DOMjudge Administrator's Manual

by the DOMjudge team Fri, 29 Dec 2017 22:00:20 +0100

This document provides information about DOMjudge installation, configuration and operation for the DOMjudge administrator. A separate manual is available for teams and for jury members. Document version: 9270426

Contents

1 DOMjudge overview

DOMjudge is a system for running programming contests, like the ACM ICPC regional and world championship programming contests.

This means that teams are on-site and have a fixed time period (mostly 5 hours) and one computer to solve a number of problems (mostly 8-11). Problems are solved by writing a program in one of the allowed languages, that reads input according to the problem input specification and writes the correct, corresponding output.

The judging is done by submitting the source code of the solution to the jury. There the jury system automatically compiles and runs the program and compares the program output with the expected output.

This software can be used to handle the submission and judging during such contests. It also handles feedback to the teams and communication on problems (clarification requests). It has web interfaces for the jury, the teams (their submissions and clarification requests) and the public (scoreboard).

1.1 Features

A global overview of the features that DOMjudge provides:

- Automatic judging with distributed (scalable) judge hosts
- Web interface for portability and simplicity
- Modular system for plugging in languages/compilers and validators
- Detailed jury information (submissions, judgings, diffs) and options (rejudge, clarifications, resubmit)
- Designed with security in mind

DOMjudge has been used in many live contests (see <https://www.domjudge.org/intro> for an overview) and is Open Source, Free Software.

1.2 Requirements

This is a (rough) list of the requirements for DOMjudge.

- At least one machine running Linux, with (sudo) root access
- Apache web server with PHP 5.3.3 or newer and PHP-command line interface
- MySQL or MariaDB database server version 5.3.3 or newer
- Compilers for the languages you want to support

A 2.2 (detailed list of requirements) is contained in the 3 (Installation and Configuration) chapter.

1.3 Copyright and licencing

DOMjudge is developed by Jaap Eldering, Nicky Gerritsen, Keith Johnson, Thijs Kinkhorst and Tobias Werth; Peter van de Werken has retired as developer. Many other people have contributed (apologies for any oversights): Michael Baer, Jeroen Bransen, Stijn van Drongelen, Rob Franken, Marc Furon, Jacob Kleerekoper, Ruud Koot, Jan Kuipers, Richard Lobb, Alex Muntada, Dominik Paulus, Bert Peters, Jeroen Schot, Matt Steele, Shuhei Takahashi, Hoai-Thu Vuong, and Jeroen van Wolffelaar. Some code has been ported from the ETH Zurich fork by Christoph Krautz, Thomas Rast et al.

This software is partly based on code by other people. These acknowledgements are made in the respective files, but we would like to name them here too (with non-GPL licences listed where applicable):

- dash (i386) is included, statically compiled from the Debian dash sources (copyright by various people under the BSD licence and a part under the GNU GPL version 2. See `COPYING.BSD` `doc/dash.copyright` for more details). Sources can be downloaded from: <https://www.domjudge.org/sources/> .
- basename.h is a modified version from the GNU libiberty library (copyright Free Software Foundation).
- lib.database.php by Jeroen van Wolffelaar et al.
- runguard.c was originally based on timeout from The Coroner's Toolkit by Wietse Venema.
- sorttable.js, by Stuart Langridge, licenced under the MIT licence, see `COPYING.MIT`. It was downloaded from <http://www.kryogenix.org/code/browser/sorttable/> .
- jscolor.js by Jan Odvarko, licenced under the GNU LGPL. It was obtained at <http://jscolor.com> .
- tabber.js by Patrick Fitzgerald, licenced under the MIT licence, see `COPYING.MIT`. It was downloaded from

  <http://www.barelyfitz.com/projects/tabber/> .
- Ace code editor by Ajax.org B.V., licenced under the BSD licence, see `COPYING.BSD`. It was downloaded from

  <https://github.com/ajaxorg/ace-builds> , the `src-min-noconflict` version.
- Flot JavaScript library by IOLA and Ole Laursen, licenced under the MIT licence, see `COPYING.MIT`. It was downloaded from <http://www.flotcharts.org/> .
- jQuery JavaScript library by the jQuery Foundation, licenced under the MIT licence, see `COPYING.MIT`. It was downloaded from <http://jquery.com/> .
- jQuery TokenInput by James Smith, dual licenced under the GPL and MIT licences, see `COPYING` and `COPYING.MIT`. It was downloaded from

  <https://github.com/loopj/jquery-tokeninput> .
- JavaScript Cookie by Klaus Hartl and Fagner Brack, licenced under the MIT licence, see `COPYING.MIT`. It was downloaded from <https://github.com/js-cookie/js-cookie> .

- The Spyc PHP YAML parser by Chris Wanstrath and Vlad Andersen, licenced under the MIT licence, see `COPYING.MIT`. It was downloaded from <https://github.com/mustangostang/spyc/> .
- The default compare script was included from the Kattis problemtools package, and licenced under the MIT licence, see `COPYING.MIT`. It was downloaded from

  <https://github.com/Kattis/problemtools/tree/master/support/default_validator> .
- The DOMjudge logo is based on the NKP 2004 logo made by Erik van Sebille.
- Several icons have been taken from the phpMyAdmin project.
- Several M4 autoconf macros from the Autoconf archive by various people are included under `m4/`. These are licenced under all-permissive and GPL3+ licences; see the respective files for details.

1.3.1 About the name and logo

The name of this judging system is inspired by a very important and well known landmark in the city of Utrecht: the dome tower, called the 'Dom' in Dutch. The logo of the 2004 Dutch Programming Championships (for which this system was originally developed) depicts a representation of the Dom in zeros and ones. We based the name and logo of DOMjudge on that.

We would like to thank Erik van Sebille, the original creator of the logo. The logo is under a GPL licence, although Erik first suggested a "free as in beer" licence first: you're allowed to use it, but you owe Erik a free beer in case might you encounter him.

1.4 Contact

The DOMjudge homepage can be found at: https://www.domjudge.org/

We have a low volume mailing list for announcements of new releases.

The authors can be reached through the development mailing list: domjudge-devel@domjudge.org . You need to be subscribed before you can post. See the list information page for subscription and more details.

Some developers and users of DOMjudge linger on the IRC channel dedicated to DOMjudge on the Freenode network: server `irc.freenode.net`, channel `#domjudge`. Feel free to drop by with your questions and comments, but note that it may sometimes take a bit longer than a few minutes to get a response.

2 Contest planning

2.1 Contest hardware

DOMjudge discerns the following kinds of hosts:

Team computer

Workstation for a team, where they develop their solutions and from which they submit them to the jury system. The only part of DOMjudge that runs here is the optional command line submit client; all other interaction by teams is done with a browser via the web interface.

DOMjudge server

A host that receives the submissions, runs the database and serves the web pages. This host will run the Apache webserver and MySQL database. Also called *domserver* for brevity.

Judgehosts

A number of hosts, at least one, that will retrieve submitted solutions from the DOMjudge server, compile and run them and send the results back to the server. They will run the `judgedaemon` from DOMjudge.

Jury / admin workstations

The jury members (persons) that want to monitor the contest need just any workstation with a web browser to access the web interface. No DOMjudge software runs on these machines.

One (virtual) machine is required to run the DOMserver. The minimum amount of judgehosts is also one, but preferably more: depending on configured timeouts, judging one solution can tie up a judgehost for several minutes, and if there's a problem with one judgehost it can be resolved while judging continues on the others.

As a rule of thumb, we recommend one judgehost per 20 teams.

However, overprovisioning does not hurt: DOMjudge scales easily in the number of judgehosts, so if hardware is available, by all means use it. But running a contest with fewer machines will equally work well, only the waiting time for teams to receive an answer may increase.

Each judgehost should be a dedicated (virtual) machine that performs no other tasks. For example, although running a judgehost on the same machine as the domserver is possible, it's not recommended except for testing purposes. Judgehosts should also not double as local workstations for jury members. Having all judgehosts be of uniform hardware configuration helps in creating a fair, reproducible setup; in the ideal case they are run on the same type of machines that the teams use.

DOMjudge supports running multiple judgedaemons in parallel on a single judgehost machine. This might be useful on multi-core machines. Note that although each judgedaemon process can be bound to one single CPU core (using Linux cgroups), shared use of other resources such as disk I/O might still have a minor effect on run times. For more details on using this, see the section 3.18 (Installation: optional features).

2.2 Requirements

2.2.1 System requirements

The requirements for the deployment of DOMjudge are:

- Computers for the domserver and judgehosts must run Linux (or the domserver possibly a Unix variant). This software has been developed mostly under Debian GNU/Linux, and the manual adds some specific hints for that, which also apply to Debian derivative distributions like Ubuntu. DOMjudge has also been tested under RedHat-like Linux distributions. We try to adhere to POSIX standards, but especially the judgehost security solution is Linux-specific.
- (Local) root access on the domserver and judgehosts for configuring sudo, installing some files with restricted permissions and for (un)mounting the proc file system. See 7.3 (Security: root privileges) for more details.
- A TCP/IP network which connects all DOMjudge and team computers. Extra network security which restricts internet access and access to other services (ssh, mail, talk, etc..) is advisable, but not provided by this software, see 7.5 (Security: external security) for more details. All network-based interactions are done over HTTP or HTTPS (tcp port 80 or 443):
  - HTTP traffic from teams, the public and jury to the web server.
  - The judgehosts connect to the DOMjudge API on the webserver over HTTP(S).
  - The 'submit' command line client connects to API on the web server also via HTTP(S).

  When using the `IP_ADDRESS` authentication scheme, then each team computer needs to have a unique IP address from the view of the DOMjudge server, see 4.3 (Contest setup: team authentication) for more details.

2.2.2 Software requirements

The following software is required for running DOMjudge.

- For every supported programming language a compiler is needed; preferably one that can generate statically linked stand-alone executables.
- Apache web server with support for PHP >= 5.4 and the mysqli, GD, curl and json extensions for PHP. PHP needs to be running as an Apache module (the most common configuration); a (fast)CGI setup is not currently supported. We also recommend the posix extension for extra debugging information. A configuration file for the Nginx webserver is also included, and may be used instead of Apache. However, this setup is less well tested and documented.
- MySQL or MariaDB >= 5.5.3 database and client software.
- PHP >= 5.4 command line interface and the curl, json, mcrypt, and gmp extensions.
- A POSIX compliant shell in `/bin/sh` (e.g. bash or ash).
- A statically compiled POSIX shell, located in `lib/judge/sh-static` (dash is included for Linux i386/amd64).
- libcgroup , to enable support for Linux cgroup accounting and security on the judgehosts. See section 3.7 (installation of a judgehost).
- A lot of standard (GNU) programs, a probably incomplete list: hostname, date, dirname, basename, touch, chmod, cp, mv, cat, grep, diff, wc, mkdir, mkfifo, mount, sleep, head, tail, pgrep, zip, unzip.
- Sudo to gain root privileges.
- A LaTeX installation to regenerate the team PDF-manual with site specific configuration settings included.

The following items are optional, but may be required to use certain functionality or are generally useful.

- phpMyAdmin , to be able to access the database in an emergency or for data import/export
- An NTP daemon (for keeping the clocks between jury system and team workstations in sync)
- libcurl and libJSONcpp to use the command line submit client.
- libmagic (for command line submit client to detect binary file submissions)
- PECL xdiff extension (to reliably make diffs between submissions, DOMjudge will try alternative approaches if it is not available)
- beep for audible notification of errors, submissions and judgings, when using the default `alert` script.

Software required for building DOMjudge:

- gcc and g++ with standard libraries. Other compilers and libraries might also work: we have successfully compiled DOMjudge sources with Clang from the LLVM project; the C library should support the POSIX.1-2008 specification.
- GNU make

2.2.3 Requirements for team workstations

In the most basic setup the team workstations only need (next to the tools needed for program development) a web browser. The web interface fully works with any known browser, but a HTML5-capable browser adds more convenience functions. With JavaScript disabled, all basic functionality remains working, with the notable exception of multiple file uploads on non-HTML5-ready browsers.

3 Installation and configuration

This chapter details a fresh installation of DOMjudge. The first section is a Quick Installation Reference, but that should only be used by those already acquainted with the system. A detailed guide follows after that.

3.1 Quick installation

Note: this is not a replacement for the thorough installation instructions below, but more a cheat-sheet for those who've already installed DOMjudge before and need a few hints. When in doubt, always consult the full installation instruction.

External software:

- Install the MySQL-server and set a root password for it.
- Install Apache, PHP and (recommended) phpMyAdmin.
- Make sure PHP works for the web server and command line scripts.
- Install necessary compilers on the judgehosts.
- See also 3.2 (an example command line for Debian and RedHat).

DOMjudge:

- Extract the source tarball and run `./configure [-enable-fhs|-prefix=<basepath>] -with-baseurl=<url>`.
- Run `make domserver judgehost docs` or just those targets you want installed on the current host.
- Run `make install-{domserver,judgehost,docs}` as root to install the system.

On the domserver host:

- Install the MySQL database using e.g. `bin/dj_setup_database -u root -r install` on the domserver host.
- Add `etc/apache.conf` to your Apache configuration, edit it to your needs, reload web server: `sudo ln -s <INSTALL_PATH>/domserver/etc/apache.conf /etc/apache2/conf-available/domjudge.conf && sudo a2enconf domjudge && sudo apache2ctl graceful`
- Check that the web interface works (/team, /public and /jury).
- Change the admin password from its default value ('admin').
- Check that the API (/api) works and create credentials for the judgehosts.
- Create teams, user accounts and add useful contest data through the jury web interface or with phpMyAdmin.
- Run the config checker in the jury web interface.

On the judgehosts:

- RedHat: `useradd -d /nonexistent -g nobody -M -n -s /bin/false domjudge-run`
  Debian: `useradd -d /nonexistent -g nogroup -s /bin/false domjudge-run`
  (check specific options of useradd, since these vary per system)
  and also execute: `groupadd domjudge-run`
- Add to `/etc/sudoers.d/` or append to `/etc/sudoers` the sudoers configuration as in `etc/sudoers-domjudge`.
- Set up cgroup support: enable kernel parameters in `/etc/default/grub` and reboot, then use `misc/create_cgroups` to create cgroups for DOMjudge.
- Put the right credentials in the file `etc/restapi.secret` on all judgehosts (copied from the domserver).
- Start the judge daemon: `bin/judgedaemon`

It should be done by now. As a check that (almost) everything works, the set of test sources can be submitted:

```
cd tests
make check
```

Note that this requires some configuration depending on the `AUTH_METHOD` selected in `etc/domserver-config.php`, see 3.8 (submit client configuration) for more details.

Then, in the main jury web interface, select the admin link *judging verifier* to automatically verify most of the test sources, except for a few with multiple possible outcomes; these have to be verified by hand. Read the test sources for a description of what should (not) happen.

Optionally:

- Install the submit client on the team workstations.
- Start the balloon notification daemon: `cd bin; ./balloons`; or use the balloon web interface.
- On the judgehosts, create a pre-built chroot tree to support interpreted or byte-compiled languages such as Java:
  `sudo bin/dj_make_chroot [optional arguments]`
  `$EDITOR lib/judge/chroot-startstop.sh`
  Modifying `chroot-startstop.sh` is typically not necessary, but might be in circumstances where your interpreters are not installed under `/usr` or require files from other locations. See also the appendix on A.1 (setting up a chroot).
- For additional features in the jury web interface, the following PHP extensions can be installed:
  - xdiff PECL extension for diffs between submissions;

3.2 Prerequisites

For a detailed list of the hardware and software requirements, please refer to the previous chapter on contest planning.

3.2.1 Debian and RedHat installation commands

For your convenience, the following command will install needed software on the DOMjudge server as mentioned above when using Debian GNU/Linux, or one of its derivate distributions like Ubuntu.

```
sudo apt install gcc g++ make zip unzip mysql-server \
        apache2 php php-cli libapache2-mod-php \
        php-gd php-curl php-mysql php-json php-zip \
        php-mcrypt php-gmp php-xml php-mbstring \
        bsdmainutils ntp phpmyadmin libcgroup-dev \
        linuxdoc-tools linuxdoc-tools-text \
        groff texlive-latex-recommended texlive-latex-extra \
        texlive-fonts-recommended texlive-lang-european
```

Note that PHP modules may need to be enabled depending on your distribution. E.g. on Ubuntu run

```
sudo phpenmod json
```

to enable the JSON module.

For Debian 9 Stretch and up, add `php-mbstring`.

The following command can be used on RedHat Enterprise Linux, and related distributions like CentOS and Fedora.

```
sudo yum install gcc gcc-c++ make zip unzip mariadb-server \
        httpd php-gd php-cli php-mbstring php-mysql \
        ntp linuxdoc-tools libcgroup-devel \
        texlive-collection-latexrecommended texlive-wrapfig
```

Note that the TeX Live packages `expdlist`, `moreverb`, and `svn` still have be installed manually to rebuild the team manuals. Furthermore, `phpmyadmin` is available from the Fedora EPEL repository .

Finally, to build the command-line submit client, install the following additional software.

```
sudo apt install libcurl4-gnutls-dev libjsoncpp-dev libmagic-dev
```

```
sudo yum install libcurl-devel jsoncpp-devel file-devel
```

The package `jsoncpp-devel` is available in Fedora, but not in RHEL/CentOS.

Libmagic is not strictly required, but highly recommended for detecting binary file submissions. Pass the option `-enable-static-linking` to configure so that these libraries are statically linked into the `submit` binary and not needed on the team workstations where `submit` is installed.

On a judgehost, the following should be sufficient. The last two lines show some example compilers to install for C, C++, Java (OpenJDK), Haskell and Pascal; change the list as appropriate.

For Debian:

```
sudo apt install make sudo debootstrap libcgroup-dev \
        php-cli php-curl php-json php-zip procps \
        gcc g++ openjdk-8-jre-headless \
        openjdk-8-jdk ghc fp-compiler
```

For RedHat:

```
sudo yum install make sudo libcgroup-devel \
        php-cli php-mbstring php-process procps-ng \
        gcc gcc-c++ glibc-static libstdc++-static \
        java-1.7.0-openjdk-headless java-1.7.0-openjdk-devel \
        ghc-compiler fpc
```

Note that `fpc` is not available in RedHat 7.

3.3 Installation system

There is a separate *maintainer installation* method meant for those wishing to do development on the DOMjudge source code. See the C (appendix with developer information) and skip the rest of this section.

The DOMjudge build/install system consists of a `configure` script and makefiles, but when installing it, some more care has to be taken than simply running '`./configure && make && make install`'. DOMjudge needs to be installed both on the server and on the judgehosts. These require different parts of the complete system to be present and can be installed separately. Within the build system these parts are referred to as `domserver, judgehost` and additionally `docs` for all documentation.

DOMjudge can be installed with two different directory layouts:

Single directory tree

With this method all DOMjudge related files and programs are installed in a single directory tree which is specified by the prefix option of configure, like

```
./configure --prefix=$HOME/domjudge --with-baseurl=https://domjudge.example.com/
```

This will install each of the `domserver, judgehost, docs` parts in a subdirectory `$HOME/domjudge/domserver` etc. These subdirectories can be overridden from the defaults with options like `-with-domserver_root=DIR`, see `configure -help` for a complete list. The prefix defaults to `/opt/domjudge`.

Besides the installed files, there will also be directories for logging, temporary files, submitted sources and judging data:

`log`
: contains all log files.

`tmp`
: contains temporary files.

`submissions`
: (optionally) on the domserver contains all correctly submitted files: as backup only, the database is the authoritative source. Note that this directory must be writable by the web server for this feature to work.

`judgings`
: location on judgehosts where submissions are tested, each in its own subdirectory.

This method of installation is the default and probably most practical for normal purposes as it keeps all files together, hence easily found.

FHS compliant

This method installs DOMjudge in directories according to the Filesystem Hierarchy Standard . It can be enabled by passing the option `-enable-fhs` to `configure` and in this case the prefix defaults to `/usr/local`. Files will be placed e.g. in `PREFIX/share/domjudge, PREFIX/bin, PREFIX/var/log, PREFIX/etc/domjudge`, while `/tmp` will be used for temporary files. You may want to pass options `-sysconfdir=/etc` and `-localstatedir=/var` to `configure` to disable the prefix for these.

Note that the `-with-baseurl` configure option is not required but highly recommended, as it allows building the submit client and team documentation with the correct URL preset. The option is required when using 3.10 (OpenID). If needed, the setting can later be updated in `etc/domserver-static.php` on the domserver, and in `etc/submit-config.h` in the source tree for rebuilding the submit client.

After running the `configure` script, the system can be built and installed. Each of the `domserver, judgehost, docs` parts can be built and installed separately, respectively by:

```
make domserver && sudo make install-domserver
make judgehost && sudo make install-judgehost
make docs && sudo make install-docs
```

Note that root privileges are required to set permissions and user and group ownership of password files and a few directories. If you run the installation targets as non-root, you will be warned that you have to perform these steps manually. Although DOMjudge can be installed as root, one should *not* run DOMjudge programs and daemons under the root user, but under a normal user: `runguard` is specifically designed to be the only part invoked as root (through sudo) to make this unnecessary. Also, running as root will give rise to problems, see A.7 (runguard: root privileges not dropped) in the common problems section.

For a list of basic make targets, run `make` in the source root directory without arguments.

3.4 Database installation

DOMjudge uses a MySQL or MariaDB database server for information storage. Where this document talks about MySQL, it can be understood to also apply to MariaDB.

The database structure and privileges are included in MySQL dump files in the sql subdirectory. The default database name is `domjudge`. This can be changed manually in the `etc/dbpasswords.secret` file: the database name as specified in this file will be used when installing.

Installation of the database is done with `bin/dj_setup_database`. For this, you need an installed and configured MySQL server and administrator access to it. Run

```
dj_setup_database genpass
dj_setup_database [-u <admin_user>] [-p <password>|-r] install
```

This first creates the DOMjudge database credentials file `etc/dbpasswords.secret` (optionally change the random generated password, although it is not needed for normal operation). Then it creates the database and user and inserts some default/example data into the domjudge database. The option `-r` will prompt for a password for mysql; when no user is specified, the mysql client will try to read credentials from `$HOME/.my.cnf` as usual. The command `uninstall` can be passed to `dj_setup_database` to remove the DOMjudge database and users; *this deletes all data!*

The domjudge database contains a number of tables, some of which need to be manually filled with data before the contest can be run. See the 4.1 (database section of Contest setup) for details.

3.4.1 Setting up replication or backups

The MySQL server is the central place of information storage for DOMjudge. Think well about what to do if the MySQL host fails or loses your data.

A very robust solution is to set up a replicating MySQL server on another host. This will be a hot copy of all data up to the second, and can take over immediately in the event of failure. The MySQL manual has more information about setting this up.

Alternatively, you can make regular backups of your data to another host, for example with `mysqldump`, or using a RAID based system.

Replication can also be used to improve performance, by directing all select-queries to one or more replicated slave servers, while updates will still be done to the master. This is not supported out of the box, and will require making changes to the DOMjudge source.

3.4.2 Storage of submissions

The database is the authoritative version for submission source files; file system storage is available as an easy way to access the source files and as backup, but only when the web server has write permissions to `<domjudge_submitdir>`. File system storage is ignored if these permissions are not set. The programs `bin/save_sources2file` and `bin/restore_sources2db` are available to store and recover the submission table in the database to/from these files.

3.5 Web server configuration

For the web interface, you need to have a web server (e.g. Apache) installed on the domserver and made sure that PHP correctly works with it. Refer to the documentation of your web server and PHP for details.

To configure the web server for DOMjudge, use the Apache configuration snippet from `etc/apache.conf`. It contains examples for configuring the DOMjudge pages with an alias directive, or as a virtualhost, optionally with SSL; it also contains PHP and security settings. Reload the web server for changes to take effect.

```
ln -s etc/apache.conf /etc/apache2/conf-available/domjudge.conf
a2enconf domjudge
service apache2 reload
```

An Nginx webserver configuration snippet is also provided in `etc/nginx-conf`. This alternative webserver is less well-tested than the Apache setup, so your mileage may vary. Feedback is very welcome, though.

The judgehosts connect to DOMjudge via the DOMjudge API so need to be able to access at least this part of the web interface.

3.6 Fine tuning server settings

For Apache, there are countless documents on how to maximize performance. Of particular importance is to ensure that the `MaxClients` setting is high enough to receive the number of parallel requests you expect, but not higher than your amount of RAM allows. Furthermore, we recommend to turn `KeepAlive` off, or at least make sure that `KeepAliveTimeout` is set to only a few seconds. Otherwise, a large number of page view requests from teams and public can easily exhaust the Apache workers, resulting in an unresponsive website, which will also affect the judgedaemons.

As for PHP, the use of an opcode cache like the Alternative PHP Cache (Debian package: `php-apc`) is beneficial for performance. For uploading large testcases, see the A.6 (section about memory limits).

It may be desirable or even necessary to fine tune some MySQL default settings:

- `max_connections`: The default 100 is too low, because of the connection caching by Apache threads. 1000 is more appropriate.
- `max_allowed_packet`: The default of 16MB might be too low when using large testcases. This should be changed both in the MySQL server and client configuration and be set to about twice the maximum testcase size.
- Root password: MySQL does not have a password for the root user by default. It's very desirable to set one.
- When maximising performance is required, you can consider to use the *Memory* table storage engine for the scorecache and rankcache tables. They will be lost in case of a full crash, but can be recalculated from the jury interface.

3.7 Installation of a judgehost

A few extra steps might need to be taken to completely install and configure a judgehost.

For running solution programs under a non-privileged user, a user and group have to be added to the system(s) that act as judgehost. This user does not need a home-directory or password, so the following command would suffice to add a user and group 'domjudge-run' with minimal privileges.

On Debian and Redhat based Linux distributions use:

```
useradd -d /nonexistent -U -M -s /bin/false domjudge-run
groupadd domjudge-run
```

For other systems check the specifics of your useradd command. This user must also be configured as the user under which programs run via `configure -enable-runuser=USER`; the default is `domjudge-run`. By default the group is set to the same, this can be modified with the option `-enable-rungroup=GROUP`

`Runguard` needs to be able to become root for certain operations like changing to the runuser and performing a chroot. Also, the default `chroot-startstop.sh` script uses sudo to gain privileges for certain operations. There's a pregenerated `/etc/sudoers.d/` snippet in `etc/sudoers-domjudge` that contains all required rules. You can put the lines in the snippet at the end of `/etc/sudoers`, or, for modern sudo versions, place the file in `/etc/sudoers.d/`. If you change the user you run the judgedaemon as, or the installation paths, be sure to update the sudoers rules accordingly.

When the chroot setting is enabled (default), a static POSIX shell has to be available for copying it to the chroot environment. For Linux i386, a static Dash shell is included, which works out of the box, also for the Linux Intel/AMD 64 architecture. For other architectures or operating systems, a shell has to be added manually. Then simply point the `lib/sh-static` symlink to this file. If you want to support languages that cannot be compiled to statically linked binaries, e.g. byte-compiled languages such as Java, or interpreted languages such as Python, then a complete chroot environment must be built and configured. See the appendix on A.1 (setting up a chroot) for more details.

DOMjudge uses Linux Control Groups or cgroups for process isolation in the judgedaemon. Linux cgroups give more accurate measurement of actually allocated memory than traditional resource limits (which is helpful with interpreters like Java that reserve but not actually use lots of memory). Also, cgroups are

used to restrict network access so no separate measures are necessary, and they allow running multiple judgedaemons on a multi-core machine by using CPU binding.

The judgedaemon needs to run a recent Linux kernel (at least 3.2.0). The following steps configure cgroups on Debian wheezy. Instructions for other distributions may be different (send us your feedback!). Edit grub config to add cgroup memory and swap accounting to the boot options. Edit `/etc/default/grub` and change the default commandline to

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet cgroup_enable=memory swapaccount=1"
```

Then run `update-grub` and reboot.

You have now configured the system to use cgroups, but you need to create the actual cgroups that DOMjudge will use. For that, you can use the script under `misc-tools/create_cgroups`. Edit the script to match your situation first. This script needs to be re-run after each boot (e.g., add it to the judgedaemon init script).

The judgehost connects to the domserver via a REST API. You need to create an account for the judgedaemons to use (this may be a shared account between all judgedaemons) with a difficult, random password and the 'judgehost' role. On each judgehost, copy from the domserver (or create) a file `etc/restapi.secret` containing the id, URL, username and password whitespace-separated on one line, for example:

```
default http://example.edu/domjudge/api/  judgehosts  MzfJYWF5agSlUfmiGEy5mgkfqU
```

Note that the password must be identical to that of the `judgehost` user in the admin web interface. Multiple lines may be specified to allow a judgedaemon to work for multiple domservers. The id is used to differentiate between multiple domservers, and should be unique within the `restapi.secret` file.

Then start the judgedaemon:

```
bin/judgedaemon
```

Upon its first connection to the domserver API the judgehost will be auto-registered and will be by default enabled. If you wish to add a new judgehost but have it initially disabled, you can add it manually through the DOMjudge web interface and set it to disabled before starting the judgedaemon.

3.8 Building and installing the submit client

DOMjudge supports two submission methods: via the command line `submit` program and via the web interface. From experience, both methods have users that prefer the one above the other.

The command line submit client sends submissions using the API interface internally. This requires the libcURL and libjsonCPP library development files at compile time. The submit client can be statically linked using the `-enable-static-linking` configure option to avoid a runtime dependency.

The submit client can be built with `make submitclient`. There is no make target to install the submit client, as its location will very much depend on the environment. You might e.g. want to copy it to all team computers or make it available on a network filesystem. Note that if the team computers run a different (version of the) operating system than the jury systems, then you need to build the submit client for that OS.

The submit client needs to know the URL of the domserver. This can be passed as a command line option or environment variable. The latter option makes for easier usage. A sample script `submit_wrapper.sh` is included, which sets this variable. See that script for more details on how to set this up.

The submit client authenticates to the DOMjudge API via either the configured authentication scheme, or can use the DOMjudge internal username and password combination for a given user account regardless of authentication scheme. For example, when the IPADDRESS scheme is used, no additional configuration is required because submissions will come from the correct IP address of the team. When another scheme is used, it may be necessary to place username and password combinations in the team's account so the submit client can use those. In this case these are always the DOMjudge internal password, so not e.g. LDAP passwords when using that scheme. The credentials are placed in the file `~/.netrc`, with example content:

```
machine domserver.example.com login user0123 password Fba^2bHzz
```

See the netrc(4) manual page for more details. You may want to distribute those `.netrc` files in advance to the team accounts. Make sure they are only readable for the user itself.

3.8.1 The submit client under Windows/Cygwin

Note: this feature is not well supported anymore; we recommend using the web interface for submitting in Windows.

The submit client can also be built under Windows when the Cygwin environment is installed. First install *Cygwin* <https://cygwin.com/install.html> , and include GCC, curl-devel and maybe some more packages. When Cygwin is correctly installed with all necessary development tools, the submit binary can be created by running `configure` followed by `make submit.exe` in the `submit` directory.

3.9 Configuration

Configuration of the judge system is mostly done by editing the configuration variables on the page `Configuration settings` available in the administrator interface, and changes take effect immediately. The administrator interface can be reached on `http(s)://yourhost.example.edu/domjudge/jury/` and the default username is `admin` with password `admin`. Make sure to change the default password immediately.

Some settings that are tightly coupled to the filesystem can be configured in the files in `etc`: `domserver-config.php, judgehost-config.php, common-config.php` for the configuration options of the domserver, judgehost and shared configuration options respectively. Descriptions of settings are included in these files. The judgedaemon must be restarted for changes to take effect, while these are directly picked up by the webinterfaces.

Besides these settings, there are a few other places where changes can be made to the system, see 3.15 (other configurable scripts).

3.10 OpenID Connect

DOMjudge supports allowing users to log in using your favorite OpenID Connect provider, such as google (or ICPC accounts in the future). Some things need to be configured on the `Configuration Settings` page for this to work properly.

3.10.1 Google

To use Google as an OpenID Provider, you need to obtain a clientID and clientSecret. You can get these values from Google Developer Console .

Navigate to `Credentials` under the `API Manager` heading. Then `Create Credentials` for `OAuth Client ID`. This will give you the client ID and secret you need. For the openid provider set the URL to `https://accounts.google.com`. You also need to set the `Authorized redirect URIs` to the full url to `/domjudge/auth/oid_cb.php`. E.g. `http(s)://yourhost.example.edu/domjudge/auth/oid_cb.php`

You also need to set the authentication scheme to `PHP_SESSIONS` in `etc/domserver-config.php` (the default) and make sure that the DOMjudge webserver base URL was set correctly with `configure -with-baseurl=<URL>`. For example, this will probably be something like this: `http(s)://yourhost.example.edu/domjudge`

Finally, enable the setting `Allow openid auth`.

3.11 Executables

DOMjudge supports executable archives (uploaded and stored in ZIP format) for configuration of languages, special run and compare programs. The archive must contain an executable file named `build` or `run`. When deploying a new (or changed) executable to a judgehost `build` is executed *once* if present. Afterwards an executable file `run` must exist (it may have existed before), that is called to execute the compile, compare, or run script. The specific formats are detailed below.

Executables may be changed via the web interface in an online editor or by uploading a replacement zip file. Changes apply immediately to all further uses of that executable.

3.12 Configuration of languages

Compilers can be configured by creating or selecting/editing an executable in the web interface. When compiling a set of source files, the `run` executable is invoked with the following arguments: destination file name, memory limit (in KB), main (first) source file, other source files. For more information, see for example the executables `c` or `java_javac_detect` in the web interface. Note that compile scripts are included for most common languages already.

Interpreted languages and non-statically linked binaries (for example, Oracle Java) can in principle also be used, but require that all dependencies are added to the chroot environment. DOMjudge comes with helper scripts to build a chroot environment: run `sudo bin/dj_make_chroot`. Options can be given change the target directory where to build the chroot environment (normally the default from configure is used) and to select the target machine architecture. Start the script without arguments for usage information. See also sections 3.7 (Installation of a judgehost) and A.1 (Problems: Setting up a chroot).

Interpreted languages do not generate an executable and in principle do not need a compilation step. However, to be able to use interpreted languages (also Oracle's Java), during the compilation step a script must be generated that will function as the executable: the script must run the interpreter on the source. See for example `pl` and `java_javac_detect` in the list of executables.

3.13 Configuration of special run and compare programs

To allow for problems that do not fit within the standard scheme of fixed input and/or output, DOMjudge has the possibility to change the way submissions are run and checked for correctness.

The back end script `testcase_run.sh` that handles the running and checking of submissions, calls separate programs for running submissions and comparison of the results. These can be specialised and adapted to the requirements per problem. For this, one has to create executable archives as described above. Then the

executable must be selected in the `special_run` and/or `special_compare` fields of the problem (an empty value means that the default run and compare scripts should be used; the defaults can be set in the global configuration settings). When creating custom run and compare programs, we recommend re-using wrapper scripts that handle the tedious, standard part. See the boolfind example for details.

3.13.1 Compare programs

Compare scripts/programs should follow the Kattis/problemarchive output validator format . DOMjudge uses the default output validator specified there as its default, which can be found at

<https://github.com/Kattis/problemtools/blob/master/support/default_validator/> .

Note that DOMjudge only supports a subset of the functionality described there. In particular, the calling syntax is

```
/path/to/compare_script/run <testdata.in> <testdata.ans> <feedbackdir> <compare_args> < <program.out
```

where `testdata.in testdata.ans` are the jury reference input and output files, `feedbackdir` the directory containing e.g. the judging response file `judgemessage.txt` to be written to (the only other permitted files there are `teammessage.txt score.txt judgeerror.txt diffposition.txt`), `compare_args` a list of arguments that can set when configuring a contest problem, and `program.out` the team's output. The validator program should not make any assumptions on its working directory.

For more details on writing and modifying a compare (or validator) scripts, see the `boolfind_cmp` example and the comments at the top of the file `testcase_run.sh`.

3.13.2 Run programs

Special run programs can be used, for example, to create an interactive problem, where the contestants' program exchanges information with a jury program and receives data depending on its own output. The problem `boolfind` is included as an example interactive problem, see `docs/examples/boolfind.pdf` for the description.

Usage is similar to compare programs: you can either create a program `run` yourself, or use the provided wrapper script, which handles bi-directional communication between a jury program and the contestants' program on stdin/stdout (see the `run` file in the `boolfind_run` executable).

For the first case, the calling syntax that the program must accept is equal to the calling syntax of `run_wrapper`, which is documented in that file. When using `run_wrapper`, you should copy it to `run` in your executable archive. The jury must write a program named exactly `runjury`, accepting the calling syntax

```
runjury <testdata.in> <program.out>
```

where the arguments are files to read input testdata from and write program output to, respectively. This program will communicate via stdin/stdout with the contestants' program. A special compare program must probably also be created, so the exact data written to <`program.out`> is not important, as long as the correctness of the contestants' program can be deduced from the contents by the compare program.

3.14 Alerting system

DOMjudge includes an alerting system. This allows the administrator to receive alerts when important system events happen, e.g. an error occurs, or a submission or judging is made.

These alerts are passed to a plugin script `alert` which can easily be adapted to fit your needs. The default script emits different beeping sounds for the different messages when the `beep` program is available, but it could for example also be modified to send a mail on specific issues, connect to monitoring software like Nagios, etc. For more details, see the script `lib/alert`.

3.15 Other configurable scripts

There are a few more places where some configuration of the system can be made. These are sometimes needed in non-standard environments.

- In `bin/dj_make_chroot` on a judgehost some changes to variables can be made, most notably `DEBMIRROR` to select a Debian mirror site near you.
- The script `lib/judge/chroot-startstop.sh` can be modified to suit your local environment. See comments in that file for more information.

3.16 Logging & debugging

All DOMjudge daemons and web interface scripts support logging and debugging in a uniform manner via functions in `lib.error.*`. There are three ways in which information is logged:

- Directly to `stderr` for daemons or to the web page for web interface scripts (the latter only on serious issues).
- To a log file set by the variable `LOGFILE`, which is set in each program. Unsetting this variable disables this method.
- To syslog. This can be configured via the `SYSLOG` configuration variable in `etc/common-config.php`. This option gives the flexibility of syslog, such as remote logging. See the syslog(daemon) documentation for more information. Unsetting this variable disables this method.

Each script also defines a default threshold level for messages to be logged to stderr (`VERBOSE`: defaults to `LOG_INFO` in daemons and `LOG_ERR` in the web interface) and for log file/syslog (`LOGLEVEL`: defaults to `LOG_DEBUG`).

In case of problems, it is advisable to check the logs for clues. Extra debugging information can be obtained by setting the config option `DEBUG` to a bitwise-or of the available `DEBUG_*` flags in `etc/common-config.php`, to e.g. generate extra SQL query and timing information in the web interface.

3.17 (Re)generating documentation and the team manual

There are three sets of documentation available under the `doc` directory in DOMjudge:

the admin-manual

for administrators of the system (this document),

the judge-manual

for judges, describing the jury web interface and giving some general information about this system,

the team-manual

for teams, explaining how to use the system and what restrictions there are.

The team manual is only available in PDF format and must be built from the LaTeX sources in `doc/team` after configuration of the system. A prebuilt team manual is included, but note that it contains default/example values for site-specific configuration settings such as the team web interface URL and judging settings such as the memory limit. We strongly recommend rebuilding the team manual to include site-specific settings and also to revise it to reflect your contest specific environment and rules.

Besides a standard LaTeX installation, the team manual requires the `svn` and `expdlist` packages. These are available in TeX Live in the `texlive-latex-extra` package in any modern Linux distribution. Alternatively, you can download and install them manually from their respective subdirectories in <http://mirror.ctan.org/macros/latex/contrib> .

When the `docs` part of DOMjudge is installed and site-specific configuration set, the team manual can be generated with the command `genteammanual` found under `docs/team`. The PDF document will be placed in the current directory or a directory given as argument. The following should do it on a Debian-like system:

```
sudo apt-get install make texlive-latex-extra texlive-latex-recommended texlive-lang-european
cd <INSTALL_PATH>/docs/team
./genteammanual [targetdir]
```

The team manual is currently available in two languages: English and Dutch. We welcome any translations to other languages.

The administrator's and judge's manuals are available in PDF and HTML format and prebuilt from SGML sources. Rebuilding these is not normally necessary. To rebuild them on a Debian-like system, the following commands should do it:

```
sudo apt-get install linuxdoc-tools make zip ghostscript groff texlive-latex-recommended
make -C doc/admin docs
make -C doc/judge docs
```

3.18 Optional features

3.18.1 Multiple judgedaemons per machine

You can run multiple judgedaemons on one multi-cpu or multi-core machine, dedicating one cpu core to each judgedaemon.

To that end, add extra unprivileged users to the system, i.e. add users `domjudge-run-<X>` (where `X` runs through `0,1,2,3`) with `useradd` as described in section 3.7 (installation of a judgehost). Finally, start each of the judgedaemons with

```
judgedaemon -n <X>
```

to bind it to core X.

3.18.2 Encrypted communications (HTTPS)

DOMjudge can be configured to run on HTTPS, so teams and judgedaemons communicate with the domserver securely over encrypted SSL/TLS connections. Setting up SSL for Apache is documented in the Apache manual and in many tutorials around the web.

The judgedaemons must recognise the CA you're using, otherwise they will refuse to connect over HTTPS. If your judgedaemon gives an error message about an untrusted certificate, put your domserver's certificate in `/etc/ssl/certs/yourname.crt` of each judgehost (and on the team machines when using the commandline submit client) and run:

```
sudo c_rehash
```

When loading teams from the ICPC registration system through the import feature in DOMjudge, the certificate from icpc.baylor.edu must similarly be accepted by your local installation or if not, added via the procedure above.

3.18.3 NTP time synchronisation

We advise to install an NTP-daemon (Network Time Protocol) to make sure the time between domserver, judgehosts, and jury and team computers is in sync.

3.18.4 Printing

It is recommended to configure the local desktop printing of team workstations where ever possible: this has the most simple interface and allows teams to print from within their editor.

If this is not feasible, DOMjudge includes support for printing via the DOMjudge web interface: the DOMjudge server then needs to be able to deliver the uploaded files to the printer. It can be enabled via the `enable_printing` configuration option in the administrator interface. The exact command used to send the files to a printer can be changed in the function `send_print()` in `lib/www/printing.php`.

3.18.5 Judging consistency

The following issues can be considered to improve consistency in judging.

- Disable CPU frequency scaling and Intel "Turbo Boost" to prevent fluctuations in CPU power.
- Disable address-space randomization to make programs with memory addressing bugs give more reproducible results. To do that, you can add the following line to `/etc/sysctl.conf`:

```
kernel.randomize_va_space=0
```

This will restore these settings permanently across reboots. Then run the following command:

```
sudo sysctl -p
```

to directly activate these settings.

3.19 Upgrading

There is some support to upgrade DOMjudge to newer versions. Note that this functionality is not extensively tested, so when you plan to upgrade, *you are strongly advised to backup the DOMjudge database and other data before continuing*. We also advise to check the `ChangeLog` file for important changes.

Upgrading the filesystem installation is probably best done by installing the new version of DOMjudge in a separate place and transferring the configuration settings from the old version.

There are SQL upgrade scripts to transform the database including its data to the layout of a newer version. The scripts can be found under `sql/upgrade` and each script applies changes between two consecutive DOMjudge versions. At the beginning of each script, a check is performed which will let MySQL bail out with an error if it should not be applied anymore. Note that the scripts must be applied in order (sorted by release). These scripts can be applied by running `dj_setup_database upgrade`. Be aware that these scripts are conservative in adding and upgrading SQL data, so check that e.g. new compile scripts are present or add them manually, and check the upgrade scripts manually for any other data upgraded.

4 Setting up a contest

After installation is successful, you want to run your contest! Configuring DOMjudge to run a contest (or a number of them, in sequence) involves the following steps:

- Configure the contest data;
- Set up authentication for teams;
- Supply in- and output testdata;
- Check that everything works.

4.1 Configure the contest data

DOMjudge stores and retrieves most of its data from the MySQL database. Some information must be filled in beforehand, other tables will be populated by DOMjudge.

You can use the jury web interface to add, edit and delete most types of data described below. It's advised to keep a version of phpMyAdmin handy in case of emergencies, or for general database operations like import and export.

This section describes the meaning of each table and what you need to put into it. Tables marked with an 'x' are the ones you have to configure with contest data before running a contest (via the jury web interface or e.g. with phpMyAdmin), the other tables are used automatically by the software:

	auditlog	Log of every state-changing event.
	balloon	Balloons to be handed out.
	clarification	Clarification requests/replies are stored here.
x	configuration	Runtime configuration settings.
x	contest	Contest definitions with start/end time.
x	contestproblem	Coupling of problems to contests and data specific to it.
x	contestteam	Coupling of teams to contests.
	event	Log of events during contests.
x	executable	Executable compile/run/compare scripts.
	internal_error	Stores errors that occurred on judgehosts including logs.
	judgehost	Computers (hostnames) that function as judgehosts.
x	judgehost_restriction	Optional restriction sets on submissions taken by judgehosts.
	judging	Judgings of submissions.
	judging_run	Result of one testcase within a judging.
x	language	Definition of allowed submission languages.
x	problem	Definition of problems (name, timelimit, etc.).
	rankcache	Cache of team ranking data for public/teams and for the jury.
	rejudging	Metadata for batched rejudging.
	role	Possible user roles.
	scorecache	Cache of the scoreboards for public/teams and for the jury.
	submission	Submission metadata of solutions to problems.
	submission_file	Submitted code files.
x	team	Definition of teams.
x	team_affiliation	Definition of institutions a team can be affiliated with.
x	team_category	Different category groups teams can be put in.
	team_unread	Records which clarifications are read by which team.
x	testcase	Definition of testdata for each problem.
x	user	Users that will able to access the system.
x	userrole	Mapping of users to their roles.

Now follows a longer description (including fields) per table that has to be filled manually. As a general remark: almost all tables have an identifier field. Most of these are numeric and automatically increasing; these do not need to be specified. The tables `executable` and `language` have text strings as identifier fields. These need to be manually specified and only alpha-numeric, dash and underscore characters are valid, i.e. `a-z, A-Z, 0-9, -, _`.

configuration

This table contains configuration settings. These entries are simply stored as `name, value` pairs, where the values are JSON encoded, `type` contains the allowed data type, and `description` documents the configuration setting.

contest

The contests that the software will run. E.g. a test session and the live contest.

`cid` is the reference ID and `contestname` is a descriptive name used in the interface, while `shortname` is the publicly visible identifier.

`activatetime`, `starttime` and `endtime` are required fields and specify when this contest is active and open for submissions. Optional `freezetime` and `unfreezetime` control scoreboard freezing and `deactivatetime` when the contest is not visible anymore. For a detailed treating of these, see section 4.2 (Contest milestones). All contest times can be specified relative to `starttime`, except of course `starttime` itself. The input given in the jury interface (either relative or absolute) is stored in the

`*time_string` fields, while a calculated absolute version is stored in the fields without the `_string` suffix.

The `public` field can be used to limit which contests are displayed as public scoreboards (as opposed to privately to a selected set of teams), while `enabled` can be used to (temporarily) disable a contest altogether.

contestproblem

This table couples problems to contests: `cid` and `probid` describe the pairing.

Furthermore, it stores problem data that is specific for the included contest: `shortname` is a contest-unique identifier string for the problem, `points` defaults to 1 and can be set to assign non-even scoring; `allow_submit` determines whether teams can submit solutions for this problem. Non-submittable problems are also not displayed on the scoreboard. This can be used to define spare problems, which can then be added to the contest quickly; `allow_judge` determines whether judgehosts will judge submissions for this problem. See also the explanation for language.

The `color` tag can be filled with a CSS colour specification to associate with this problem; see also section 6.2.1 (Scoreboard: colours).

contestteam

This table couples teams to contests. Teams can only submit solutions to problems in contests that are public or which they are part of.

executable

This table stores zip-bundles of executable scripts that can be used as compile, run, and compare scripts.

judgehost_restriction

This table encodes restriction sets for selecting which submissions are sent to a judgehost. The restrictions are JSON encoded in the `restrictions` column, and can be set in the admin web interface to restrict on specific contests, problems, languages, and to never rejudge on the same judgehost. A restriction set can be assigned to judgehost(s) on the edit page of the judgehosts overview.

language

Programming languages in which to accept and judge submissions. `langid` is a string of maximum length 8, which references the language. `name` is the displayed name of the language; `extensions` is a JSON encoded list of recognized filename extensions; `allow_submit` determines whether teams can submit using this language; `allow_judge` determines whether judgehosts will judge submissions for this problem. This can for example be set to *no* to temporarily hold judging when a problem occurs with the judging of a specific language; after resolution of the problem this can be set to *yes* again.

`time_factor` is the relative factor by which the timelimit is multiplied for solutions in this language; `compile_script` refers to a compile executable script that is used for this language.

problem

This table contains the problem definitions. `probid` is the reference ID, `cid` is the contest ID this problem is (only) defined for: a problem cannot be used in multiple contests. `name` is the full name (description) of the problem.

`allow_submit` determines whether teams can submit solutions for this problem. Non-submittable problems are also not displayed on the scoreboard. This can be used to define spare problems, which can then be added to the contest quickly; `allow_judge` determines whether judgehosts will judge submissions for this problem. See also the explanation for language.

`timelimit` is the timelimit in seconds within which solutions for this problem have to run (taking into account `time_factor` per language). See also A.9 (enforcement of time limits) for more details.

`memlimit` is the memory limit in kB allotted for this problem. If empty then the global configuration setting `memory_limit` is used. Equivalently for `outputlimit`.

`special_run` if not empty defines a custom run program `run_<special_run>` to run compiled submissions for this problem and `special_compare` if not empty defines a custom compare program `compare_<special_compare>` to compare output for this problem.

The `color` tag can be filled with a CSS colour specification to associate with this problem; see also section 6.2.1 (Scoreboard: colours).

In `problemtext` a PDF, HTML or plain text document can be placed which allows team, public and jury to download the problem statement. Note that no additional filtering takes place, so HTML (and PDF to some extent) should be from a trusted source to prevent cross site scripting or other attacks. The file type is stored in `problemtext_type`.

team

Table of teams: `teamid` is (internal) ID of the team, while `externalid` can be used to store an ID for im/exporting to other systems. `name` the displayed name of the team, `categoryid` is the ID of the category the team is in; `affilid` is the affiliation ID of the team.

When `enabled` is set to 0, the team immediately disappears from the scoreboards and cannot use the team web interface anymore, even when already logged in. One use case could be to disqualify a team on the spot.

`members` are the names of the team members, separated by newlines and `room` is the location or room of the team, both for display only; `comments` can be filled with arbitrary useful information and is only visible to the jury. The timestamp `teampage_first_visited` and the `hostname` field indicate when/whether/from where a team visited its team web interface.

The `penalty` field can be used to give this team a (positive or negative) number of penalty minutes to correct for exceptional circumstances.

team_affiliation

`affilid` is the reference ID and `name` the name of the institution. `country` should be the 3 character ISO 3166-1 alpha-3 abbreviation of the country and `comments` is a free form field that is displayed in the jury interface.

A country flag can be displayed on the scoreboard. For this to work, the `country` field must match a (flag) picture in `www/images/countries/<country>.png`. All country flags are present there, named with their 3-character ISO codes. See also `www/images/countries/README`.

team_category

`categoryid` is the reference ID and `name` is a string: the name of the category. `sortorder` is the order at which this group must be sorted in the scoreboard, where a higher number sorts lower and equal sort depending on score.

The `color` is again a CSS colour specification used to discern different categories easily. See also section 6.2.1 (Scoreboard: colours).

The `visible` flag determines whether teams in this category are displayed on the public/team scoreboard. This feature can be used to remove teams from the public scoreboard by assigning them to a separate, invisible category.

testcase

The testcase table contains testdata for each problem; `testcaseid` is a unique identifier, `input` and `output` contain the testcase input/output and `image` an optional graphical representation of the testcase for the jury. The fields `md5sum_input`, `md5sum_output`, and `md5sum_image` contain their respective md5 hashes to check for up-to-date-ness of cached versions by the judgehosts and `image_thumb` and `image_type` a thumbnail version and mimetype string for the image. The field `probid` is the corresponding problem and `rank` determines the order of the testcases for one problem. `description` is an optional description for this testcase. See also 4.4 (providing testdata).

user

This table has the users that the system knows about with their login credentials. Each user may have one or more roles, like being part of a team, being a jury member or administrator. There are also functional accounts, like for judgedaemons.

4.2 Contest milestones

The `contest` table specifies timestamps for each contest that mark specific milestones in the course of the contest.

The triplet *activatetime*, *starttime* and *endtime* define when the contest runs and are required fields (activatetime and starttime may be equal).

activatetime is the moment when a contest first becomes visible to the public and teams . Nothing can be submitted yet and the problem set is not revealed. Clarifications can be viewed and sent.

At starttime, the scoreboard is displayed and submissions are accepted. At endtime the contest stops. New incoming submissions will still be processed and judged, but the result will not be shown anymore to teams; they instead receive the verdict'too-late'. Unjudged submissions received before endtime will still be judged normally.

freezetime and *unfreezetime* control scoreboard freezing. freezetime is the time after which the public and team scoreboard are not updated anymore (frozen). This is meant to make the last stages of the contest more thrilling, because no-one knows who has won. Leaving them empty disables this feature. When using this feature, unfreezetime can be set to automatically 'unfreeze' the scoreboard at that time. For a more elaborate description, see also section 6.2.3 (Scoreboard: freezing and defrosting).

The scoreboard, results and clarifications will remain to be displayed to team and public after a contest, until the *deactivatetime*.

All events happen at the first moment of the defined time. That is: for a contest with starttime "12:00:00" and endtime "17:00:00", the first submission will be accepted at 12:00:00 and the last one at 16:59:59.

The following ordering must always hold: activatetime <= starttime < (freezetime <=) endtime (<= unfreezetime) (<= deactivatime).

4.3 User authentication

The authentication system lets the domserver know which user it is dealing with and which role(s) the user has. This system is modular, allowing flexible addition of new methods, if required. The following methods are available by default for authentication.

4.3.1 PHP session with passwords (default)

Each user receives a password and PHP's session management is used to keep track of which user is logged in. This method is easiest to setup. It does require the administrator to generate users and passwords for all teams (this can be done in the jury interface) and distribute those, though. Also, each team has to login each time they (re)start their browser. The password is stored in a salted MD5 hash in the `password` field in database (user table).

4.3.2 IP-address based

The IP-address of a team's workstation is used as the primary means of authentication. The system assumes that someone coming from a specific IP is the user with that IP listed in the user table. When a team browses to the web interface, this is checked and the appropriate team page is presented.

This method has the advantage that teams do not have to login. A requirement for this method is that each team computer has a separate IP-address from the view of the domserver, though, so this is most suitable for onsite contests and might not work with online contests if multiple teams are located behind a router, for example.

There are two possible ways of configuring team IP-addresses.

Supply it beforehand

Before the contest starts, when entering teams into the database, add the IP that each team will have to that user's entry in the `ip_address` field. When the teams arrive, everything will work directly and without further configuration (except when teams switch workplaces). If possible, this is the recommended modus operandi, because it's the least hassle just before and during the contest.

Use one-time passwords

Supply the teams with a one time username and password with which to authenticate. Beforehand, generate passwords for each team in the jury interface. When the test session (or contest) starts and a team connects to the web interface and have an unknown IP, they will be prompted for username and password. Once supplied, the IP is stored and the password is removed and not needed anymore the next time.

This is also a secure option, but requires a bit more hassle from the teams, and maybe from the organisers who have to distribute pieces of paper.

Note: the web interface will only allow a team to authenticate themselves once. If an IP is set, a next authentication will be refused (to avoid trouble with lingering passwords). In order to fully re-authenticate a team, the IP address needs to be unset. You might also want to generate a new password for this specific user. Furthermore, a team must explicitly connect to the team interface URL, because with an unknown IP, the root DOMjudge website will redirect to the public interface.

4.3.3 Using an external LDAP server

This method can be useful when you want to integrate DOMjudge into a larger system, or already have credentials on an LDAP server available. The `username` field in the database must contain the LDAP username of the DOMjudge user. Furthermore, in `etc/domserver-config.php` the `LDAP_*` configuration settings must be adapted to your setup. Note that multiple (backup) servers can be specified: they are queried in order to try to successfully authenticate. After successful authentication against the LDAP server(s), PHP sessions are used to track login into DOMjudge.

4.3.4 Fixed authentication

This method automatically authenticates each connection to the web interface as a fixed, configurable user. This can be useful for testing or demonstration purposes, but probably not for real use scenarios.

4.3.5 Adding new authentication methods

The authentication system is modular and adding new authentication methods is fairly easy. The authentication is handled in the file `lib/www/auth.php`. Adding a new method amounts to editing the functions in that file to handle your specific case.

4.4 Providing testdata

Testdata is used to judge the problems: when a submission run is given the input testdata, the resulting output is compared to the reference output data using a *compare script*. The default compare script simply checks if the outputs are equal up to whitespace differences, but more elaborate comparisons can be done, see e.g. the `float` and `boolfind_cmp` scripts.

The database has a separate table named testcase, which can be manipulated from the web interface. Under a problem, click on the testcase link. There the files can be uploaded. The judgehosts cache a copy based on MD5 sum, so if you need to make changes later, re-upload the data in the web interface and it will automatically be picked up.

Testdata can also be imported into the system from a problem zip file, following the Kattis problem package format .

4.5 Start the daemons

Once everything is configured, you can start the daemons. They all run as a normal user on the system. The needed root privileges are gained through sudo only when necessary.

- One or more judgedaemons: one on each judgehost (or optionally multiple per host; then the `-n X` option should be used to bind a judgedaemon to CPU X to prevent CPU resource conflicts).
- Optionally the balloon notification daemon (as an alternative to the web interface).

4.6 Check that everything works

If the daemons have started without any problems, you've come a long way! Now to check that you're ready for a contest.

First, go to the jury interface: `http(s)://yourhost.example.edu/domjudge/jury`. Look under all the menu items to see whether the displayed data looks sane. Use the config-checker under 'Admin Functions' for some sanity checks on your configuration.

Go to a team workstation and see if you can access the team page and if you can submit solutions.

Next, it is time to submit some test solutions. If you have the default Hello World problem enabled, you can submit some of the example sources from under the `doc/examples` directory. They should give 'CORRECT'.

You can also try some (or all) of the sources under `tests`. Use `make check` to submit a variety of tests; this should work when the submit client is available and the default example problems are in the active contest. There's also `make stress-test`, but be warned that these tests might crash a judgedaemon. The results can be checked in the web interface; each source file specifies the expected outcome with some explanations. For convenience, there is a link *judging verifier* in the admin web interface; this will automatically check whether submitted sources from the `tests` directory were judged as expected. Note that a few sources have multiple possible outcomes: these must be verified manually.

When all this worked, you're quite ready for a contest. Or at least, the practice session of a contest.

4.7 Testing jury solutions

Before running a real contest, you and/or the jury will want to test the jury's reference solutions on the system.

The simplest way to do this is to include the jury solutions in a problem zip file and upload this. You can also upload a zip file containing just solutions to an existing problem. Note that the zip archive has to adhere to the Kattis problem package format . For this to work, the jury/admin who uploads the problem has to have an associated team to which the solutions will be assigned. The solutions will automatically be judged if the contest is active (but it need not have started yet). You can verify whether the submissions gave the expected answer from the link on the jury/admin index page.

5 Team Workstations

Here's a quick checklist for configuring the team workstations. Of course, when hosting many teams, it makes sense to generate a preconfigured account that has these features and can be distributed over the workstations.

1. The central tool teams use to interact with DOMjudge is the web browser.
   - If possible, set the Home Page to `your.domjudge.location/team/`
   - Go to the team page and check if this team is correctly identified.
   - If using https and a self signed certificate, add this certificate to the browser certificate list to prevent annoying dialogs.
2. Make sure compilers for the supported languages are installed and working.
3. Provide teams with the command line submit client and check that it works.
   - If needed, set environment variables to configure the client.
   - Optionally distribute `.netrc` files with team credentials.
   - If using https and a self signed certificate, add this certificate to the local trust store (see 3.18.2 (HTTPS setup)).
4. Make the sample in- and output data from the problem set available.
5. Add your SSH key to their authorized_keys file, so you can always access their account for wiping and emergencies.
6. Check that internet access is blocked.

6 Web interface

The web interface is the main point of interaction with the system. Here you can view submissions coming in, control judging, view the standings and edit data.

6.1 Jury and Administrator view

The jury interface has two possible views: one for jury members, and one for DOMjudge administrators. The second view is the same as the jury view, but with more features added, and can be enabled by giving a user the 'admin' role (instead of or next to the 'jury' role).

This separation is handy as a matter of security (jury members cannot (accidentally) modify things that shouldn't be) and clarity (jury members are not confused / distracted by options they don't need).

Options offered to administrators only:

- Adding and editing any contest data
- Managing team passwords
- The config checker
- Refreshing the scoreboard & hostname caches
- Rejudge 'correct' submissions
- Restart 'pending' judgings

Furthermore, some quick link menu items might differ according to usefulness for jury or admins.

A note on rejudging: it is policy within the DOMjudge system that a correct solution cannot be reverted to incorrect. Therefore, administrator rights are required to rejudge correct or pending (hence, possibly correct) submissions. For some more details on rejudging, see the jury manual.

6.2 The scoreboard

The scoreboard is the canonical overview for anyone interested in the contest, be it jury, teams or the general public. It deserves to get a section of its own.

6.2.1 Colours and sorting

Each problem can be associated with a specific colour, e.g. the colour of the corresponding balloon that is handed out. DOMjudge can display this colour on the scoreboard, if you fill in the 'color' attribute in the 'problem' table; set it to a valid CSS colour value (e.g. 'green' or '#ff0000', although a name is preferred for displaying colour names).

It's possible to have different categories of teams participating, this is controlled through the 'team_category' table. Each category has its own background colour in the scoreboard. This colour can be set with the 'color' attribute to a valid CSS colour value.

If you wish, you can also define a sortorder in the category table. This is the first field that the scoreboard is sorted on. If you want regular teams to be sorted first, but after them you want to sort both spectator- and

business teams equally, you define '0' for the regular category and '1' for the other categories. To completely remove a category from the public (but not the jury) scoreboard, the category visible flag can be set to '0'.

6.2.2 Starting and ending

A contest can be selected for viewing after its activation time, but the scoreboard will only become visible to public and teams once the contest starts. Thus no data such as problems and teams is revealed before then.

When the contest ends, the scores will remain displayed until the deactivation time passes.

6.2.3 Freezing and defrosting

DOMjudge has the option to 'freeze' the public- and team scoreboards at some point during the contest. This means that scores are no longer updated and remain to be displayed as they were at the time of the freeze. This is often done to keep the last hour interesting for all. The scoreboard freeze time can be set with the 'freezetime' attribute in the contest table.

The scoreboard freezing works by looking at the time a submission is made. Therefore it's possible that submissions from (just) before the freezetime but judged after it can still cause updates to the public scoreboard. A rejudging during the freeze may also cause such updates.

If you do not set any freeze time, this option does nothing. If you set it, the public and team scoreboards will not be updated anymore once this time has arrived. The jury will however still see the actual scoreboard.

Once the contest is over, the scores are not directly 'unfrozen'. This is done to keep them secret until e.g. the prize ceremony. You can release the final scores to team and public interfaces when the time is right. You can do this either by setting a predefined 'unfreezetime' in the contest table, or you push the 'unfreeze now' button in the jury web interface, under contests.

6.2.4 Clickability

Almost every cell is clickable in the jury interface and gives detailed information relevant to that cell. This is (of course) not available in the team and public scoreboards, except that in the team and public interface the team name cell links to a page with some more information and optionally a team picture, and the problem header cells link to the problem text, if available.

6.2.5 Caching

The scoreboard is not recalculated on every page load, but rather cached in the database. It should be safe for repeated reloads from many clients. In exceptional situations (should never occur in normal operation, e.g. a bug in DOMjudge), the cache may become inaccurate. The jury administrator interface contains an option to recalculate a fresh version of the entire scoreboard. You should use this option only when actually necessary, since it puts quite a load on the database.

6.2.6 Exporting to an external website

In many cases you might want to create a copy of the scoreboard for external viewing from the internet. The command `bin/static_scoreboard` is provided just for that. It writes to stdout a version of the scoreboard with refresh meta-tags and links to team pages removed. This command can for example be run every minute and the output be placed as static content on a publicly reachable webserver.

6.3 Balloons

In many contests balloons are handed out to teams that solve a particular problem. DOMjudge can help in this process: both a web interface and a notification daemon are available to notify that a new balloon needs to be handed out. Note that only one should be used at a time.

The web based tool is reachable from the main page in the jury interface, where each balloon has to be checked off by the person handing it out.

For the daemon, set the `BALLOON_CMD` in `etc/domserver-config.php` to define how notifications are sent. Examples are to mail to a specific mailbox or to send prints to a printer. When configured, start `bin/balloons` and notification will start.

Notifications will stop as soon as the scoreboard is frozen. Enable the `show_balloons_postfreeze` configuration option to keep issuing balloon notifications after the freeze.

7 Security

This judging system was developed with security as one of the main goals in mind. To implement this rigorously in various aspects (restricting team access to others and the internet, restricting access to the submitted programs on the domjudge systems, etc...) requires root privileges to different parts of the whole contest environment. Also, security measures might depend on the environment. Therefore we have decided not to implement security measures which are not directly related to the judging system itself. We do have some suggestions on how you can setup external security.

7.1 Considerations

Security considerations for a programming contest are a bit different from those in normal conditions: normally users only have to be protected from deliberately harming each other. During a contest we also have to restrict users from cooperatively communicating, accessing restricted resources (like the internet) and restrict user programs running on judgehosts.

We expect that chances are small that people are trying to cheat during a programming contest: you have to hack the system and make use of that within very limited time. And you have to not get caught and disqualified afterwards. Therefore passive security measures of warning people of the consequences and only check (or probe) things might be enough.

However we wanted the system to be as secure as possible within reason. Furthermore this software is open source, so users can try to find weak spots before the contest.

7.2 Internal security

Internal security of the system relies on users not being able to get to any vital data (jury input/output and users' solutions). Data is stored in two places: in files on the DOMjudge system account and in the SQL database.

Files should be protected by restricting permission to the relevant directories.

Note: the database password is stored in `etc/dbpasswords.secret`. This file has to be non-readable to teams, but has to be readable to the web server to let the jury web interface work. A solution is to make it readable to a special group the web server runs as. This is done when using the default configuration and installation method and when `make install-{domserver,judgehost}` is run as root. The webserver group can be set with `configure -with-webserver-group=GROUP`; by default it is tried to be determined from groups available on the system, e.g. `www-data` or `apache`.

Judgehosts and the domserver communicate with each other over HTTP. Also all parties accessing the domserver web interface obviously use this protocol. We advise to setup HTTPS so interactions between domserver, judgehosts and teams are all protected. If you need to use a self-signed certificate, you can consider to install it on the team workstations beforehand to minimize hassle.

When using IP address authentication, one has to be careful that teams are not able to spoof their IP (for which they normally need root/administrator privileges), as they would then be able to view other teams' submission info (not their code) and clarifications and submit as that team. *Note:* This means that care has to be taken e.g. that teams cannot simply login onto one another's computer and spoof their identity.

Problem texts can be uploaded to DOMjudge. No filtering is performed there, so make sure they are from trusted sources to, in the case of HTML, prevent cross site scripting code to be injected.

7.3 Root privileges

A difficult issue is the securing of submitted programs run by the jury. We do not have any control over these sources and do not want to rely on checking them manually or filtering on things like system calls (which can be obscured and are different per language).

Therefore we decided to tackle this issue by running these programs in a environment as restrictive as possible. This is done by setting up a minimal chroot environment with Linux cgroup process control. For this, root privileges on the judgehosts and statically compiled programs are needed. By also limiting all kinds of system resources (memory, processes, time, unprivileged user and network access) we protect the system from programs which try to hack or could crash the system.

7.4 File system privileges

Of course you must make sure that the file system privileges are set such that there's no unauthorised access to sensitive data, like submitted solutions or passwords. This is quite system dependent. At least <`judgehost_judgedir`> should not be readable by other users than DOMjudge.

7.4.1 Permissions for the web server

The default installation sets permissions correctly for the web server user (commonly `www-data` or `apache`). The following information is for those who want to verify the setup or make modifications to the settings.

Care should be taken with the `etc` directory: the `domserver-{config,static}.php`, `dbpasswords.secret` and `restapi.secret` files should all be readable, but `dbpasswords.secret` and `restapi.secret` should not be readable by anyone else. This can be done for example by setting the `etc` directory to owner:group <DOMjudge account>:<Web server group> and permissions `drwxr-x--`, denying users other than yourself and the web server group access to the configuration and password files.

If you want the web server to also store incoming submission sources on the file system (next to the database), then <`domserver_submitdir`> must be writable for the web server, see also 3.4.2 (storage of submissions).

You should take care not to serve any files over the web that are not under the DOMjudge 'www/' directory, because they might contain sensitive data (e.g. those under `etc/`). DOMjudge comes with `.htaccess` files that try to prevent this, but double-check that it's not accessible.

7.5 External security

The following security issues are *not* handled by DOMjudge, but left to the administrator to set up.

Network traffic between team computers, domserver and the internet should be limited to what is allowed. Possible ways of enforcing this might be: monitor traffic, modify firewall rules on team computers or (what we implemented with great satisfaction) put all team computers behind a firewalling router.

Solutions are run within a restricted (chroot/cgroup) environment on the judgehosts which restricts outgoing network access.

A Common problems and their solutions

A.1 Setting up a pre-built chroot tree

With the default configuration, submitted programs are run within a minimal chroot environment. For this the programs have to be statically linked, because they do not have access to shared libraries.

To be able to run submissions of languages that don't support statically linked binaries (think interpreted languages and bytecode compiled languages, amongst which Java and C#), one can build a bigger chroot environment which contains all necessary ingredients to let the submissions run. DOMjudge supports this with some manual setup.

First of all, a chroot tree with those interpreters or runtimes needs to be created. The script `bin/dj_make_chroot` can create one from Debian or Ubuntu GNU/Linux sources. Add the required packages for the needed languages to the `INSTALLDEBS` variable in the script; run the script without arguments for basic usage information.

This pre-built chroot tree is not directly used as chroot environment for judgings. Instead, some subdirectories of this tree are bind-mounted into a chroot tree that is specifically created for each judging. The bind-mounting and a few other things are done by the script `lib/judge/chroot-startstop.sh`, when the pre-built tree is available.

Finally, if necessary edit the script `lib/judge/chroot-startstop.sh` and adapt it to work with your local system. In case you changed the default pre-built chroot directory, make sure to also update the sudo rules and the `CHROOTORIGINAL` variable in `chroot-startstop.sh`.

A.2 Java compilers

Java is difficult to deal with in an automatic way. It is probably most preferable to use OpenJDK or Oracle Java, because that's the version contestants will be used to. The GNU Compiler for Java (GCJ) is easier to deal with but may lack some features.

The recommended way of using Java is by setting up the chroot as in the previous section, but you can also choose one of the alternatives as described below.

1. As an alternative the `gcj` compiler from GNU can be used instead of Oracle's version. This one generates true machine code and can link statically. However a few function calls cannot be linked statically (see 'GCJ compiler warnings' in this FAQ). Secondly, the static library `libgcj.a` doesn't seem to be included in all GNU/Linux distributions: at least not in RedHat Enterprise Linux 4.
2. One can disable the chroot environment in `etc/judgehost-config.php` by disabling `USE_CHROOT`. Disabling the chroot environment removes this layer of security against submissions that attempt to cheat, but it is a simple solution to getting Java to work, for demo or testing purposes. **Note**: no guarantees about system security can be made when running a contest with chroot disabled!

A.3 The Java virtual machine (jvm) and memory limits

DOMjudge imposes memory limits on submitted solutions. These limits are imposed before the compiled submissions are started. On the other hand, the Java virtual machine is started via a compile-time generated script which is run as a wrapper around the program. This means that the memory limits imposed by DOMjudge are for the jvm and the running program within it. As the jvm uses approximately 300MB, this reduces the limit by this significant amount. See the `java_javac` and `java_javac_detect` compile executable scripts for the implementation details.

If you see error messages of the form

```
Error occurred during initialization of VM
java.lang.OutOfMemoryError: unable to create new native thread
```

or

```
Error occurred during initialization of VM
Could not reserve enough space for object heap
```

Then the problem is probably that the jvm needs more memory than what is reserved by the Java compile script. You should try to increase the `MEMRESERVED` variable in the java compile executable and check that the configuration variable `memory limit` is set larger than `MEMRESERVED`. If that does not help, you should try to increase the configuration variable `process limit` (since the JVM uses a lot of processes for garbage collection).

Note that (especially on x86_64 machines) the jvm seems to preallocate huge amounts of memory, up to 2 GB! This is not actually all used, but the memory restriction in DOMjudge will flag it as such, unless Linux cgroups are enabled, then the actual memory used is measured. Thus, we strongly recommend using Linux cgroups when using the Oracle jvm.

A.4 Java class naming

Java requires a specific naming of the main class. When declaring the main class `public`, the filename must match the class name. Therefore one should *not* declare the main class public; from experience however, many teams do so. Secondly, the Java compiler generates a bytecode file depending on the class name. There are two ways to handle this.

The simplest Java compile script `java_javac` requires the main class to be named `Main` with method

```
public static void main(String args[])
```

The alternative (and default) is to use the script `java_javac_detect`, which automatically detects the main class and even corrects the source filename when it is declared public.

When using the GNU gcj compiler, the same holds for the `java_gcj` script as for `java_javac`.

A.5 GCJ compiler warnings

When using the GNU GCJ compiler script `java_gcj` for compiling Java sources, it can give a whole lot of warning messages of the form

```
/usr/lib/gcc-lib/i386-linux/3.2.3/libgcj.a(gc_dlopen.o)(.text+0xbc):
In function 'GC_dlopen': Using 'dlopen' in statically linked
applications requires at runtime the shared libraries from the glibc
version used for linking
```

These are generated because you are trying to compile statically linked sources, but some functions can not be static, e.g. the 'dlopen' function above. These are *warnings* and can be safely ignored, because under normal programming contest conditions people are not allowed to use these functions anyway (and they are not accessible within the chroot-ed environment the program is run in).

A.6 Memory limit errors in the web interface

When uploading large testdata files, one can run into an error in the jury web interface of the form:

```
Fatal error: Allowed memory size of XX bytes exhausted (tried to
allocate YY bytes) in /home/domjudge/system/lib/lib.database.php
on line 154
```

This means that the PHP engine has run out of memory. The solution is to raise the memory limits for PHP. This can be done by either editing `etc/apache.conf` and raising the `memory_limit`, `upload_max_filesize` and `post_max_size` values to well above the size of your largest testcase. You can change these parameters under the jury directory or by directly editing the global Apache or `php.ini` configuration. Note also that `max_file_uploads` must be larger than the maximum number of testcases per problem to be able to upload and edit these in the web interface.

The optional PHP Suhosin module may also impose additional limits; check your error logging to see if these are triggered. You may also need to raise MySQL's `max_allowed_packet` parameter in `/etc/mysql/my.cnf` on both server and client.

A.7 Compiler errors: 'runguard: root privileges not dropped'

```
Compiling failed with exitcode 255, compiler output:
/home/domjudge/system/bin/runguard: root privileges not dropped
```

When the above error occurs on submitting any source, this indicates that you are running the `judgedaemon` as root user. You should not run any part of DOMjudge as root; the parts that require it will gain root by themselves through sudo. Either run it as yourself or, probably better, create dedicated a user `domjudge` under which to install and run everything.

Also do not confuse this with the `domjudge-run` user: this is a special user to run submissions as and should also not be used to run normal DOMjudge processes; this user is only for internal use.

A.8 found processes still running ... apport

```
error: found processes still running as 'domjudge-run', check manually:
2342 apport
```

The above error occurs on submitting segmentation fault solutions if you have apport installed (which is default on Ubuntu). Disable or uninstall the apport daemon on all judgehosts.

A.9 Enforcement of time limits

Time limits within DOMjudge are enforced primarily in CPU time, and secondly a more lax wall clock time limit is used to make sure that submissions cannot idle and hog judgedaemons. The way that time limits are calculated and passed through the system involves a number of steps, so documented here.

Time limits are set per problem in seconds. Each language in turn may define a time factor (defaulting to 1) that multiplies it to get a specific time limit for that problem/language combination. This is the *soft timelimit*. The configuration setting `timelimit overshoot` is then used to calculate a *hard timelimit*. This overshoot can be specified in terms of an absolute and relative margin.

The `soft:hard` timelimit pair is passed to `testcase_run.sh` and then on to `runguard` as both wall clock and CPU limit. Since the CPU option is passed second, this one is used by `runguard` when reporting whether the soft, actual timelimit has been surpassed. The submitted program gets killed when either the hard wall clock or CPU time has passed.

B Multi-site contests

This manual assumed you are running a singe-site contest; that is, the teams are located closely together, probably in a single physical location. In a multi-site or distributed contest, teams from several remote locations use the same DOMjudge installation. An example is a national contest where teams can participate at their local institution.

DOMjudge supports such a setup on the condition that a central installation of DOMjudge is used to which the teams connect over the internet. It is here where all submission processing and judging takes place. Because DOMjudge uses a web interface for all interactions, teams and judges will interface with the system just as if it were local. Still, there are some specific considerations for a multi-site contest.

Network: there must be a relatively reliable network connection between the locations and the central DOMjudge installation, because teams cannot submit or query the scoreboard if the network is down. Because of traversing an unsecured network, you may want to consider HTTPS for encrypting the traffic. If you want to limit internet access, it must be done in such a way that the remote DOMjudge installation can still be reached.

Team authentication: the IP-based authentication will still work as long as each team workstation has a different public IP address. If some teams are behind a NAT-router and thus all present themselves to DOMjudge with the same IP-address, another authentication scheme must be used (e.g. PHP sessions).

Judges: if the people reviewing the submissions will be located remotely as well, it's important to agree beforehand on who-does-what, using the submissions claim feature and how responding to incoming clarification requests is handled. Having a shared chat/IM channel may help when unexpected issues arise.

Scoreboard: by default DOMjudge presents all teams in the same scoreboard. Per-site scoreboards can be implemented either by using team categories or team affiliations in combination with the scoreboard filtering option.

C Developer information

This section contains instructions specifically for those wishing to modify the DOMjudge source. If you have any questions about developing DOMjudge, or if you want to share your changes that may be useful to others, please don't hesitate to contact us through our development mailing list .

C.1 Bootstrapping from Git repository sources

The installation steps in this document assume that you are using a downloaded tarball from the DOMjudge website. If you want to install from Git repository sources, because you want to use the bleeding edge code or consider to send a patch to the developers, the configure/build system first has to be bootstrapped.

This requires additional software to be installed:

- The GNU autoconf/automake toolset
- Composer - PHP Package Manager.
- Linuxdoc and groff to build the admin and judge documentation from SGML sources and a LaTeX installation to generate the PDF admin, judge and default team manual.

On Debian(-based) systems, the following apt-get command should install the additionally required packages (next to the 3.2 (standard set of packages)):

```
sudo apt install autoconf automake git composer
```

Composer is packaged since Debian Stretch and Ubuntu Xenial. Alternatively, it can be installed by following the documentation located here .

When this software is present, bootstrapping can be done by running `make dist`, which creates the `configure` script, downloads and installs the PHP dependencies via composer and generates documentation from SGML/LaTeX sources.

C.2 Maintainer mode installation

Besides the two modes of installation described in section 3.3 (Installation system), DOMjudge provides a special maintainer mode installation. This method does an in-place installation within the source tree. This allows one to immediately see effects when modifying code.

This method requires some special steps which can most easily be run via makefile rules as follows:

```
make maintainer-conf [CONFIGURE_FLAGS=<extra options for ./configure>]
make maintainer-install
```

Note that these targets have to be executed *separately* and they replace the steps described in the section 3.3 (Installation system); also no `-prefix` flag or other directories have to be specified to `configure`. In this case the binaries (e.g. `judgedaemon` and `dj_setup_database`) can be found in their respective source directories, and are also symlinked in `bin`.

C.3 Makefile structure

The Makefiles in the source tree use a recursion mechanism to run make targets within the relevant subdirectories. The recursion is handled by the `REC_TARGETS` and `SUBDIRS` variables and the recursion step is executed in `Makefile.global`. Any target added to the `REC_TARGETS` list will be recursively called in all directories in `SUBDIRS`. Moreover, a local variant of the target with `-l` appended is called after recursing into the subdirectories, so recursion is depth-first.

The targets `dist, clean, distclean, maintainer-clean` are recursive by default, which means that these call their local `-l` variants in all directories containing a Makefile. This allows for true depth-first traversal, which is necessary to correctly run the `*clean` targets: otherwise e.g. `paths.mk` will be deleted before subdirectory `*clean` targets are called that depend on information in it.