocity templates:

*vs_build_solutions*a List of MSVSSolution objects. This list is empty if no build command line was matched.

MSVSSolution contains the following properties:

Path – the path of the sln file.
Name – the name of the sln file.
Configuration – the configuration used for the build.
Succeeded – the number of projects successfully built.
Failed – the number of projects that failed to build.
Skipped – the number of projects skipped.
Projects – a List containing MSVSProject objects:

MSVSProject contains the following properties:

Name – the name of the vcproj file.
Errors – the number of errors in the project build.
Warnings – the number of warnings in the project build.

This is a snippet of a sample template:

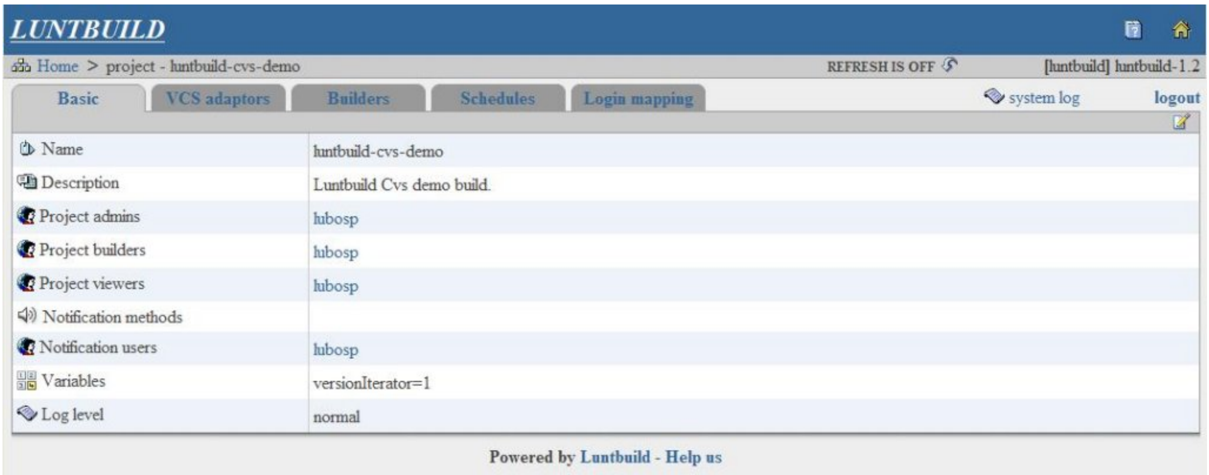
```
#foreach($solution in $build_vs_solutions)
  <tr>
    <td>$solution.Name</td>
    <td>$solution.Configuration</td>
  </tr>
  #foreach($project in $solution.Projects)
    <tr>
      <td>$project.Name</td>
      <td>$project.Error errors</td>
      <td>$project.Warning warnings</td>
    </tr>
  #end
  <tr>
    <td>$solution.Succeeded succeeded</td>
    <td>$solution.Failed failed</td>
    <td>$solution.Skipped skipped</td>
  </tr>
#end
```

Chapter 9. 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.



LUNTBUILD

Home > project - luntbuild-cvs-demo   REFRESH IS OFF   [luntbuild] luntbuild-1.2

Basic | VCS adaptors | Builders | Schedules | Login mapping   system log   logout

Name	luntbuild-cvs-demo
Description	Luntbuild Cvs demo build.
Project admins	lubosp
Project builders	lubosp
Project viewers	lubosp
Notification methods	
Notification users	lubosp
Variables	versionIterator=1
Log level	normal

Powered by Luntbuild - Help us

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 10. Creating Version Control System (VCS) Adaptor for the Project

- Select VCS Adaptors tab.
- Click on New VCS Adaptor icon in the upper right corner of the tab's frame.
- Select Version Control System.

Setting AccuRev connection information.

To download AccuRev go to http://www.accurev.com/download/index.htm [http://www.accurev.com/download/index.htm]. Here is the list of properties for this adaptor:

AccuRev port

The AccuRev port in the format of <servername>:<port>, where <servername> and <port> will be replaced by the actual AccuRev server name and the port number. This property is optional and overrides acclient.cnf values.

Note

Default values for AccuRev needs to be defined in acclient.cnf and wspaces files. Please consult AccuRev User's Guide for details.

Accurev executable path

The directory path, where your accurev 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 Base Clearcase connection information.

Using base Clearcase (i.e., non-UCM) is supported in two different manners: using snapshot views, or using dynamic views. In either case, much of the same configuration applies to both. This section will detail the configuration options available to you in either case.

Base Clearcase connection information.

The information presented in this section applies to either snapshot or dynamic views.

You should have Clearcase client installed on the build machine. Also you should make sure that the account running your application server or servlet container is able to access your Clearcase server and that it can make views. Here is the list of properties for this adaptor:

View tag

The name of the Clearcase view tag to use. If a view by this name does not already exist, it will be created for you. If you do not specify this, luntbuild will compute a view tag for you comprised of the luntbuild install location, project name, etc.

View stgloc name

Name of the Clearcase server-side view storage location which will be used as `-stgloc` option when creating Clearcase view for the current project. Either this property or "Explicit path for view storage" property should be specified.

View storage path

Explicit path for view storage. This property is required only when the "Clearcase view stgloc name" property is empty. In the case of snapshot views, if this is specified it will be used as -vws option instead of using the -stgloc option to create Clearcase view. In the case of dynamic views, this is the path to the view storage.

Note

This value should be a writable UNC path on Windows platform.

Config spec

Config spec used by Luntbuild to create Clearcase snapshot view for a build.

Modification detection config

This property will take effect if there are some LATEST versions from some branch to fetch in the above config spec. It is used by Luntbuild to determine, if there are any changes in the repository since the last build. This property consists of multiple entries, where each entry is of the format "<path>[:<branch>]". <path> is a path inside a VOB, which should be visible by the above config spec. Luntbuild will lookup any changes in any branch inside this path recursively, or it will lookup changes in the specified branch, if <branch> is specified. Multiple entries are separated by ";" or line terminator.

Options for mkview command

You may optionally specify extra options for the cleartool mkview sub command used by Luntbuild to create related Clearcase view for the current project. Options that can be specified here are restricted to -tmode, -ptime, and -cachesize. For example you can specify "-tmode insert_cr" to use Windows end of line text mode.

Cleartool executable path

The directory path, where your cleartool executable file resides in. It should be specified here, if it does not exist in the system path.

History format parameters

The revision log for each build is generated via the `lshistory` cleartool subcommand. Luntbuild includes `date:%d user:%u action:%e %n\\n` already. If you would like to append additional information, then you may enter additional format parameters here (e.g., `%c` to retrieve file checkin comments). See the Clearcase man pages on `fmt_ccase` for more information.

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.

Dynamic Clearcase connection information

In addition to the basic Clearcase connection information, the following configuration options are available when using dynamic views.

Mvfs path

This is the root path from which all Clearcase views are accessible. On Windows, for example, this defaults to `M:\\` .

Project path

Path relative to the view root in which the desired sources can be found. This is primarily useful for constraining the scope of the history gathering. If left blank, then the view root direcotry will serve as the directory from which history is gathered. Secondarily, it is also useful for determining what the Clearcase working directory should be (see below).

It's worth noting that making use Dynamic Clearcase probably requires some additional configuration within other tabs. Because your Clearcase elements are available within the confines of your dynamic view and are not physically copied to your local storage, the Schedule working directory computed on your behalf will not point to your Clearcase view directory. That said, it is unwise to set your Schedule's working directory to your Clearcase view directory since that directory may not be available for luntbuild's use. Instead, you should set the Builder's working directory to the desired location. If you explicitly set your view tag, then this is a simple exercise of using your Mvfs Path compined with your view tag. For example, with a view tag of `luntbuild` and an Mvfs path of `M:\\` , you could set your builder working directory to `M:\\luntbuild` .

To make things easier for you, a computed pseudo-property is available to you. You can make use of this via an OGNL expression within your builder's directory setting. For example, the following expression will retrieve the Clearcase working directory (see above): `${build.vcsList[0].getClearcaseWorkDir(build.schedule)}` . Note that this assumes a single Vcs adapter setup, or at least one where your dynamic ClearCase definition is first in the list.

Finally, a point of process. When using a Dynamic view, it is generally advisable to include a time rule in order to stabilize the contents of your dynamic view during the build process. If you're operating against `/main/LATEST` , the following makes a decent config spec: `/main/LATEST -nocheckout -time now` . Each time your view is "checked out," the view's config spec is reset, thus redefinining "now" to be the time at which the view's config spec was reset.

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 File system connection information.

Source directory

This is an optional property. If specified, changes can be detected in the source directory based on modification time, and modified files under this directory will be copied to the project work directory to perform build.

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 Perforce connection information.

You should have Perforce client installed on the build machine. Contact http://www.perforce.com for licensing information. Here is the list of properties for this adaptor:

Perforce port

The Perforce port in the format of <port>, or <servername>:<port>, where <servername> and <port> will be replaced by the actual Perforce server name and the port number.

User name

User name to access the above Perforce server. This user should have the rights to create and edit client specifications and to checkout and label code.

Password

Password for the above user. Can be blank, if your Perforce server does not use password based security.

Line end

Set line ending character(s) for client text files. The following values are possible:

local: use mode native to the client
unix: UNIX style
mac: Macintosh style
win: Windows style
share: writes UNIX style but reads UNIX, Mac or Windows style

This property is optional. If not specified, the value will default to "local".

P4 executable path

The directory path, where your p4 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 Subversion connection information.

In order to use this adaptor, Subversion client software should be installed on your build machine. You can download subversion from http://subversion.tigris.org [http://subversion.tigris.org/].

Note

Please keep time of the build server machine in sync with the Subversion server machine to allow build server to detect repository changes in Subversion server more accurately.

Here is the list of properties for this adaptor:

Repository url base

The base part of Subversion url, for example, you can enter "svn://buildmachine.foobar.com/", or "file:///c:/svn_repository", or "svn://buildmachine.foobar.com/myproject/othersubdirectory", etc. Other definitions such as tags directory, branches directory, or modules are relative to this base url.

Note

If you are using https:// schema, you should make sure that svn server certificate has been accepted permanently by your build machine.

Directory for trunk

Directory used to hold trunk for this url base. This directory is relative to the url base. Leave it blank, if you didn't define any trunk directory in the above url base.

Directory for branches

Directory used to hold branches for this url base. This directory is relative to the url base. If left blank, "branches" will be used as the default value.

Directory for tags

Directory used to hold tags for this url base. This directory is relative to the url base. If left blank, "tags" will be used as the default value.

Username

User name to use to login to Subversion.

Password

Password to use to login to Subversion.

Svn executable path

The directory path, where your svn 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 Clearcase UCM connection information.

You should have Clearcase client installed on the build machine. Also you should make sure that the account running your application server or servlet container is able to access your Clearcase server and that it can make snapshot views. Here is the list of properties for this adaptor:

View stgloc name

Name of the Clearcase view storage location, which will be used as -stgloc option when creating Clearcase view for this project.

Project VOB tag

Tag for the project vob, for example: \pvob1.

View storage path

Explicit path for view storage. This property is required only when the "Clearcase view stgloc name" property is empty. If specified, it will be used as -vws option instead of -stgloc option when creating Clearcase view. This view will be created automatically by the luntbuild system. Just make sure there is no other view with this name. Example: \\mycomputer\cc_vws1\luntbuild-mymodule.vws

Note

This value should be a writable UNC path on Windows platform.

UCM stream name

Name of the UCM stream.

What to build

Specifies baselines you want to build inside the stream. Multiple baselines are separated by space. The following values have particular meaning:

<latest>: means build with all the latest code from every component
<latest baselines>: means build with all the latest baselines from every component
<recommended baselines>: means build with all the recommended baselines
<foundation baselines>: means build with all the foundation baselines

Config Spec

Put your config spec in here. Add a load line after the config spec you use in your dynamic views for each directory you need, like this:

```
include \\server\ClearCase\configspecs\myconfigspec.txt
load \myvob\modules\build
load \myvob\modules\mymodule
```

Modification detection config

This property will only take effect when the "What to build" property equals to "latest". It is used by Luntbuild to lookup if there are any changes in the repository since the last build. This property comprises of multiple entries with each entry in the format "<path>:<branch>". <path> is a path inside a VOB, (but starting from the vob name, for example: \myvob\modules\mymodule:mymodule_dev_branch), which should be visible using the above config spec. Luntbuild will lookup any changes at any branch inside this path recursively, or it will lookup changes in the specified branch, if <branch> is specified. Multiple entries are separated by ";" or line terminator. Refer to the Clearcase User's Guide for details.

Note

If the branch is a subbranch, you don't need to specify the names of any "super"-branches, just the name of the actual branch is enough.

Options for snapshot view

You may optionally specify extra options for the cleartool mkview sub command used by Luntbuild to create related clearcase snapshot view for the current project. Options that can be specified here are restricted to -tmode, -ptime, and -cachesize. For example you can specify "-tmode insert_cr" to use Windows end of line text mode.

Cleartool executable path

The directory path, where your cleartool 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.

Note

The Clearcase UCM adaptor currently does not support labeling operation.

Setting Visual Sourcesafe connection information.

In order to use this VCS adaptor, visual sourcesafe need to be installed in your build machine. Download Visual Sourcesafe from http://download.microsoft.com. The following list of properties needs to be configured:

Note

In order to keep history command of Visual Sourcesafe accurate, time setting of all developer workstations, and the build server should be kept in sync.

Sourcesafe path

The directory where your srcsafe.ini resides in. For example: \\machine1\directory1. You should use explicit hostname eg. "machine1", not the ip address of the "machine1", or you should specify ip address in the Sourcesafe path.

Username

User name to use to login the above Sourcesafe database.

Password

Password for the above user name.

Datetime format

Specify the date/time format used for the Sourcesafe history command. This property is optional. If left empty, Luntbuild will use "M/dd/yy;h:mm:ssa" as the default value. The default value is suitable for English language operating systems using US locale. For other English speaking countries with different date format like UK, Australia, and Canada the Visual Sourcesafe Date format to use (assuming you're using the appropriate locale setup as Visual Sourcesafe honors the local locale settings) should be as follows:

'd/M/yy;H:mm'

If Luntbuild is running on non-english operating systems, use the following method to determine the datetime format:

Open Visual Sourcesafe *installed on your build machine*, select an existing VSS database and choose to view one of the projects with files in it. There should be a list of files shown with several fields including the "Date-Time" field. You should use the "datetime format" property from value specified in this field. For example, if one of the values of this field is "04-07-18 20:19", the "datetime format" property should be "yy-MM-dd;HH:mm". The *semicolon* between date and time format should be specified. You are encouraged to specify the property as "yy-MM-dd;HH:mm:ss" to add the accuracy. Take another example, if the value shown in Visual Sourcesafe is "7/18/04 8:19p", the "datetime format" should be "M/dd/yy;h:mma". Format "M/dd/yy;h:mm:ssa" would increase the accuracy in this case.

The following is a list of format character meanings copied from JDK document:

Table 10.1. Date/Time format characters

Character	Meaning	Example
y	Year	1996 ; 96
M	Month in year	July ; Jul ; 07
d	Day in month	10
a	Am/pm marker	p
H	Hour in day (0-23)	0
h	Hour in am/pm (1-12)	12
m	Minute in hour	30
s	Second in minute	55

For details about the format string, please refer to http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html [http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html]

Ss.exe path

The directory path, where your ss.exe 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.

Note

Make sure you are using English version of Visual Sourcesafe. If you must use other language version, please download source code of Luntbuild and modify below line of VssAdaptor.java:

```
    authorPattern = Pattern.compile("^User:(.*)Date:.*");
```

You need to change the author pattern according to the HISTORY command output of your Sourcesafe installation.

Setting StarTeam connection information.

For Windows platform, you will need to have a full installation of StarTeam SDK runtime (which will install some runtime dlls and put them in the Windows system path). Normally this is the part of StarTeam client installation. Please go to http://www.borland.com for licensing information. Here is the list of properties for this adaptor:

Project location

Location of a StarTeam project is defined as: <servername>:<portnum>/<projectname>, where <servername> is the host where the StarTeam server runs, <portnum> is the port number the StarTeam server uses, default value is 49201. <projectname> is a StarTeam project under this StarTeam server.

User

User name to login to the StarTeam server.

Password

Password to login to the StarTeam server.

Convert EOL?

The following values are possible:

all: all ASCII files will have their end-of-line characters adjusted to the EOL type of the local machine on checkout

no: the files will be checked out with whatever EOL characters are used on the server

This property is optional. If not specified, it will default to *yes*.

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.

Using multiple Version Control Adaptors.

For each project, you can define one or more of the above Version Control Systems. When a build is performed for such a project, contents of all repositories is checked out to the build's work directory. For example, you may have a project with its client module in a Cvs repository, and the server module in a VSS repository. This approach is also applicable for projects with modules in different repositories of the same VCS type. For example, you may have a project with the client module in a Cvs repository, and the server module in different Cvs repository.

Chapter 11. Creating VCS Module(s)

Click on New Module icon in upper right corner of the Modules frame. If you have multiple VCS modules defined, the retrieval process will start from the first module. Following modules will override previous modules if the part of the modules overlaps. For example, if you define module1 with destination path "/foo/bar", and later you define another module with destination path "/foo", the contents of module2 will override the contents of module1. But if module1 is defined with destination path "/foo", and module2 is defined with destination path "/foo/bar", only contents under directory "/foo/bar" will be overridden by module2.

Setting AccuRev module information.

Label
The label is the transaction number to which to sync. Specify the transaction number you want to build at.

Depot
The AccuRev depot to check the code out of.

Backing stream
The backing stream for this build module. The backing stream should be able to have streams created from it by the build user.

Build stream
The name of the stream to create from the backing stream. If it doesn't exist it will be created. A reference tree will be created from this stream with a '_reference' suffix appended to the build stream name.

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.

Setting Perforce module information.

Depot path
Specify the Perforce depot side path, such as "//depot/testperforce/...".

Label
Specify the label for the above depot path. This property is optional. When empty, the latest version (head) of the above depot path will be retrieved.

Client path
Specify the client side path, such as "//myclient/testperforce/...".

Note

To exclude files or directories, create a separate module for each exclusion and precede the Depot path property with a minus (-) sign, as follows:

```
    Depot path:  -//depot.side
    Client path: //client.side
```

The module definition for Perforce maps a repository "Depot path" to "Client path". Luntbuild also supports Perforce "Label" property. "Depot path" represents a path in Perforce repository, such as "//depot/testperforce/...". "Client path" represents a client path (where the contents of the depot path is checked out), such as "//myclient/testperforce/...". "Label" is a Perforce label, used if you want to retrieve a particular snapshot of given "Depot path", or it can be left empty to retrieve the head version of "Depot path". The client path defined in "Client path" does not need to exist. Luntbuild will create the path if it does not exist. The user specified in Perforce connection information at the project level should have enough access rights to create and edit Perforce client specification.

Setting Subversion module information.

Source path

Represents a path in the Subversion repository, for example "testsvn", "testsvn/web", or "/testsvn". When "branch" or "label" properties are defined, this path will be mapped to another path in the svn repository.

Branch

Specify the branch for above source path. This property is optional. When left empty, trunk is assumed.

Note

Subversion does not internally has the notion of branch. Value specified here will be used by Luntbuild to do url mapping for the above source path so that actual effect is just like a branch in Cvs.

Label

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

Note

Subversion does not internally has the notion of label. Value specified here will be used by Luntbuild to do url mapping for the above source path so that actual effect is just like a tag in Cvs.

Destination path

This property is optional. If specified, the contents from Subversion repository will be retrieved to the "destination path" relative to the project work directory. Otherwise the contents will be retrieved to "source path" (with no regard to "branch" or "label") relative to the project work directory.

"Source path" represents a path in the Svn repository, for example "testsvn", "testsvn/web", or "/testsvn". This path will be mapped to another path in the Svn repository based on other properties. In order to demonstrate this path mapping, we define following properties:

Repository url base: svn://localhost
Directory for trunk: trunk
Directory for branches: branches
Directory for tags: tags

We will examine the following module settings and give them the url mapping:

Trunk

Source path: testsvn/web
Branch: <empty>
Label: <empty>
Destination path: <empty>

Luntbuild will check out code from url "svn://localhost/trunk/testsvn/web" to directory "<project work directory>/testsvn/web".

Branches

Source path: testsvn/web
Branch: simplified-chinese
Label: <empty>
Destination path: <empty>

Luntbuild will check out code from url "svn://localhost/branches/simplified-chinese/testsvn/web" to directory "<project work directory>/testsvn/web".

Tags

Source path: testsvn/web
Branch: <empty>
Label: v1_0
Destination path: <empty>

Luntbuild will check out code from url "svn://localhost/tags/v1_0/testsvn/web" to directory "<project work directory>/testsvn/web".

Tags and path

Source path: testsvn/web
Branch: simplified-chinese
Label: v1_0
Destination path: testsvn/web/simplified-chinese

Luntbuild will check out code from url "svn://localhost/tags/v1_0/testsvn/web" to directory "<project work directory>/testsvn/web/simplified-chinese".

Note

Branch definition is ignored here because label definition takes preference.

When Luntbuild tags a version for example "v1_0" for code checked out to directory "<project work directory>/testsvn/web", the following command will be issued: "svn copy <project work directory>/testsvn/web svn://localhost/tags/v1_0/testsvn/web"

Of course you can avoid the above url mapping, by giving "Directory for trunk" property empty value, and giving "Branch" and "Label" properties both empty values. This way, you can control where to check out the code from, and where to put checked out code to, by just using the "Source path" and "Destination path" properties (in this case, source path will only be prefixed with "repository url base" property defined at the project level).

Setting Visual Sourcesafe module information.

Source path

Specify the path in the VSS repository, for example: "testvss", or "/testvss".

Note

You should not add $ in front of this path, in order to specify the whole repository, you should just enter "/".

Label

Specify the label for the above source path. This property is optional. If left empty, latest version is assumed.

Destination path

Specify the destination directory relative to the project work directory, where the contents under the above source path should be retrieved to. This property is optional. If left empty, retrieved code will be put into directory defined by the source path, relative to the project work directory.

"Source path" represents a project path relative to the root of Sourcesafe, for example "testvss", "/testvss", or "/testvss/web", etc. Path "/" or "\" can be used to retrieve the whole contents of the repository. "Label" stands for a VSS label. VSS implements branches by creating a new shared Sourcesafe projects. So you may need to configure different modules in order to get code from different branches. If "Label" is left empty, Luntbuild will get latest code for that module from VSS. If "Destination path" is defined, contents from Sourcesafe will be retrieved to "Destination path" relative to the project work directory. Otherwise the contents will be put to "Source path" relative to project work directory.

Setting StarTeam module information.

StarTeam view

Specify a StarTeam view. This property is optional. If it is left empty, the root view of the current StarTeam project will be used.

Source path

Specify a path relative to the root of the above StarTeam view. Enter "/" to specify the root.

Label

Specify the label for the above StarTeam view. This property is optional. When left empty, latest version of the specified view is assumed.

Destination path

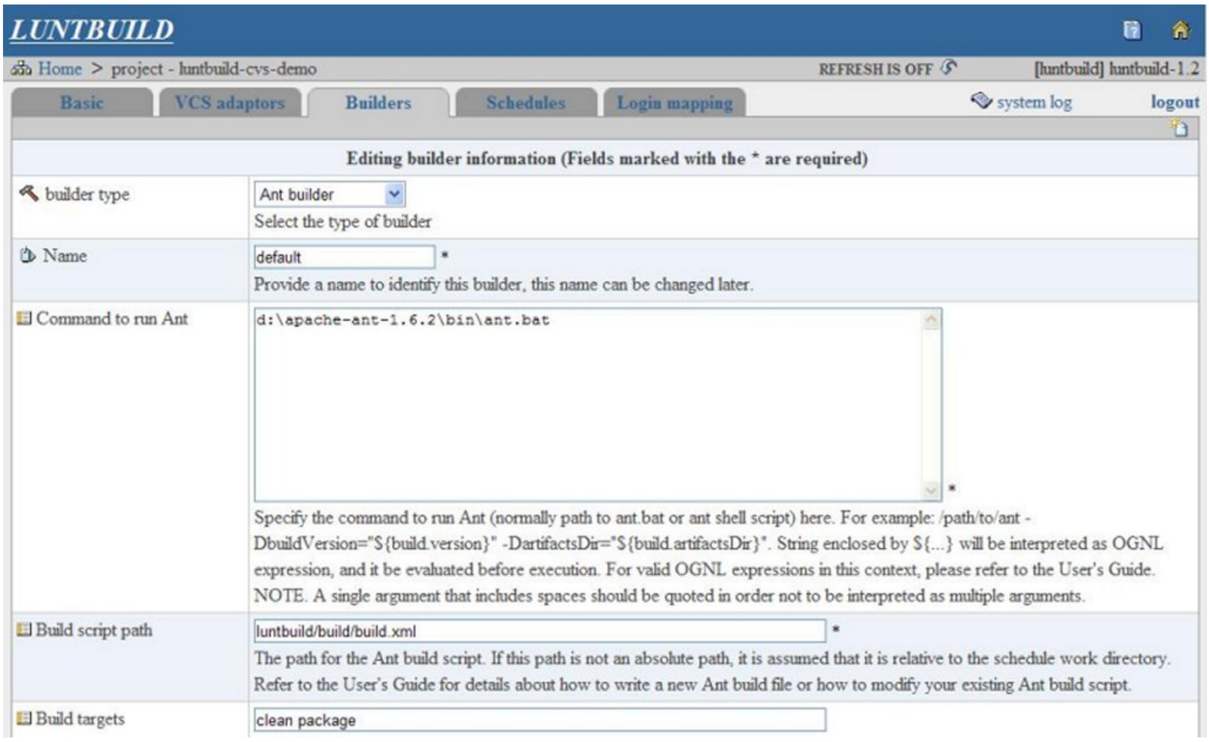
Specify the destination directory relative to the project work directory, where the contents under the above source path should be retrieved to. This property is optional. When left empty, retrieved code will be put into directory specified in source path, relative to the project work directory.

"StarTeam view" stands for a StarTeam view, and "Label" stands for a label of this StarTeam view. If "StarTeam view" is left empty, the root StarTeam view will be used. "Source path" is a path relative to the root of the chosen StarTeam view. If "Destination path" is defined, the contents from StarTeam repository will be retrieved to the "Destination path" relative to the project work directory, otherwise the contents will be put to the "Source path" relative to the project work directory.

Chapter 12. Creating the Project Builder(s)

Builder(s) are responsible for executing a build for a particular schedule of the project.

To create Builder(s), click on Builders tab, and click on New icon in the upper right corner of the tab's frame. Builders editor tab will display.



Editing builder information (Fields marked with the * are required)

builder type: Ant builder
Select the type of builder

Name: default *
Provide a name to identify this builder, this name can be changed later.

Command to run Ant:

```
d:\apache-ant-1.6.2\bin\ant.bat
```

*
Specify the command to run Ant (normally path to ant.bat or ant shell script) here. For example: /path/to/ant -DbuildVersion="${build.version}" -DartifactsDir="${build.artifactsDir}". String enclosed by ${...} will be interpreted as OGNL expression, and it be evaluated before execution. For valid OGNL expressions in this context, please refer to the User's Guide. NOTE. A single argument that includes spaces should be quoted in order not to be interpreted as multiple arguments.

Build script path: luntbuild/build/build.xml *
The path for the Ant build script. If this path is not an absolute path, it is assumed that it is relative to the schedule work directory. Refer to the User's Guide for details about how to write a new Ant build file or how to modify your existing Ant build script.

Build targets: clean package

Select the appropriate Builder type. The following Builders are available:

Ant Builder
Command Builder
Maven Builder
Maven2 Builder
Rake Builder

You can create as many builders as needed for different tasks for the given project. You will then select particular builders and/or post-builders for each schedule of this project as appropriate from the set of builders defined here.

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".

Configuring Command Builder.

Name

Provide a name to identify this builder, this name can be changed later.

Build command

Specify the build command. For example: /path/to/command.bat "${build.version}" "${build.artifactsDir}". 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 containing spaces should be quoted in order not be interpreted as multiple arguments.

Run command in directory

The directory path to run the build command in. If this path is not an absolute path, it is assumed to be relative to the project work directory.

Wait for process to finish before continuing?

This property determines whether the build will wait for the command execution to complete before continuing.

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. If left empty, the *result==0* value is assumed. Root object used for OGNL expression evaluation here is current Builder [../javadoc/com/luntsys/luntbuild/builders/Builder.html] object.

Configuring Maven Builder.

Name

Provide a name to identify this builder, this name can be changed later.

Command to run Maven

Specify command to run Maven (normally path to maven.bat or maven shell script). For example: /path/to/maven. 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

In order to use Luntbuild provided build version number in Maven, write your project.xml like the following:

```
<project>
...
<!--Use value of variable "buildVersion" as current version,
this variable is defined in Luntbuild's Maven
builder configuration page-->
<currentVersion>${buildVersion}</currentVersion>
...
</project>
```

Note

Single argument containing spaces should be quoted in order not be interpreted as multiple arguments.

Directory to run Maven in

Specify the directory to run Maven in. If this path is not an absolute path, it is assumed to be relative to the project work directory.

Goals to build

Specify the goals to build. Use space to separate different goals (goal name containing spaces should be quoted in order not to be interpreted as multiple goals). You can also use ${...} to pass OGNL variables as the goal name. For example you can use ${build.schedule.name} to use different goals 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. The logContainsLine() method uses String.matches() method [http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html#matches(java.lang.String)] to match the each (whole line) of the build log. If the expression matches only substring of the log file line the method logContainsLine() returns false!

Configuring Maven2 Builder.

Name

Provide a name to identify this builder, this name can be changed later.

Command to run Maven2

Specify command to run Maven (normally path to maven.bat or maven shell script). For example: /path/to/maven. 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

In order to use Luntbuild provided build version number in Maven2, write your pom.xml like the following:

```
<project>
...
<!--Use value of variable "buildVersion" as current version,
this variable is defined in Luntbuild's Maven
builder configuration page-->
<currentVersion>${buildVersion}</currentVersion>
...
</project>
```

Note

Single argument containing spaces should be quoted in order not be interpreted as multiple arguments.

Directory to run Maven2 in

Specify the directory to run Maven2 in. If this path is not an absolute path, it is assumed to be relative to the project work directory.

Goals to build

Specify the goals to build. Use space to separate different goals (goal name containing spaces should be quoted in order not to be interpreted as multiple goals). You can also use ${...} to pass OGNL variables as the goal name. For example you can use ${build.schedule.name} to use different goals 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("\\[INFO\\].*BUILD SUCCESSFUL.*")* value is assumed. When this expression evaluates to true, the build is considered successful. The logContainsLine() method uses

String.matches() method [http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html#matches(java.lang.String)] to match the each (whole line) of the build log. If the expression matches only substring of the log file line the method logContainsLine() returns false!

Configuring Rake Builder.

Rake [http://rake.rubyforge.org/] builder a simple Ruby build program with capabilities similar to make.

Name

Provide a name to identify this builder, this name can be changed later.

Command to run Rake

Specify the command to run Rake (normally path to rake.bat). For example: /path/to/rake. 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.

You can modify the command to add Rake command line options and properties.

Build script path

The path of the Rake 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 Rake 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 Rake 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("Command failed with status")* 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 Rake 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 Rake execution of the build returns 0, and the build log does not contain any line starting with "ERROR".

Chapter 13. Creating Build Schedule(s) for the Project

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:

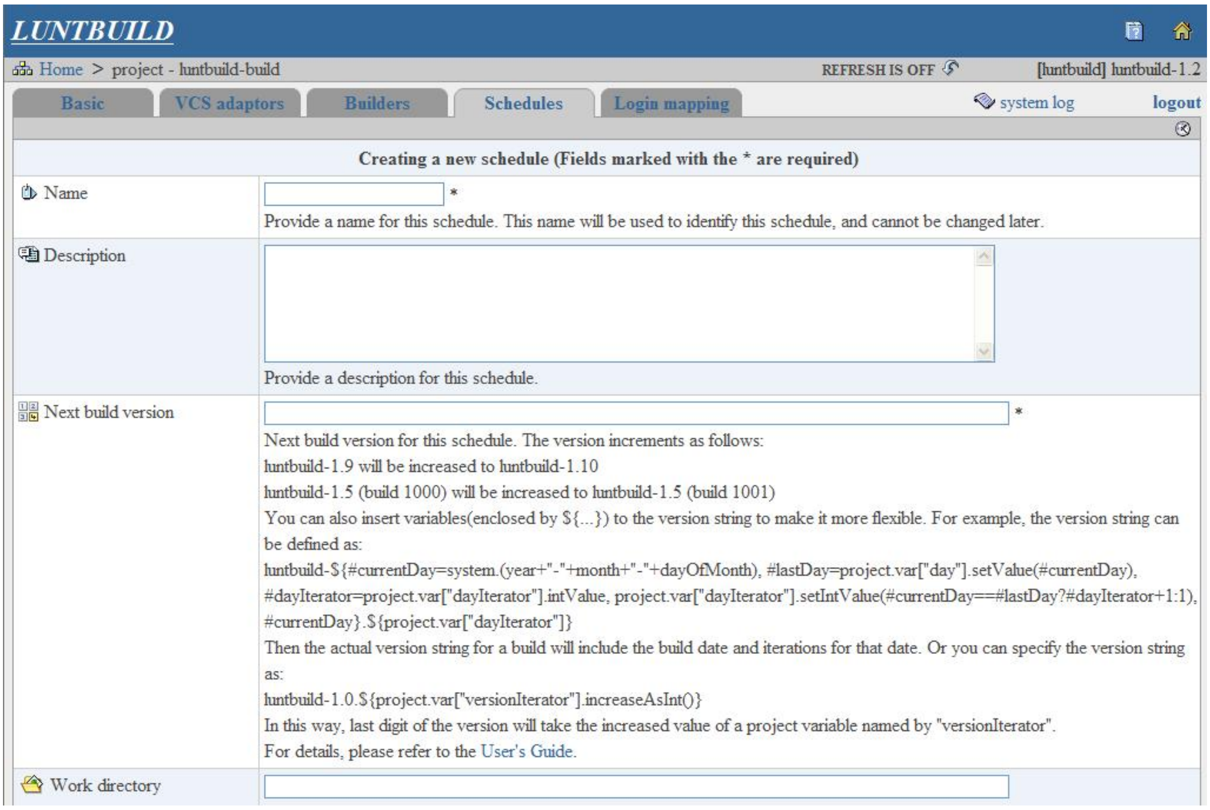
The build's work directory can be shared between multiple schedules of the same project. In this case the builds of those schedules use the same work directory, thus saving the disk space. Luntbuild guarantees that builds that share the same work directory cannot be executed at the same time. If first 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.



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

Schedule Parameters.

Name. Provide a name for this schedule. This name will be used to identify this schedule, and cannot be changed later. Keep in mind that schedule name is used as subdirectory of the project publish directory.

Description. Provide a description for this schedule.

Next build version. Specify the string for the next build version, keep in mind the name will be used as a subdirectory in the schedule's publish directory. The version string is incremented by Luntbuild as follows:

luntbuild-1.0 will be increased to luntbuild-1.1
luntbuild-1.2.9 will be increased to luntbuild-1.2.10
luntbuild-1.5(1000) will be increased to luntbuild-1.5(1001)
In general, the last number in the "Next build version" will be incremented with every build. For example "luntbuild-1.2.0" will increase to "luntbuild-1.2.3" after three builds. However, if there are OGNL expressions (encapsulated within ${...}) embedded in, the last number will not be increased automatically. Luntbuild will evaluate every embedded OGNL expression to get the actual version string for the specific build. When performing the evaluation, current Schedule [../javadoc/com/luntsys/luntbuild/db/Schedule.html] object will be used as the OGNL root object. Here are some examples of using OGNL expression to achieve various versioning strategy:

Scenario 1: *Put current date and iteration of this date as the part of the build version.*

Define "next build version" of every schedule to be:

```
foo-${#currentDay=system.(year+"-"+month+"-"+dayOfMonth), \
#lastDay=project.var["day"].setValue(#currentDay), \
#dayIterator=project.var["dayIterator"].intValue, \
project.var["dayIterator"]. setIntValue(#currentDay==#lastDay?#dayIterator+1:1), \
#currentDay}.${project.var["dayIterator"]}
```

The actual version string for a build will include the build date and iterations for that date.

Scenario 2: *"test" and "release" schedule shares and increases the same version string, while "continuous integration" schedule uses another independent version.*

Both "test" and "release" schedule set "next build version" to be:

```
foo-${project.var["majorVersionPart"]}.\
${project.var["minorVersionPart"].increaseAsInt()}
```

For "continuous integration" schedule, set "next build version" to be:

```
foo-1
```

Scenario3: *"release" schedule increases release part of the version, while "nightly" schedule increases iteration part of the version. When release part of the version changes, iteration part should be reset to 1.*

Define the following variables for the project:

fixPart=foo-1.1
releasePart=1
iterationPart=0

Define "next build version" of "nightly" schedule as:

```
${project.var["fixPart"]}.${project.var["releasePart"]}-build-${project.var["iterationPart"]
```

Define "next build version" of "release" schedule as:

```
${project.var["fixPart"]}.${project.var["iterationPart"].setValue(1), project.var["releaseP
```

This way, builds in "release" schedule will get versions like: foo-1.1.1, foo-1.1.2, foo-1.1.3, ..., and builds in "nightly" schedule will get versions like: foo-1.1.1 build 1, foo-1.1.1 build 2, foo-1.1.1 build3, ..., foo-1.1.2 build 1, foo-1.1.2 build2, ...

Note

After the evaluation of an OGNL expression, Luntbuild will substitute all "." characters in the version string with "_", and all blank characters with "-". This string is then used as the label which will be applied to source code in the Version Control System for the particular build. For example, if the build's version is "v1.0 build256", source code for this build will be labeled as "v1_0-build256".

Note

Luntbuild labels the source code based on the version number. If there are multiple projects, and/or multiple builds configured in Luntbuild, you should make sure there are no duplicate version strings in Luntbuild. For example, if you configure build1 with next build version as "v1.0", and configure build2 with next build version as "v1.5" and both of these builds contain the same module and they are using the same Version Control System, then after five builds or more, version number of build1 will have duplicate version number(s) with the early builds of build2. The best practice is to name the version number for each build with project/build prefix, such as "luntbuild-dev-0.9.2".

Work directory. Work directory for the schedule. Non-absolute path will be assumed to be relative to Luntbuild's top level work directory. If left empty, <global_work_dir>/<project_name> will be assumed, where <global_work_dir> stands for Luntbuild's top level work directory, and <project_name> stands for project name of this schedule. It is possible to use the same work directory for multiple schedules of the same project. See build work directory.

Trigger type. Select the trigger type for this schedule. Value "manual" means build of this schedule can only be triggered manually. Value "simple" can be used to configure a periodic trigger (repeated every N minutes). Value "cron" can be used to configure a cron-like trigger. Refer to *http://www.opensymphony.com/quartz/* for details about how to configure cron trigger.

Cron expression. Set the cron expression for this schedule, the format is <seconds> <minutes> <hours> <day-of-month> <month> <day-of-week>. For example *0 1 * * ?* means 1:00am every day. For details of the format, refer to *Cron triggers tutorial* [http://www.opensymphony.com/quartz/wikidocs/TutorialLesson6.html].

Repeat interval (minutes). Set the repeat interval for this schedule in minutes.

Build necessary condition. The "Build necessary condition" is optional. If left empty, the "vcsModified or dependencyNewer" value is assumed. This property is used by Luntbuild to determine, if the current build is necessary when the build strategy of the build schedule is set to "build when necessary". The "Build necessary condition" is an OGNL expression. When this expression evaluates to true, the build is considered necessary. Root object used for OGNL expression evaluation here is current Schedule [../javadoc/com/luntsys/luntbuild/db/Schedule.html] object. Here are some examples to show the format of this OGNL expression:

1. *vcsModified* - this expression will evaluate to true if the repository content of the current build changes

2. *dependencyNewer* - this expression will evaluate to true if new builds are generated in one of the dependent schedule

3. *dependencySuccessful* - this expression will evaluate to true if latest builds in all dependent schedules are successful

4. *always* - this expression will always evaluate to true to force the build

5. *never* - this expression will always evaluate to false to pause the build

6. *alwaysIfFailed* - this expression will always evaluate to true if last build has failed, and will have the value of "vcsModified or dependencyNewer" if last build is successful.

7. *project["testcvs"].vcsModified* - this expression will evaluate to true if the repository content of the "testcvs" project changes.

8. *system.execute("/path/to/command.sh") == 0* - this expression will evaluate to true if the return code of the execution of the specified command is 0.

   ### Note

   Special characters such as '\', '"', should be escaped with '\', just like in Java strings.

The above expressions can be prefixed with '!' to inverse the value, for example *!modified* will be true when there are no modifications in the repository of the current project.

The above expressions can be joined with "and", and "or". For example, the expression: *modified or execute("/path/to/command.sh")==0* will be true, if repository content of the current project changes, or execution of the specified command returns 0.

The above expressions can be prefixed with '!' to negate the value, for example "!modified" will evaluate to true when there are no modifications in the repository of current view. The above expressions can also be joined with "and", "or". For example, the expression *modified or execute("/path/to/command.sh")==0* will evaluate to true if the repository content of the current view changes, or execution of the specified command returns 0.

Please go to *http://www.ognl.org* to learn more about general grammar of an OGNL expression.

Associated builders. Select builders associated with the current schedule. They will be executed one by one in the selected order.

Associated post-builders. Select post-builders associated with the current schedule. Associated post-builders will be executed after all associated builders, if the condition indicated by "post-build strategy" is met.

Build type. Select the build type for this schedule, clean build can be more reliable, but can be slower. Incremental build can be quicker, but less reliable. We suggest that all important build schedules such as nightly or release should use clean build, and very frequent build schedules such as hourly development build, can be incremental.

Note

This setting will only take effect when the build is not triggered manually. For manual builds, this value will be shown as the default value, when the schedule is being manually build.

Post-build strategy. Select the post-build strategy for this schedule. There are following strategies:

do not post-build	Do not execute post-build script after the build.
post-build when success	Execute post-build script only when the build was successful.
post-build when failed	Execute post-build script only when the build has failed.
post-build always	Always execute post-build script after the build.

Note

This setting will only take effect when the build is not triggered manually. For manual builds, this value will be shown as the default value when the schedule is being build manually.

Label strategy. Choose the label strategy for this schedule. There are following strategies:

label successful builds	Label the repository only for the successful builds.
do not label	Do not label the repository after the build.
label always	Always label the repository after the build.

Note

If the build is not labeled when it is initially built, it will not be rebuildable later.

Note

This setting will only take effect when the build is not triggered manually. For manual builds, this value will be shown as the default value when the schedule is being build manually.

Notify strategy. Choose the notify strategy for this schedule. There are following strategies:

notify when status changed	Send notification when status of the current build changes against the last build. That is, notification will be sent when the current build succeeds and the last build failed, or the current build fails and the last build succeeded.
notify when failed	Send notification only when the build failed.
notify when success	Send notification only when the build succeeded.
do not notify	Do not send notification after the build.
notify always	Always send notification after the build.

Note

This setting will only take effect when the build is not triggered manually. For manual builds, this value will be shown as the default value when the schedule is being build manually.

Dependent schedules. Select the schedules dependent on the current schedule. If scheduleA depends on scheduleB, Luntbuild will trigger the build of scheduleB before triggering the build in scheduleA. Schedule dependency might be used to define project dependencies.

Dependency triggering strategy. Choose the dependency triggering strategy defining when the current schedule is triggered. Following trigger strategies are available:

schedules this schedule depends on
Trigger schedules the current schedule depends on. Triggering of these schedules will happen before the current schedule is triggered. For example, if the current schedule uses several components built in other schedules, you can use this strategy to make sure that all components used by this schedule are up to date.

schedules that depends on this schedule
Trigger schedules that depends on the current schedule. Triggering of these schedules will happen after the current schedule is triggered. For example, if the current schedule builds a component that is used by other schedules, you can use this strategy to make sure that all components are up to date before starting the schedules that use this component.

all dependent schedules
This is the combination of the above two strategies, that is, it triggers the schedules current schedule depends on before actually triggering current schedule, and then it triggers the schedules that depends on the current schedule after triggering the current schedule.

do not trigger any dependent schedules
Do not trigger neither the schedules the current schedule depends on, nor the schedules that depends on the current schedule.

Note

This setting will only take effect when the build is not triggered manually. For manual builds, this value will be shown as the default value when the schedule is being build manually.

Build cleanup strategy. Select the build cleanup strategy for this schedule.

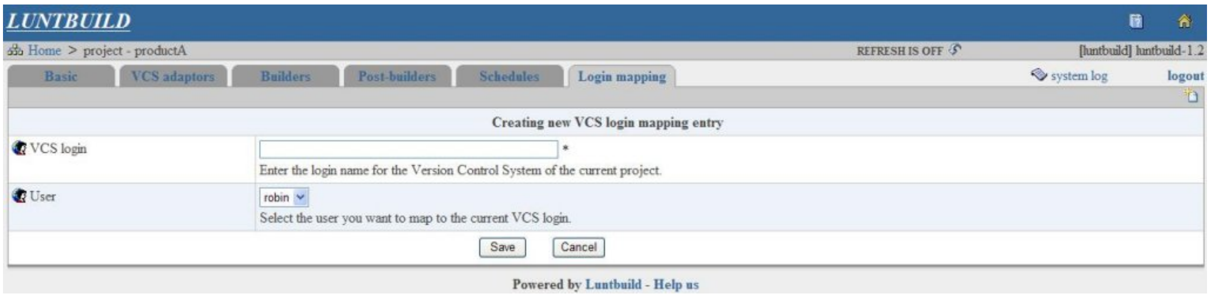
do not cleanup builds automatically: Builds can be deleted only manually.
keep builds by day: Keep builds for specified number of days.
keep builds by count: Keep specified number of builds.

Note

If a build for the given project is currently running and a new build for the same project starts (either automatically or manually), the build is placed into the pending builds queue for the given schedule. A queue of pending builds is displayed on the project's Schedule tab page.

Chapter 14. Defining Login Mapping for a Project

This page shows the mapping from VCS login to the user of the current project. When Luntbuild retrieves the list of VCS users who have recently checked code into the repository, it will use this mapping to identify corresponding users and send them notification, if requested. If the particular VCS login is not mapped, it will automatically map to the user with the same name.



Creating new VCS login mapping entry

VCS login *
Enter the login name for the Version Control System of the current project.

User robin
Select the user you want to map to the current VCS login.

VCS login   Enter the login name for the Version Control System of the current project.

User   Select the user you want to map to the current VCS login.

Chapter 15. Creating/Modifying project's Ant Builder

As introduced in chapter , you need to provide a build file, which will be used by Ant to build your project. This build file is a regular Ant build file. Before your build file is executed, Luntbuild will pre-define the following properties:

buildVersion
This property contains the current version of this build.

buildDate
This property contains the build time/date of this build.

artifactsDir
This property specifies the artifacts/publishing directory for the current build. You should generate all your final build artifacts to directory defined by property "artifactsDir", or its subdirectories. Luntbuild only stores the information about this build under this directory. You can then export your internal artifacts of this build as follows:

```
<zip basedir="stage" destfile="${artifactsDir}/${buildVersion}.zip"/>
```

These properties are not only useful when writing the Ant build file, but also useful when you run your application. For example, you can show the build version number in your application's About dialog. To achieve this, you should write these properties into a file which will be packaged in your application's distribution. You can insert the following lines inside a package target of your Ant build file:

```
<propertyfile file="stage/buildInfo.properties">
<entry key="buildVersion" value="${buildVersion}"/>
<entry key="buildDate" value="${buildDate}"/>
</propertyfile>
```

Although Luntbuild pre-defines the above properties before it runs Ant, you are encouraged to specify the default values for those properties at the beginning of your build file, such as:

```
<property name="buildVersion" value="luntbuild-1.0"/>
<property name="artifactsDir" value="distribute"/>
<property name="buildDate" value=""/>
```

Note

You do not need to redirect any generated outputs or errors from the Ant build file to your log file. Just let Ant to output them to stdout or stderr. Luntbuild will capture them and write them into the prepared log file and publish the log file on the build's artifactsDir.

Chapter 16. Creating/Modifying Project's Maven Builder

As introduced in chapter , you need to provide Maven a build goal, which will be used by Maven to build your project. Before your Maven goal is executed, Luntbuild will pre-define the following properties:

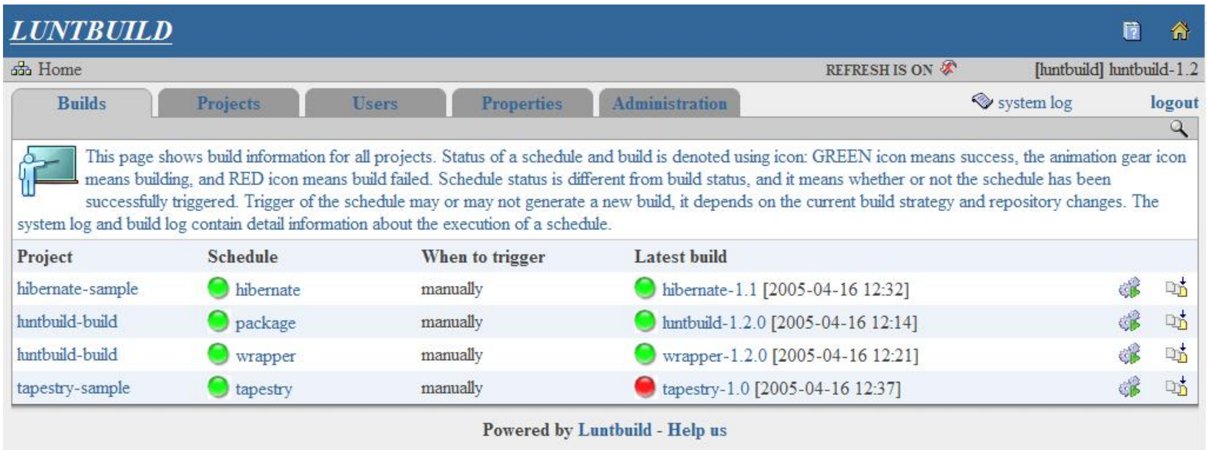
buildVersion
: This property contains the current version of this build.

buildDate
: This property contains the build time/date of this build.

artifactsDir
: This property specifies the artifacts/publishing directory for the current build. You should generate all your final build artifacts to directory defined by property "artifactsDir", or its subdirectories. Luntbuild only stores the information about this build under this directory. You can then export your internal artifacts of this build as follows:

```
<zip basedir="stage" destfile="${artifactsDir}/${buildVersion}.zip"/>
```

Chapter 17. Snapshot of all Build schedules



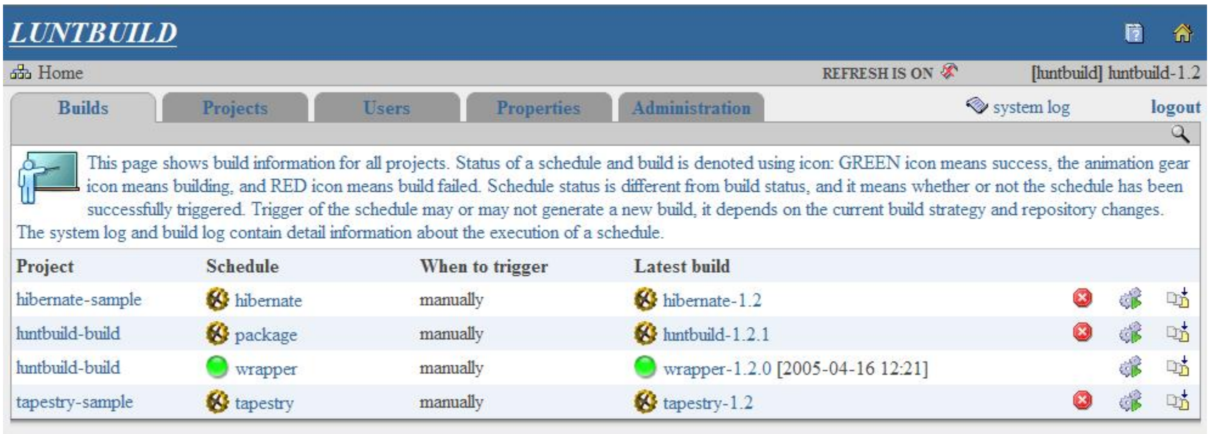
This page shows build information for all projects. Status of a schedule and build is denoted using icon: GREEN icon means success, the animation gear icon means building, and RED icon means build failed. Schedule status is different from build status, and it means whether or not the schedule has been 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 system log and build log contain detail information about the execution of a schedule.

Project	Schedule	When to trigger	Latest build
hibernate-sample	🟢 hibernate	manually	🟢 hibernate-1.1 [2005-04-16 12:32]
luntbuild-build	🟢 package	manually	🟢 luntbuild-1.2.0 [2005-04-16 12:14]
luntbuild-build	🟢 wrapper	manually	🟢 wrapper-1.2.0 [2005-04-16 12:21]
tapestry-sample	🟢 tapestry	manually	🔴 tapestry-1.0 [2005-04-16 12:37]

Powered by Luntbuild - Help us

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:



This page shows build information for all projects. Status of a schedule and build is denoted using icon: GREEN icon means success, the animation gear icon means building, and RED icon means build failed. Schedule status is different from build status, and it means 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 system log and build log contain detail information about the execution of a schedule.

Project	Schedule	When to trigger	Latest build
hibernate-sample	⚙ hibernate	manually	⚙ hibernate-1.2
luntbuild-build	⚙ package	manually	⚙ luntbuild-1.2.1
luntbuild-build	🟢 wrapper	manually	🟢 wrapper-1.2.0 [2005-04-16 12:21]
tapestry-sample	⚙ tapestry	manually	⚙ tapestry-1.2

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.

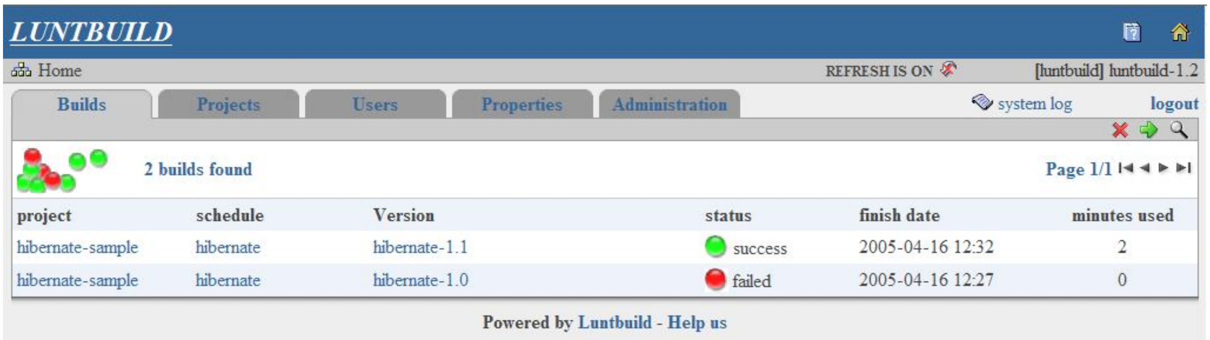
do not build

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".

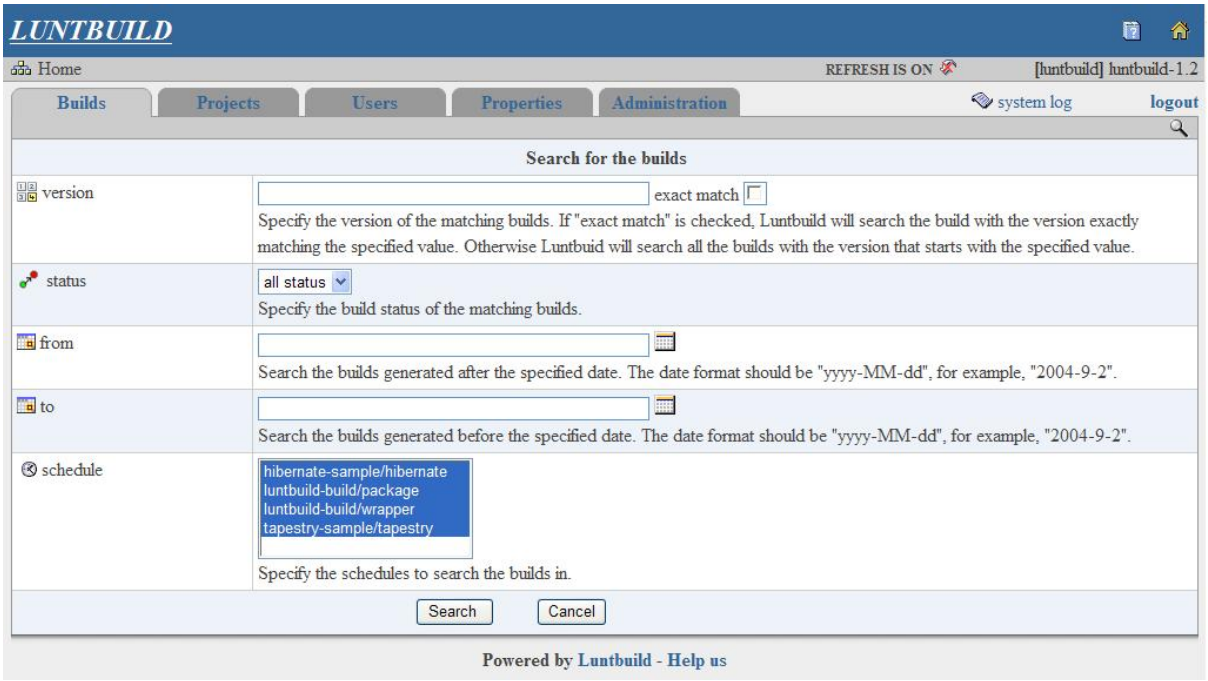
History builds for each schedule can be accessed by clicking the icon "history builds" on the right side of the schedule row. The list of all builds for the given schedule will display. You can access detailed information about a particular build by following the version hyperlink for that build. This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



project	schedule	Version	status	finish date	minutes used
hibernate-sample	hibernate	hibernate-1.1	success	2005-04-16 12:32	2
hibernate-sample	hibernate	hibernate-1.0	failed	2005-04-16 12:27	0

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



Search for the builds

version: exact match □
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 Luntbuid will search all the builds with the version that starts with the specified value.

status: all status
Specify the build status of the matching builds.

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:
hibernate-sample/hibernate
luntbuild-build/package
luntbuild-build/wrapper
tapestry-sample/tapestry
Specify the schedules to search the builds in.

Powered by Luntbuild - Help us

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 .

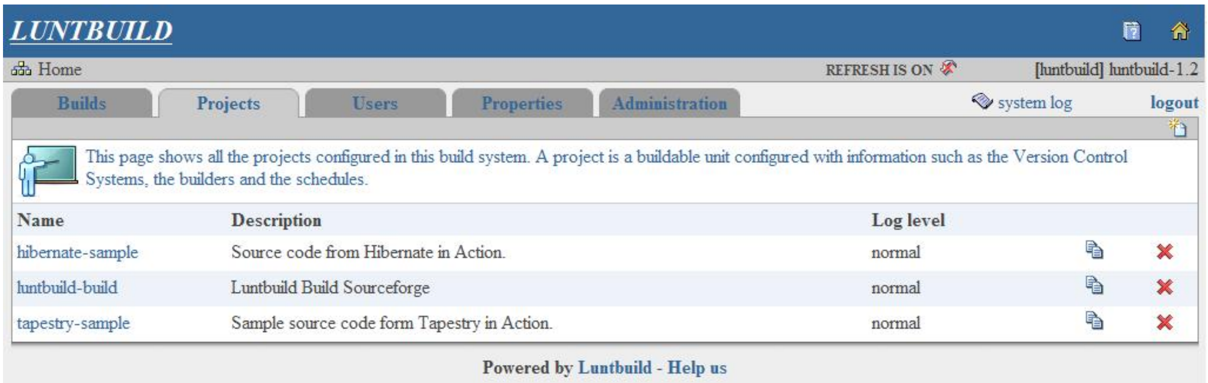
You can move (or promote) the displayed list of builds by clicking in on icon , which will display "Move builds" page. On this page you can select a "Destination schedule" to move the displayed builds to. Specify the destination schedule for these builds. The move function enables you:

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.

Chapter 18. Snapshot of all Projects



This page shows all the projects configured in this build system. A project is a buildable unit configured with information such as the Version Control Systems, the builders and the schedules.

Name	Description	Log level		
hibernate-sample	Source code from Hibernate in Action.	normal		✖
luntbuild-build	Luntbuild Build Sourceforge	normal		✖
tapestry-sample	Sample source code form Tapestry in Action.	normal		✖

This page shows all the projects currently configured in the system. A project can be created by entering the Version Control System(s) information, Builder(s) information and Schedule(s) information for a project. A project might be cloned (created a copy of project by clicking on icon ✖. This is useful for creating projects that are similar to existing projects.

Chapter 19. Using Luntbuild External API

Luntbuild provides a suite of remoting API with the following abilities:

1. Create or edit a project.
2. Trigger a build in any build schedule.
3. Configure properties of projects, as well as global properties.
4. Search builds and get a build information such as its artifacts url, etc.

By utilizing the Hessian web service protocol, this API is very easy to use. Basically two jar files needs to be included in the classpath, *hessian-3.0.8.jar* and *luntbuild-api.jar*. They can be found in the "remoting" directory. There are also some examples to demonstrate usage of the API. The remoting API Javadoc is available here. [http://luntbuild.javaforge.com/remote-api/javadoc/index.html] Following examples are available:

1. Edit Properties [http://luntbuild.javaforge.com/remote-api/samples/EditProperties.java.html] - This class gives an example of using luntbuild web service API to edit propertie of a project
2. Search Builds [http://luntbuild.javaforge.com/remote-api/samples/SearchBuilds.java.html] - This class demonstrates search builds in the system, and how to access build information
3. Trigger Build [http://luntbuild.javaforge.com/remote-api/samples/TriggerBuild.java.html] - This class gives an example of using luntbuild web service API to trigger a build
4. LuntbuildConnection [http://luntbuild.javaforge.com/remote-api/samples/LuntbuildConnection.java.html]- This class demonstrates access to project, schedule and build data. It shows how to create, edit and delete a project (supported since Luntbuild 1.3). This class is part of the Luntclipse, Eclipse plugin for Luntbuild.

The provided TriggerBuild example can be used to implement a real-time Continuous Integration, that is, whenever a checkin is made into the repository, Luntbuild can trigger immediate build. We create a sample Cvs repository to show how to do this:

1. Create a manually triggered schedule in a project you will use to implement real-time continuous integration. To get the build fast, you can configure the build to build incrementally.
2. Checkout "loginfo" file under CVSROOT directory of your cvs repository, and append a line like this:

   ```
   testcvs cmd /c d:/lunt/cvs/lunt/luntbuild/remoting/samples/trigger_build.bat
   ```

   ## Note

   Before editing, this file should be checked out first using your cvs client, just like you edit other files in your cvs repository.

   where *testcvs* should be replaced with the directory path of your Cvs repository. All checkins under this path will trigger the trigger_build.bat command. The file trigger_build.bat resides in the "remoting/samples" directory. Of course, you can copy the related files (the remoting API jar files, the TriggerBuild.class file, and the trigger_build.bat file) to any other machine as long as JDK1.4 or higher are installed. The path to trigger_build.bat need to be changed to fit your environment. Contents of trigger_build.bat should be changed to reflect proper classpaths, your Luntbuild server url, your desired project, and build schedule. On Unix platform, you can create the trigger_build.sh script easily based on the contents of the file trigger_build.bat.

3. Check in the "loginfo" file. From now on, the checkins under your configured path will trigger the trigger_build command, which will result in a build you have configured above.

Note

Any other Version Control System that can trigger an external command on checkin can use this method to implement real-time Continuous Integration.

Chapter 20. Debugging Build Problems

If you encounter a problem during the build of your project, first check the build log of the failed build. By default, Luntbuild will only write log of informational, warning, and error messages. You can edit a project information and set "Log level" property to "verbose". Luntbuild system log is written to <luntbuild install dir>/logs directory. The "text" version of the log, "luntbuild_log.txt" retains the information even on Luntbuild (servlet container) restart. The "html" version of the log, "luntbuild_log.html" contains only the information of the currently running Luntbuild application. The log includes information about modification detection, and other information not related to any specific build. You may need to look at this log to diagnose problems such as scheduled or manual build that did not execute for unknown reason etc. The most recent log file can be viewed using "system log" link at the top of every page. However, for older log files, you will need to go to the logs directory.

Chapter 21. Luntbuild Security

Luntbuild has build-in security for user authentication and authorization handling. To be independent of the servlet container (application server) Luntbuild is installed in, all configuration is done inside the Luntbuild application itself.

General security concept overview.

Only authenticated users are allowed access to Luntbuild. Luntbuild functions are assigned following basic roles:

site admin
project admin
project builder
project viewer
role description

site admin

This role is intended to represent the root user. A user with an assigned role site-admin has unrestricted access to all of the Luntbuild's functionality. Following are the tasks the site admin is allowed to execute:

user management
global property management
create projects
manage schedules
assign site-admin role to different users
assign project admin role to different users
inspect system log

project admin

This role covers functionality for project management. Following are the tasks the project admin is allowed to execute:

modify project settings
manage VCS modules
manage builds
assign users to project internal roles

project builder

This role is restricted to managing the build related tasks. Following are the tasks the project builder is allowed to execute:

manually trigger builds
assign build schedules
manage build results

project viewer

This is the most restrictive role. Following are the tasks the project viewer is allowed to execute:

view build results
view build log
download build artifacts

How to configure security for Luntbuild.

Security for Luntbuild is based on two security providers. One security provider is configured by modifying the configuration file located for example in <your app. server>/webapps/luntbuild/WEB-INFO/applicationContext.xml. For security reasons, you should consider changing the site admin password. Look for the section *inMemoryAuthenticationDAO*, and change the default site admin password to different value:

```
<bean id="inMemoryAuthenticationDAO" class="net.sf.acegisecurity.providers.dao.memory.InMemoryDaoImpl">
    <property name="userMap">
        <value>
            <!-- this is the build-in site admin user - please change the password for security reasons.
                 However, name of the user, and its role should not be changed! -->
                luntbuild=luntbuild,ROLE_SITE_ADMIN,ROLE_AUTHENTICATED
                anonymous=anonymous,ROLE_AUTHENTICATED,ROLE_ANONYMOUS
        </value>
    </property>
</bean>
```

Note

You can also change the site admin password in the installer.

The user anonymous is a user with just minimal view privileges.

All other users should be created by using Users tab page, see chapter Adding Luntbuild Users for details.

The second security provider is database based and has to be configured in the Project page to define project related roles. See chapter Creating a Project for details.

Chapter 22. Data export and import.

Data of Luntbuild (projects, schedules, etc.) can be exported or imported using "Administration" tab.

Data export.

You can specify the path of the file you want to export to, and click on the "Export" button. Data will be exported. The exported data is in XML format, and can be edited (carefully 8-), and you can import the exported data to other Luntbuild installations. If relative path is specified, the file will be located relative to the Luntbuild installation directory. We do recommend to specify the absolute path to the file to be able to locate the file easily.

Note

File specified here will be created on Luntbuild server, NOT on the machine running web browser.

Note

This operation may take a long time if there are many builds in the system. We strongly suggest to remove non-necessary builds in order to speed up this process and to speed up the import using this export file.

Data import.

You can specify the path of the file you want to import from, and click on the "Import" button. Data will be imported.

Note

This operation can only be performed on an empty database. It may takes a long time depending on the size of the file being imported.

Data migration by using export and import.

Data migration can be done easily by using export and import. For example, you currently have Luntbuild running on HSQLDB database for some time, and you want to switch to MySQL for some reason. You can do the following steps to perform the transition:

1. Start Luntbuild running on HSQLDB, and export data to a file, let's say data.xml.
2. Stop Luntbuild, edit applicationContext.xml file to switch to use MySQL.
3. Start Luntbuild, and import data.xml into the system.
4. You are done!

Appendix A. OGNL Expressions

Luntbuild Builder and Schedule can use OGNL expressions. Builder evaluates OGNL expressions of its Build Properties values. Schedule evaluates OGNL expressions of its Next build version. While Builder evaluates OGNL expressions in context of com.luntsys.luntbuild.builders.Builder class, Schedule evaluates OGNL expressions in context of com.luntsys.luntbuild.db.Schedule class. Please see javadoc [http://luntbuild.javaforge.com/docs/javadoc/index.html] for detail of classes involved in OGNL expression evaluation.

Following are some examples of available Builder OGNL expressions:

build.version	Build version
build.artifactsDir	Build artifacts directory, subdirectory of the build.publishDir.
build.publishDir	Build publish directory.
build.junitHtmlReportDir	JUnit tests Html reports directory
build.schedule.name	Build schedule name.
build.schedule.description	Build schedule description.
build.schedule.workingDir	Build schedule work directory.
build.schedule.vcsModified	Was VCS modified?
build.schedule.dependencyNewer	Is dependent schedule newer?
build.schedule.project.name	Build project name.
build.schedule.project.description	Build project description.

Here are some examples to show the format of the OGNL expressions:

1. *vcsModified* - this expression will evaluate to true if the repository content of the current build changes

2. *getProject("testcvs").isVcsModifiedSince("11/11/2005")* - this expression will evaluate to true if the "testcvs" project changed since "11/11/2005". For current project the expression would be *getProject().isVcsModifiedSince("11/11/2005").*

3. *getProject("testcvs").isVcsModified()* - this expression will evaluate to true if the "testcvs" project changed since the last build. For current project the expression would be *getProject().isVcsModified()*.

4. *ant("/path/to/command.xml", "targetA") == 0* - this expression will evaluate to true if execution of targetA of ant script file /path/to/command.xml returns *success*.

   ### Note

   Special characters such as '\', '"', should be escaped with '\', just like in Java strings. The Ant file path will be assumed relative to current project work directory, if it is not an absolute path. If the target is "" or null, the default target will be assumed.

5. *execute("/path/to/command.sh") == 0* - this expression will evaluate to true if the return code of the execution of the specified command is 0.

   ### Note

   Special characters such as '\', '"', should be escaped with '\', just like in Java strings.

The above expressions can be prefixed with '!' to inverse the value, for example *!modified* will be true when there are no modifications in the repository of the current project.

The above expressions can be joined with "and", and "or". For example, the expression: *modified or execute("/path/to/command.sh")==0* will be true, if repository content of the current project changes, or execution of the specified command returns 0.

The above expressions can be prefixed with '!' to negate the value, for example "!modified" will evaluate to true when there are no modifications in the repository of current view. The above expressions can also be joined with "and", "or". For example, the expression *modified or execute("/path/to/command.sh")==0* will evaluate to true if the repository content of the current view changes, or execution of the specified command returns 0.

Sometimes in might be handy to have your own class available for OGNL evaluation. Luntbuild supports a simple extension mechanism, so that when a JAR containing luntbuild_extension.properties is placed into Luntbuild's classpath, a class that has been defined in the property file can be accessed trough the *system* object like this:

```
${system.getExtension("MyExtension").getHelloWorld()}
```

Contents of the property file look like this:

```
luntbuild.extension.name=MyExtension
luntbuild.extension.class=org.example.MyExtension
```

The mechanism is built completely into com.luntsys.luntbuild.utility.OgnlHelper class so, that the helper classes are instantiated once and instances stored in a static Hashtable. Methods of the extension classes should be made thread safe.

Please go to *http://www.ognl.org* to learn more about general grammar of an OGNL expression.