E-Learning – VELS
User Manual

Projekt:	E-Learning – VELS
Autor(en):	Andreas Platschek, Martin Mosbeck
Reviewer:	Axel Jantsch
Version:	0.2
Datum:	June 15, 2016

Contents

Version	Autor	Date	Comment
0.1	Martin Mosbeck	14.09.2015	First shot at User Manual.
0.2	Martin Mosbeck	28.05.2016	Updates for public release.

1. Manual Overview

The following document aims to inform an operator on how to configure and run a course using the VELS VHDL E-Learning System. Therefore the implementation is only discussed in depth on a need to know basis. If you are interested in implementation details please look at the VELS Specification or the code itself.

Section 2 gives an overview of the whole system. It describes the parts out of which it is composed, how they fit together and what interfaces to change, configure, test and debug are available.

Section 3 talks about the log and error files of the VELS system.

Section 5 describes how to set up and configure the VELS System on a given server. The instructions start at how to configure the server for VELS and then tells which configurations have to be done to create a course. Lastly a configuration check-list is given and a way to test the setup before deployment described.

2. Overview of the VELS system

The VELS system is an e-learning system for students who are learning VHDL. The interaction between students and the system is solely via email. Configuration of the system is done with a configuration file (for configuration items that can not change during runtime) and a web interface (for configuration items that can change dynamically).

The VELS system can roughly be divided into 3 parts:

1. The autosub submission system.
2. The configuration and status web interface VELS_WEB.
3. The individual tasks.

The autosub submission system has the following responsibilities:

- Fetch, process and send e-mails.
- Store and manage tasks, users and submissions.
- Initiate generation of tasks and testing of submissions.
- Produce statistics.

The configuration and status web interface VELS_WEB offers the following for a course operator:

- Configure a task queue for the course.
- Whitelist, create, edit and view users and their course progress.
- Change general configuration parameters of the system.
- Modify the messages that get sent to the students at certain events.
- View statistics about e-mails and task submission success.

The responsibilities of a specific task are the following:

- Generate random task parameters.
- Generate an appropriate task description and the files that shall be provided to the student.
- Generate testbenches for task submissions.
- Test student submissions and offer meaningful error messages.

The following subsections aim to describe the VELS parts internals in order to develop an understanding what configurations are needed for the VELS system.

2.1. The autosub submission system

The autosub submission system is a daemon that is composed of multiple threads communicating over queues. This multi-threading approach is especially important for the testing process. As the test of a single task can take rather long and each student can hand in multiple (correct and incorrect) solutions. Furthermore the goal is to assure, that there still is progress, even if a test is running.

Therefore the following threads are implemented:

fetcher: The fetcher's purpose is to check the mailbox for incoming e-mails using IMAP 4 and SSL. If a new e-mail is in the mailbox the fetcher interprets the header and performs some basic checks (e.g. Is the user on the Whitelist?). It then triggers other tasks to act on the e-mail.

One further purpose of the fetcher is to archive e-mails that have already been handled. If a response to an e-mail was sent, the message-id of the original e-mail is passed back to the fetcher who then moves the original e-mail from the Inbox to the directory used for archiving of e-mails.

sender: The sender thread sends out response e-mails based using SMTP. The content of the e-mail depends on the type, the necessary information is provided by the thread that triggers the response.

generator: The generator thread generates new tasks for students. This way each student will get its own individual task. The generator calls the individual generator executables.

activator: As tasks can have an arbitrary start time, someone has to check whether or not this start time has already come and if the task shall be active. This is done by the activator thread – it periodically checks for all tasks that are not active yet, whether or not its start time has already come.

workers: The worker threads are the ones that check a submitted solution. Since testing a solution can take a considerable amount of time, multiple workers threads are created. The number of workers is configured via the configuration file (see Section 5.2), ideally this is set to something like 2x Number of CPU-Cores. The workers call the individual test executables.

dailystats: The dailystats thread is used to get a timeline of some very basic statistics. Currently the number users, sent e-mails, received e-mails and received questions over time are evaluated and plotted into a graph. This graph can e.g. be viewed in the Statistics tab of the web interface.

For storing data two databases are used. The name and location of those databases can be configured through the configuration file, in the following we will refer to them with the default name that is used, if no other name is provided in the configuration file:

semester.db: The `semester.db` database is used to store all data that is specific to one semester. This includes, users, parameterization of tasks assigned for users, statistics, etc. The `semester.db` default location is in the root autosub directory `"autosub/src"`.

course.db: The `course.db` database consists of configuration data that will very likely be reused in multiple semesters. This separation makes it very easy to start a new semester: just backup `semester.db` to some safe location and remove it in the original location (a new empty one will be made automatically upon starting the autosub daemon).Then use the VELS web interface to perform the small modifcations(year/semester,deadlines, etc.) needed in `course.db`. The `course.db` default location is in the root autosub directory `"autosub/src"`.

In addition there is one important directory that is used by autosub to store data in, the directory: `autosub/src/users`. In this directory a unique directory for every user is created, and this directory the description of each individual received task and submissions for the individual tasks are stored. Every Task directory has the form `"autosub/src/users/<user_id>/Task<nr>/"`. Submissions are stored in separate folders which are named `"Submission<nr>_<date>_<time>/"`. Task description files are stored in the folder named `"desc/"`.

2.2. The web inteface VELS_WEB

The configuration and status web interface VELS_WEB was designed as an easy interface for a course operator to view and change course parameters and monitor the course progress. It was only designed for internal operator use! Therefore it should only reachable via the internal institute network using a web browser.

The menu items of the VELS_WEB are the following:

Start: The start page.

Users: View of the registered students, allows to change or a student via the "Edit" button, delete a student via the "Delete" button and show the progress of a student in the course via the "View" button. The table is sortable by column by clicking on the head of the column.

Tasks: The task configuration, will be discussed in Section 5.8.

Whitelist: Change whitelisting for students, will be discussed in Section 5.7.

General Config: Change dynamic configuration items of VELS, will be discussed in Section 5.6.

Special Messages: View and change messages the student gets sent at special events (e.g. welcome, not registered, usage, etc.).

Statistics: View statistics about sent and received emails and task success of the students.

User Tasks: View mappings from students to individual tasks.

2.3. The tasks structure

Every task is located in an individual folder in `"autosub/tasks/implementation"`. Every task is composed of the following elements:

Minimal	• **Generator executable:** Generates random parameters for the task, creates entity and behavioral vhdl files and description test and pdf file for the student. Stores the individual task parameters in the *UserTasks* database table. • **Tester executable:** Given the individual task parameters, tries to compile the student's solution. Generates an individual test bench for the student's solution. Tests the solution and creates feedback text and files for the student. • **description.txt:** Contains a textual task description. The text in description.txt is sent to the user as the body of the e-mail that contains a new task description.

Optional	• **Scripts:** Scripts that are called from the executables in order to aid the generation or test process • **Templates:** These files have to be filled with parameters and can be used to generate entity declarations, test benches or description texts and files for the students. • **Static:** These files are static for the task, therefore the same for every student. • **Exam:** Files that are needed for the VELS exam mode.

A sequence for the task generation can go as follows:

1. The generator executable is called by the autosub generator thread.
2. The generator executable calls a task generation script `"scripts/generateTask.py"`.
3. The task generation script generates random task parameters and returns them, fills a LaTeX description template and vhdl templates and stores the results in the directory `"tmp/"`.
4. The generator executable generates the task description pdf file from the filled template and copies it, all filled vhdl files and static vhdl files to the users task description path. `"autosub/users/<user_id>/Task<task_nr>/desc/"`.
5. The generator executable stores the path to files that need to be attached to the task email for the student in the `semester.db UserTasks` database table.

The usual sequence for task submission testing is the following:

1. The tester executable is called by a autosub worker thread.
2. The tester executable calls a testbench generation script `"scripts/generateTestBench.py"`.
3. The tesbench generation script creates a testbench with test vectors and returns it.
4. The tester executable stores the testbench and copies needed files from the task. description in the taks directory `"austosub/users/<user_id>/Task<task_nr>"`

5. The tester executable checks if the appropiate submission files are present. These files are copied to the user- task- folder by the fetcher thread.
6. The tester executable analyzes files generated by the task generation, if errors occur the process stops and sends an email to student and system administrator.
7. The tester executable analyzes the student submission files, if errors occur they are written to a `error_msg` file and the process stops.
8. The tester elaborates and runs the testbench, if the tests fail meaningful messages are written to a `error_msg` file and the process stops.
9. The tester returns success.

3. Logging and Error Detection of the VELS System

The autosub system is implemented as a daemon process. As such it does not directly print any information to the `stdout` (i.e. the screen), instead all messages for the administrator are stored in log files. All log and error files are stored in the root autosub directory `"autosub/src"`. Due to the modular nature of autosub, not everything can be included into one big fat log file (although most of it is), but some things are sent to separate files. In essence autosub uses 3 files that will tell the administrator about everything that is going on:

autosub.log This is the main log file that is used by the actual autosub daemon software to log everything. The format in this log file is as follows:

```
<date> <time> [<logger name>] <loglevel> <logmessage>
```

Here is a small section of `autosub.log` taken from startup – first the daemon checks whether a directory `users` used to store the users submissions is existing, then it checks if necessary tables in the database exist. After those checks have been done further threads (as described in Section 2.1) are started, and announce themselves.

```
2015-09-17 22:23:40,629 [autosub.py  ] WARNING: Directory already exists: users
2015-09-17 22:23:40,630 [autosub.py  ] DEBUG: table SpecialMessages does not exist
2015-09-17 22:23:43,679 [autosub.py  ] DEBUG: table TaskConfiguration does not exist
2015-09-17 22:23:43,955 [autosub.py  ] DEBUG: table GeneralConfig does not exist
2015-09-17 22:23:45,878 [sender      ] INFO: Starting Mail Sender Thread!
2015-09-17 22:23:45,884 [fetcher     ] INFO: Starting Mail Fetcher Thread!
2015-09-17 22:23:45,884 [fetcher     ] DEBUG: Imapserver: 'mail.intern.tuwien.ac.at'
2015-09-17 22:23:45,884 [generator   ] INFO: Task Generator thread started
2015-09-17 22:23:45,890 [activator   ] INFO: Starting activator
2015-09-17 22:23:45,890 [Main        ] INFO: Used config-file: testzid.cfg
2015-09-17 22:23:45,891 [Worker1     ] INFO: Starting Worker1
2015-09-17 22:23:45,892 [Main        ] INFO: All threads started successfully
2015-09-17 22:23:45,892 [Worker1     ] INFO: Worker1: waiting for a new job.
2015-09-17 22:23:45,893 [Worker2     ] INFO: Starting Worker2
```

As you can see from theses examples, a lot of information is available, with exact timestamps and a way to know which thread the message is coming from. In addition the debug level gives a sense of how important (or even critical) the message is.

The log-level is currently hard-coded into the logger thread (`logger.py`) and can only be changed in the code. Log-file size can get several MB in size we do currently see no need to set the log-level to something other than DEBUG and gather all information that can be very valuable in case something goes wrong.

autosub.stderr and autosub.stdout As already explained in Section 2.1, the autosub daemon calls generator and tester scripts for the individual tasks. The output of these tasks can not easily be piped into the autosub logging thread. If it could, there would also be no way to check if the script actually do it, and the developer of a task could instead again just print to `stdin` and `stderr`. Therefore the logging is simply done by piping the output of called generator and tester scripts into these files.

This means, that all messages printed by the generator and tester executables are piped into those two files and can be used to find out if the generator and / or tester executable (or the programs they call) cause any problems.

While these three files give all information needed to debug problems, autosub tries to give information on severe problems on all available channels. An example would be, if no task is configured, the email sent to the student will contain the message that something went wrong as well – although the error can be seen in the `autosub.log` file as well.

4. Pre-Requisites

In the following, the installation of pre-requisites for autosub is documented for the two current debian versions. Wheezy is currently the old-stable which was used for development of autosub, and jessie is the current stable which has been used to test autosub as well.

4.1. Debian 7 (Wheezy)

The probably easiest step is to install standard debian packages:

```
apt-get install python3 python3-mock python3-pip python3-nose zip flip git
apt-get install gnat-4.6-base libgnat-4.6 texlive-latex-extra pgf graphviz
apt-get build-dep python-matplotlib
```

Unfortunately there is a small bug in one of the latex packages that we use, therefore you need to apply a patch to that package:

```
cd /usr/share/texlive/texmf-dist/tex/generic/pst-circ
patch < /path/to/repo/autosub_internal/patch/pst-circ.tex.patch
```

In case the pdfs you get are not beautiful as you are used to, it might have happended that an update was applied over that patch and you need to re-apply it (sorry for that little mess).

Next we need to upgrade easy_install for python3 to a newer version – this is the pre-requisite for the installation of the matplotlib for python3.

```
easy_install3 -U distribute
```

And finally we may now install some libraries that are not (yet) available in the debian package system via using pip:

```
pip-3.2 install graphviz
pip-3.2 install bitstring
pip-3.2 install matplotlib
```

Last, we install ghdl, which is used as the VHDL Simulation tool. You may download the debian package from sourceforge [1] and install it using dpkg:

```
dpkg -i Downloads/ghdl_0.31-2wheezy1_amd64.deb
```

All the dependencies of that package have already been resolved by the above list of packages installed with apt.

At this point you may run the test-suite, as explained in Section 5.11. If those tests run through without complications, everything you need to run the VELS E-Learning system has been installed successfully.

4.2. Debian 8 (Jessie)

For jessie everything is even easier as more libraries are available for python3 in the apt package pool now:

```
apt-get install python3 python3-mock zip flip graphviz python3-pip
apt-get install python3-matplotlib python3-nose gnat-4.9-base libgnat-4.9
apt-get install zlib1g-dev texlive-latex-extra pgf git
```

Unfortunately there is a small bug in one of the latex packages that we use, therefore you need to apply a patch to that package:

```
cd /usr/share/texlive/texmf-dist/tex/generic/pst-circ
patch < /path/to/repo/autosub_internal/patch/pst-circ.tex.patch
```

In case the pdfs you get are not beautiful as you are used to, it might have happended that an update was applied over that patch and you need to re-apply it (sorry for that little mess).

So now, only two more packages that need to be installed using pip:

```
pip3 install bitstring
pip3 install graphviz
```

[1] http://sourceforge.net/projects/ghdl-updates/files/Builds/ghdl-0.31/Debian/

An finally ghdl is installed, this time a bit newer version[2] as there are no .deb packages for jessie of the older one (and vice-versa).

```
dpkg -i ghdl_0.33-1jessie1_amd64.deb
```

At this point you may run the test-suite, as explained in Section 5.11. If those tests run through without complications, everything you need to run the VELS E-Learning system has been installed successfully.

[2] http://sourceforge.net/projects/ghdl-updates/files/Builds/ghdl-0.33/debian/

5. Setting up the VELS system

5.1. VELS server setup

The first thing to do when you want to install the autosub submission system and the VELS web interface is to clone the git repositories:

```
git clone https://github.com/andipla/autosub.git
```

for the autosub project, as well as the VELS specific part from our local server:

```
git clone vels@vels.ict.tuwien.ac.at:/shared/autosub_internal
```

In the following if not explicitly stated the autosub (not autosub_internal) repository will be used. The difference between those two is, that the first one is located on github containing all the already released source and documentation, while the second one contains source and documents that are specific to the ICT VELS project and need to be generalized before they can be released into the wild.

5.2. Autosub submission config file

The autosub config file is used to configure the connection to the mail server and set initial configuration parameters for the system. It is usually located in the autosub root folder `"autosub/src/"` and is used when starting the autosub system. It is divided in groups and entries. An example is given in the next subsection 5.3.

The configuration file should include the following entries:

Group	Field	Description
imapserver	servername	Hostname.domain of the imap server.
imapserver	username	Username used to login to the imap server.
imapserver	password	Password used to login to the imap server.
imapserver	e-mail	E-Mail address at the imap server.
smtpserver	servername	Hostname.domain of the smtp server.
general	num_workers	Number of Worker threads. This influences how many tests can conducted in parallel.
general	queue_size	Size of the thread communication queues.
general	poll_period	Period in seconds at which the mailbox is checked.
general	semesterdb	The name and path of the database that shall be used for data specific to this semester. (default: semester.db)
general	coursedb	The name and path of the database that shall be used for data specific to this course. (default: course.db)
general	logfile	The name and path of the autosub logfile. (default: autosub.log)
general	specialpath	Not used at the moment. Will be used in the future
general	course_name	The name of course.
challenge	num_tasks	Number of tasks for the course.
challenge	mode	The mode in which to run (exam or normal).

For the SMTP connection it is assumed that the e-mail, user and password are the same as for IMAP.

5.3. Example Configuration File

```
[imapserver]
servername: imap.gmail.com
serverport: 587
username: submission@gmail.com
password: mysupersecurepassword
email: submission@gmail.com

[smtpserver]
servername: smtp.gmail.com
username: submission@gmail.com
password: mysupersecurepassword
email: submission@gmail.com

[general]
num_workers: 3
queue_size: 200
poll_period: 60
coursedb: /tmp/course.db
semesterdb: /tmp/semester.db
logfile: /home/andi/working_git/autosub/src/autosub.log

[challenge]
num_tasks: 6
```

Listing 1: example.cfg

5.4. Starting the Daemon

Autosub is implemented as a daemon process, that means all messages provided are written to files (see Section 3 for details on those files) – nothing is written to the console. The daemon is started using a shell-script located in `autosub/src/`:

```
./autosub.sh start
```

This starts the daemon using the default configuration file named `default.cfg`. If you wan to use your own configuration file, you have to pass the files name to the scrip when starting the daemon:

```
./autosub.sh start myconfig.cfg
```

To stop the daemon just run the command:

```
./autosub.sh stop
```

5.5. Setting up the VELS_WEB Configuration Interface

To use the VELS_WEB Configuration Interface it has first to be installed and configured using these steps:

Change to the VELS_WEB directory.

```
python3 installer.py <pathtoautosub> <pathtoconfigfile>
```

Use the same configfile you used for starting the autosub daemon! This step will also download web2py to the user's home folder and set needed symbolic links to connect VELS_WEB to the autosub system.

To use https you have to use a SSL key. The system expects the keyfiles (server.crt , server.csr , server.key) to be in the web2py directory. To generate keys and place them run the following (this can be skipped if you already have keyfiles you can use!):

```
./genkeys.sh
```

To start the VELS_WEB daemon at port <port> and with password <password> run:

```
./daemon.sh <port> <password>
```

5.6. General Configuration

Configuration items that can be changed dynamically are changeable in VELS_WEB -> General Config. These configurable items are:

- **Number of tasks:** Can also be specified in the config file.
- **Registration Deadline:** Users who are try to register after this deadline will get an error e-mail.
- **Archive Directory:** Directory in which processed e-mails are moved, this directory has to be present on the IMAP server!
- **Administration Email:** Email addresses which get question and system error emails.

5.7. Whitelisting

Students that participate in the course have to be whitelisted in the VELS system. If the student tries to write an e-mail to the system from an e-mail address that is not on the Whitelist, an E-Mail with an error message is sent to him.

Whitelisting can be done in VELS_WEB under the tab *Whitelist*. Email addresses can be added one at a time or multiple at a time (mass subscription). Removal of a single e-mail address can also be done in the VELS_WEB.

5.8. Configuring the Tasks

Existing tasks can be assembled into a task queue. This configuration is done in VELS_WEB. Each task in the queue has to be created with the following properties:

- **TaskStart:** The start datetime for the task. The task will automatically be set to active if this datetime is reached. Users waiting for a task to become active will automatically receive an e-mail with the task description for that task.
- **TaskDeadline:** The end datetime for the task. Submissions for a tasks after this datetime will be rejected.
- **PathToTask:** The root path of the task, by default located in `"autosub/src/tasks/implementation/VHDL"`, but that may be any arbitrary path.
- **GeneratorExecutable:** The path to the generator executable relative to the root path, by default the generator is located in the root path and named `generator.sh`.
- **TestExecutable:** The path to the tester executable relative to the root path, by default the generator is located in the root path and named `test.sh`.
- **Score:** The score a student gets for successful completion of the task. The scores for all completed tasks are added and can therefore also be used for grading.
- **TaskOperator:** The e-mail of the operator of the task. Currently this email is not used by the autosub system.
- **TaskActive:** The state of the task, for inactive tasks the generator won't be called.
- **Position:** The position of the task in the task queue. Do not change this during operation of your course!

5.9. Configuration Checklist

1. Installed all needed libraries and tools for autosub, the tasks and VELS_WEB.
2. Configured the email server, including an email archive folder.
3. Created a config file for autosub.
4. Started the autosub system via `autosub.sh start <configfile>`.
5. Started VELS_WEB using the daemon
6. Configured all parameters in General Config in VELS_WEB.
7. Configured all Tasks in VELS_WEB.
8. Whitelisted all students in VELS_WEB.

If you forget one of this steps or mis-configure, autosub tries to generate a meaningful message, still its nicer to get everything running without being yelled at.

5.10. The Exam Mode

In Exam Mode the students additionally get sent a minimal testbench to test their design. To enable test mode, change the challenge-mode to exam in VELS_WEB -> General Config for an existing course (databases exist) and in the config file for a new course (databases don't exist).

The remaining configuration is similar to configuration for normal mode.

5.11. Testing VELS

VELS has a testing suite located in `"autosub/src/tests"`. It consists of unit and doctests, testing both the autosub system and the tasks itself. With this test suite you can test if everything is set up the right way before starting a course.To run the test suite issue the following command from the `"autosub/src/"` directory:

```
nosetests3 --with-doctest --doctest-extension=txt --nologcapture -v
```

This will run all doctest as well as unittest test cases.

If you also want the code coverage of the test-suite then run it as follows:

```
nosetests3 --with-doctest --doctest-extension=txt --nologcapture -v --with-coverage
```

While the above commands run all of the tests, it is also possible to include or exclude only specific tests. In example it is possible to only execute the load test:

```
nosetests3 --nologcapture -s -v  tests/load_test.py:Test_LoadTest
```

or it is possible to exclude the load test (which makes sense, as that one takes a considerable amount of time):

```
nosetests3 --nologcapture -s -v --ignore-files=load_test.py
```

The following is tested by the testsuite:

- Connecting to the databases.
- Logging a message.
- Sending an email.
- Functionality of the activator thread.
- Functionality of the generator thread.
- Functionality of the sender thread.
- Functionality of the fetcher thread.
- Functionality of the common used functions.
- Behavior under high load (load test).
- Creation off all description files for every task.
- Compilation of all generated VHDL files with ghdl.
- Tester functionalities for given right submissions with ghdl.