DOMjudge Administrator's Manual

by the DOMjudge team $Date: 2011-08-24 23:44:38 +0200 (Wed, 24 Aug 2011) $

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: $Rev: 3616 $

Contents

1 DOMjudge overview

DOMjudge is a system for running programming contests like the ACM 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 6-10). 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 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 more
- Detailed jury information (submissions, judgings) and options (rejudge, clarifications)
- Designed with security in mind
- Has been used in many live contests
- 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 and PHP-command line interface
- MySQL database server version 4.1.0 or newer
- Compilers for the languages you want to support

A 2.3 (detailed list of requirements) is contained in the 2 (Installation and Configuration) chapter.

1.3 Copyright and licencing

DOMjudge is developed by Jaap Eldering, Thijs Kinkhorst, Peter van de Werken and Tobias Werth. Development is hosted at Study Association A-Eskwadraat , Utrecht University , The Netherlands.

It is Copyright (c) 2004 - 2011 by The DOMjudge Developers.

DOMjudge, including its documentation, is free software; you can redistribute it and/or modify it under the terms of the *GNU General Public License* <http://www.gnu.org/copyleft/gpl.html> as published by the Free Software Foundation; either version 2, or (at your option) any later version. See the file COPYING.

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:

- dash (i386) is included, built from the Debian dash sources (copyright various people, see `doc/dash.copyright`).
- mkstemps.h and basename.h are modified versions from the GNU libiberty library (copyright Free Software Foundation).
- lib.database.php by Jeroen van Wolffelaar et al.
- submit.cc and submitdaemon.cc are based on submit.pl and submitdaemon.pl by Eelco Dolstra.
- runguard.c was originally based on timeout from The Coroner's Toolkit by Wietse Venema.
- sorttable.js by Stuart Langridge.
- jscolor.js by Jan Odvarko.
- The DOMjudge logo is based on the NKP 2004 logo made by Erik van Sebille.
- Several M4 autoconf macros from the Autoconf archive by various people are included under `m4/`.

1.3.1 Non-GPL licenced parts of DOMjudge

A binary version of the dash shell (statically compiled) is distributed with DOMjudge. This program is copyright by various people under the BSD licence and a part under the GNU GPL version 2, see `COPYING.BSD` and `doc/dash.copyright` for more details. Sources can be downloaded from:

<http://domjudge.sourceforge.net/sources/> .

The `sorttable.js` script is copyright by Stuart Langridge and licenced under the MIT/X11 licence, see `COPYING.MIT`. This software was downloaded from:

<http://www.kryogenix.org/code/browser/sorttable/> . jscolor.js is copyright Jan Odvarko and licenced under the GNU LGPL. It was obtained at <http://jscolor.com> .

The M4 autoconf macros are licenced under all-permissive and GPL3+ licences; see the respective files for details.

1.3.2 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 original 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, but Erik 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: http://domjudge.sourceforge.net/

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

The authors can be reached through the development mailing list: domjudge-devel@lists.a-eskwadraat.nl . See the list information page for details.

2 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.

2.1 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, set a root password for it and make it accessible from all judgehosts.
- 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 2.3.3 (an example command line for Debian GNU/Linux).

DOMjudge:

- Extract the source tarball and run `./configure [-enable-fhs] -prefix=<basepath>`.
- 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 `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 .../domserver/etc/apache.conf /etc/apache2/conf.d/domjudge.conf && sudo apache2ctl graceful`
- Check that the web interface works (/team, /public and /jury) and check that the jury interface is password protected. Optionally add (more) users to `etc/htpasswd-{jury,plugin}`.
- 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)

- Add to `/etc/sudoers`:
  `domjudge ALL=(root) NOPASSWD: <path-to-runguard>/runguard *`
  `domjudge ALL=(root) NOPASSWD: /bin/cp -pR /dev/null ./dev/null`
- Copy the file `etc/dbpasswords.secret` from the domserver to all judgehosts to synchronise database passwords.
- 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
./check-judgings
```

The `check-judgings` script automatically verifies most of the test sources, except for a few with multiple possible outcomes; these have to be verified by hand. Read the sources for a description of what should (not) happen. You may want to change the `AUTH_METHOD` to `FIXED` and set the environment variable `SUBMITBASEURL` to your DOMjudge base URL, e.g. `http://domjudge.example.com/`.

Optionally:

- Install the submit client on the team workstations.
- Generate one-time passwords for all the teams in the web interface.
- Further tighten the security of the system, e.g. by applying firewall rules.
- Start the balloon notification daemon: `cd bin; ./balloons`; or use the balloon web interface.
- Setup the Java chroot environment on the judgehosts to use Java with chroot:
  `bin/dj_make_chroot <chrootdir> <architecture>`
  `$EDITOR judge/chroot-startstop.sh`
  enable the `chroot-startstop.sh` script in `etc/judgehost-config.php` and add the following lines to `/etc/sudoers`:
  `domjudge ALL=(root) NOPASSWD: /bin/mount -n -t proc -bind /proc proc`
  `domjudge ALL=(root) NOPASSWD: /bin/umount /*/proc`
  `domjudge ALL=(root) NOPASSWD: /bin/mount -bind <chrootdir>/*`
  `domjudge ALL=(root) NOPASSWD: /bin/umount <judgehost_judgedir>/*`
- For additional features in the jury web interface, the following PHP extensions can be installed:
  - GeSHI or the PEAR Text_Highlighter class for source syntax highlighting;
  - xdiff PECL extension for diffs between submissions;
  - zip PHP-bundled extension (`-enable-zip`) for uploading problem data as zip-bundles (enabled by default in Debian, but not in all other Linux distributions).

2.2 Concepts

This manual assumes you are aware of some of the concepts used within DOMjudge. Here's an overview.

DOMjudge discerns three different 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 Apache, and MySQL. Optionally these tasks can be further split out to separate machines, but that's normally not necessary and not supported out of the box.

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. Since this is computationally intensive, there should ideally be at least a couple of these. They will run the judgedaemon from DOMjudge. For security and performance reasons it is highly recommended not to use the server as a judgehost.

Note that the judges (persons) are not required and not recommended to work on any of the DOMjudge server or judgehosts. They can just access the system via the jury web interface and working e.g. on judgehosts can interfere with system stability.

2.3 Requirements

2.3.1 System requirements

The requirements for the deployment of DOMjudge are:

- Computers for the domserver and judgehosts must run Linux or a Unix variant. This software has been developed mostly under Debian GNU/Linux and has been tested a bit under other Linux distributions and FreeBSD. We try to adhere to POSIX standards.
- (Local) root access on the jury computers for configuring sudo, installing some files with restricted permissions and for (un)mounting the proc file system when using Oracle (previously Sun) Java. See 6.3 (Security: root privileges) for more details.
- A TCP/IP network which connects all jury 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 6.5 (Security: external security) for more details. TCP/IP networking is used in a few different ways:
  - The judgehosts use TCP/IP connections to connect to the MySQL database on port 3306.
  - HTTP traffic from teams, the public and jury to the web server, port 80 or 443.
  - The 'submit' command line client connects to the web server also via HTTP.

  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 3.3 (Contest setup: team authentication) for more details.

2.3.2 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 >= 5.0.0 and the mysql extension for PHP. We also recommend the posix extension for extra debugging information.
- MySQL >= 4.1.x database and client software
- PHP >= 5.0.0 command line interface and the mysql extension.
- 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 IA32)
- 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
- sudo to gain root privileges
- Apache htpasswd
- xsltproc

  from the GNOME XSLT library package.
- 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.

- 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 judgehosts in sync)
- libcurl (to use the command line submit client with the web interface)
- libmagic (for command line submit client to detect binary file submissions)
- GeSHi or PEAR Text_Highlighter class (to use syntax highlighting in the Show Source section of the jury interface)
- PECL xdiff extension (to reliably make diffs between submissions, DOMjudge will try alternative approaches if it's not available)
- PHP zip Extension (to upload problem data via zip bundles)
- beep for audible notification of errors, submissions and judgings, when using the default `alert` script.

Software required for building DOMjudge from distributed sources.

- 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

- The Boost regular expression library and the GNU Multiple Precision library to build the `checktestdata` program for advanced checking of input/output data correctness. These are optional and can be disabled with the configure option `-disable-checktestdata`.

Additional software required for building DOMjudge from a Subversion checkout.

- The GNU autoconf/automake toolset
- Flex and bisonc++ for generating the parsing code of the optional `checktestdata` script.
- Linuxdoc and Xfig/transfig to build the admin and judge documentation from SGML sources and a LaTeX installation to generate the PDF admin, judge and default team manual.

2.3.3 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, with the exception of notification of new clarifications in the menu bar. That can be updated without reloading the page by using AJAX. This is supported by any reasonably current browser with JavaScript enabled.

2.3.4 Debian installation command

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.

```
apt-get install gcc g++ make libcurl4-gnutls-dev mysql-server \
        apache2 php5 php5-cli libapache2-mod-php5 php5-mysql php-geshi \
        ntp sudo procps sharutils \
        phpmyadmin xsltproc libboost-regex-dev libgmp3-dev \
        linuxdoc-tools transfig texlive-latex-recommended texlive-latex-extra
```

On a judgehost, the following should be sufficient. The last line shows some example compilers to install for C, C++, Java (GNU), Java (Oracle/Sun), Haskell and Pascal; change the list as appropriate.

```
apt-get install make sudo php5-cli php5-mysql ntp xsltproc procps sharutils \
        gcc g++ gcj openjdk-6-jre-headless ghc fp-compiler
```

2.4 Installation system

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.

When installing from a Subversion checkout, the configure/build system first has to be bootstrapped. This can be done by running `make dist`, which creates the `configure` script and generates documentation from SGML/LaTeX sources. Note that this requires additional software as specified in the 2.3 (software requirements).

There are three different methods for installing DOMjudge:

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
```

This will install each of the `domserver, judgehost, docs` parts in a subdirectory `$HOME/domjudge/domserver` etc. Note that 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. The system needs root access to this directory! (for chroot and mounting of proc-fs).

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, /var/log, /tmp, /etc/domjudge`.

Maintainer install

The last installation method is meant for developers/maintainers of DOMjudge and 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 the latter requests root privileges via `su`.

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 && make install-docs
```

Note that even when installing e.g. in your own home directory, root privileges are still required for domserver and judgehost installation, because user and group ownership of password files, some directories and to give sudo access to `runguard`. One should *not* run DOMjudge programs and daemons under the root user however, but under a normal user: `runguard` is specifically designed to be the only part invoked as root (through sudo) to make this unnecessary and running as root will give rise to problems, see A.8 (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.

2.4.1 Makefile structure

The following information is meant for developers or other people who want to make changes to the sources.

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.

2.5 Configuration

Configuration of the judge system is mostly done by editing the configuration file(s) in `etc`: `domserver-config.php, judgehost-config.php, common-config.php` for the configuration options of the domserver, judgehost and shared configuration options respectively. The latter should be synchronised between domserver and judgehosts. Descriptions of settings are included in these files.

Besides these settings, there are a few other places where changes can be made to the system, see 2.9 (other configurable scripts).

2.6 Configuration of languages

Configuration of the compilers of the supported languages should be done separately. For each supported language a shell-script named `compile_<lang>.sh` should be created and placed in `lib/judge` on the judgehosts, where <lang> is the ID of the language as specified in the database. For more information, see for example `compile_c.sh`, and `compile.sh` in `lib/judge` for syntax. Note that compile scripts are included for the most common languages already.

Interpreted languages and non-statically linked binaries can in principle also be used, but then the option `USE_CHROOT` should be disabled (or all dependencies be added to the chroot environment). 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 (Sun) Java), a script must be generated during the compilation step, which will function as the executable: the script must run the interpreter on the source. See `compile_perl.sh` and `compile_java_javac.sh` in `lib/judge` for examples.

DOMjudge supports the use of Oracle (Sun) Java within a chroot environment. For this, a chroot environment which includes the Java libraries must first be built. This can be accomplished with the included script

`dj_make_chroot`: run this as root and pass as arguments the target directory to build the chroot environment in and as second argument the target machine architecture. Start the script without arguments for usage information. See also sections 2.14 (Installation of a judgehost) and A.1 (Problems: Java & chroot).

2.7 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 and comparison of the results. These can be specialised and adapted to the requirements per problem. For this, one has to create programs or scripts named `run_<tag>` and/or `compare_<tag>` in the `lib/judge` directory, see `run` and `compare` for examples and usage information. Then the `<tag>` must be specified in the `special_run` and/or `special_compare` fields of the problem (empty means that the default script run and compare scripts should be used). To simplify the use of custom run and compare programs, DOMjudge comes with wrapper scripts that handle the tedious, standard part. In most cases it will probably be convenient to use these, see `run_wrapper` and `compare_wrapper` for details, and the usage explanations below.

2.7.1 Compare programs

Implementing a special compare program, also called a *validator*, can be done in two ways: either write a program that is called directly (by `testcase_run.sh`) or use a copy of the `compare_wrapper` script. In the first case, the compare program must adhere to the C (ICPC validator interface). In the latter case, the wrapper generates the XML result file and handles redirection of input/output for you. Use this wrapper by copying it to `compare_<tag>` and let the jury write a checker program which can be called as

```
check_<tag> <testdata.in> <program.out> <testdata.out>
```

This program should write some kind of difference to stdout. No output results in a *correct* verdict and a nonzero exitcode in an internal (system) error. See as an example the included program `check_float`, which compares floating point numbers. The name of the check program and any parameters can also be modified in the `compare_wrapper` script.

For example, to compare output while ignoring DOS/UNIX newline differences, one can copy `compare_wrapper` to `compare_dos_newline_OK` and in that file set the variable `CHECK_PROGRAM="‘which diff‘"` and replace the line

```
"$CHECK_PROGRAM" $CHECK_OPTIONS "$TESTIN" "$PROGRAM" "$TESTOUT" > "$DIFFOUT"
```

by the lines

```
sed -i 's/\r$//' "$TESTOUT"
sed 's/\r$//' "$PROGRAM" | $CHECK_PROGRAM -a - "$TESTOUT" > "$DIFFOUT"
```

Note that these commands will modify the local copy of the jury testdata, but the original output generated by the team's solution is retained, and a plain diff output is generated. Next, for each problem that you want to use this validator for, set the `special_compare` field to `dos_newline_OK`. As an alternative to this modified validator script, one can accept presentation errors as correct answers by uncommenting the line

```
        'presentation-error' => 'correct',
```

in the `RESULTS_REMAP` array in the file `etc/judgehost-config.php`.

For more details on modifying validator scripts, see the comments at the top of the files `testcase_run.sh`, `compare_wrapper` and (when not using the wrapper) the appendix on the C (ICPC validator interface).

DOMjudge supports a `presentation-error` result. The default `compare` program returns this result when output only differs by whitespace; this is counted as an incorrect submission. The script `compare_wrapper` does not support presentation error results however. By default presentation errors are remapped to wrong answer; this can be changed in `etc/judgehost-config.php`.

2.7.2 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_<tag>` yourself, or use the provided wrapper script, which handles bi-directional communication between a jury program and the contestants' program on stdin/stdout.

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 or symlink it to another name `run_<tag>` and the jury must write a program named exactly `runjury_<tag>`, accepting the calling syntax

```
runjury_<tag> <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.

2.8 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`.

2.9 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.
- Optional scripts `submit/submit_copy.sh` and `lib/judge/chroot-startstop.sh` can be modified to suit your local environment. See comments in those files for more information.

2.10 Submission methods

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 can send submissions by either using the web interface internally (*http* protocol, the default), or using a special command line submit protocol, called *Dolstra*. The latter has some special features but is not usually needed. See D (Submitdaemon and the Dolstra protocol) for details on this.

Using the http protocol with the submit client requires the libcURL library development files at compile time (the submit client is statically linked to libcURL to avoid a runtime dependency).

The database is the authoritative version for submission sources; file system storage is available as an easy way to access the source files and as backup. The program `bin/restore_sources2db` is available to recover the submission table in the database from these files. The command line daemon will automatically store sources on the file system; the web server needs write permissions on <`domjudge_submitdir`> and ignores file system storage if these permissions are not set.

2.11 Database installation

DOMjudge uses a MySQL database server for information storage.

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 for the `jury` user 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 [-u <admin_user>] [-p <password>|-r] install
```

to create the database, users and insert some default/example data into the domjudge database. The option `-r` will prompt for a password; 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 3.1 (database section of Contest setup) for details.

2.11.1 Fine tuning settings

For Apache, there are countless documents on how to maximise 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.

As for PHP, the use of an opcode cache like the Alternative PHP Cache (Debian package: `php-apc`) is beneficial for performance.

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.
- `skip-networking` or `bind-address`: By default MySQL only listens on a local socket, but judgehosts need to connect remotely to it. When enabling remote connections, you may want to limit it to only the IP's of judgehosts in the MySQL user configuration (or with firewall rules).
- 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* (formerly *Heap*) table for the scoreboard_public and scoreboard_jury tables. They will be lost in case of a full crash, but can be recalculated from the jury interface.

2.11.2 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 use 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.

2.12 Web server configuration

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

You should turn PHP's `magic_quotes_*` options off. We also recommend to turn off `register_globals`. If you want to upload large testcases in the jury web interface, it may be necessary to raise some PHP limits or you'll get "memory exhausted" errors. Make sure that the parameters `memory_limit`, `upload_max_filesize` and `post_max_size` in `php.ini` are all well above the size of your largest testcase.

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. The Apache configuration snippet by default includes HTTP basic-auth authentication to the jury and plugin interfaces. A default user `domjudge_jury` with password equal to that in `etc/dbpasswords.secret` is set for the jury interface. Additional users can be added with the `htpasswd` program to either file `etc/htpasswd-{jury,plugin}`. Reload the web server for changes to take effect.

See also section 6.4.1 (Security: webserver privileges) for some details on file permissions for the `etc/dbpasswords.secret` and `etc/htpasswd-{jury,plugin}` files.

2.13 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_ERROR` 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.

2.14 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 has 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 'domjudge-run' with minimal privileges.

On RedHat:

```
useradd -d /nonexistent -g nobody -M -n -s /bin/false domjudge-run
```

On Debian:

```
useradd -d /nonexistent -g nogroup -s /bin/false 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`.

`Runguard` needs to be able to become root for certain operations like changing to the runuser and performing a chroot. The following line must be added to `/etc/sudoers`, replacing <domjudge> with the user you intend to run the judgedaemon as, and the full path to runguard.

```
domjudge ALL=(root) NOPASSWD: <path-to-runguard>/runguard *
domjudge ALL=(root) NOPASSWD: /bin/cp -pR /dev/null ./dev/null
```

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. 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 use the default `chroot-startstop.sh` script, then the following lines must be added to `/etc/sudoers`:

```
domjudge ALL=(root) NOPASSWD: /bin/mount -n -t proc --bind /proc proc
domjudge ALL=(root) NOPASSWD: /bin/umount /*/proc
domjudge ALL=(root) NOPASSWD: /bin/mount --bind <chrootdir>/*
domjudge ALL=(root) NOPASSWD: /bin/umount <judgehost_judgedir>/*
```

Here the user `domjudge` must be replaced by the user you intend to run the judgedaemon as, <`chrootdir`> by the path the chroot environment was installed to and <`judgehost_judgedir`> by the value specified in `configure`.

2.15 Building and installing the submit client

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 address 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.

2.15.1 The submit client under Windows/Cygwin

The submit client can also be built under Windows when the Cygwin environment is installed. First the Cygwin *setup.exe* <http://cygwin.com/setup.exe> program must be downloaded and installed with GCC, curl-devel and maybe some more packages included.

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.

2.16 (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 option `-w WEBBASEURI` can be passed to set the base URI of the DOMjudge webinterface; it should end with a slash and defaults to `http://example.com/domjudge/`. The following should do it on a Debian-like system:

```
sudo apt-get install make transfig texlive-latex-extra texlive-latex-recommended
cd .../docs/team
./genteammanual [-w http://your.location.example.com/domjudge/] [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 transfig texlive-latex-recommended
make -C doc/admin docs
make -C doc/judge docs
```

2.17 Optional features

2.17.1 Source code syntax highlighting

To support coloured display of submitted source code in the jury interface, two external classes of syntax highlighters are supported:

GeSHi <http://qbnz.com/highlighter> and the

PEAR <http://pear.php.net>

Text_Highlighter class <http://pear.php.net/package/Text_Highlighter/> . DOMjudge tries to find either of those in your PHP include path. When none are found, DOMjudge falls back to source code display without highlighting.

GeSHi

If you run a Debian-like system, you can simply install the `php-geshi` package. If not, download GeSHi and place `geshi.php` and the `geshi/` directory in DOMjudge's `lib/www` directory on the server.

PEAR Text Highlighter

You can install the Text Highlighter system wide with the PEAR-provided tools, like this: `pear install Text_Highlighter`.

Alternatively you can download the source code from the Text_Highlighter website and unpack that under the `lib/www` directory on the domserver. Rename the resulting `Text_Highlighter-x.y.z` directory to just `Text`.

2.17.2 NTP time synchronisation

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

2.17.3 The plugin web interface

Next to the public, team and jury web interfaces, DOMjudge also provides a *plugin* web interface. This web interface is still in development so subject to change. The interface provides contest data from DOMjudge in XML format and is meant to provide external programs (plugins) with data on the contest. This allows for all kinds of extensions beyond the core functionality of DOMjudge such as providing a fancy scoreboard with more statistics, aggregation of scoreboard data for a final presentation during the prize ceremony.

As we are still thinking about possible uses and thus the data to be provided, the exact specification of this interface may change. Also, we are especially interested in feedback and ideas.

There are currently two data-sets provided within the `plugin` subdirectory of the DOMjudge web interface, both in XML format:

`scoreboard.php`

: This page provides a representation of the scoreboard. Additionally it includes legend tables for problems, languages, affiliations and team categories. It does not accept any arguments.

`event.php`

: This page provides a representation of events that happened during the contest, including submissions, judgings, contest state changes and general clarifications. This page accepts two arguments `fromid` and `toid` to limit the output to events with event ID in that range.

See these pages or the accompanying `xsd`-files for the exact structure.

2.18 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-database-setup upgrade`.

3 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.

3.1 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:

	clarification	Clarification requests/replies are stored here.
x	configuration	Runtime configuration settings.
x	contest	Contest definitions with start/end time.
	event	Log of events during contests.
x	judgehost	Computers (hostnames) that function as 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, corresponding contest, etc.).
	submission	Submissions of solutions to problems.
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.
	scoreboard_jury	Cache of the scoreboards for public/teams and for the jury
	scoreboard_public	separately, because of possibility of score freezing.

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 `language`, `problem`, `team`, and `team_affiliation` 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 and is work in progress. These entries are simply stored as `name, value` pairs.

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.

`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. For a detailed treating of these, see section 3.2 (Contest milestones).

The `enabled` field can be unset to allow for easier editing of contest times, as disabled contests are not checked to overlap with other contests. A disabled contest will also not become active.

judgehost

List here the hosts that will be judging the submissions. `hostname` is the (short) hostname of a judge computer. `active` indicates whether this host should judge incoming submissions. `polltime` is an internally used variable to detect whether a judgedaemon is running on the host.

language

Programming languages in which to accept and judge submissions. `langid` is a string of maximum length 8, which references the language; it is used internally as extension for source files and must match the first extension listed for the language in the `LANG_EXTS` setting in the configuration files. This reference is also used to call the correct compile script (`lib/judge/compile_c.sh`, etc.), so when adding a new language, check that these match.

`name` is the displayed name of the language; `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. For example Java is/was known to be structurally slower than C/C++.

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).

`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 5.2.1 (Scoreboard: colours).

team

Table of teams: `login` is the account/login-name of the team (which is referenced to in other tables as `teamid`) and `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.

`authtoken` is a generic field used by several of the supported authentication mechanisms to store a piece of information it needs to identify the team. The content of the field for each of the mechanisms is:

- IPADDRESS: field contains the IP address of the team's workstation
- PHP_SESSIONS: contains a hash of the password that the team can log in with
- LDAP: contains the LDAP name (e.g. CN) corresponding to this DOMjudge user

`members` are the names of the team members, separated by newlines and `room` is the room the team is located, 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.

team_affiliation

`affilid` is the reference ID and `name` the name of the institution. `country` should be the 2 character ISO 3166-1 alpha-2 abbreviation of the country and `comments` is a free form field that is displayed in the jury interface.

Both for the country and the affiliation, a logo can be displayed on the scoreboard. For this to work, the `affilid` must match a logo picture located in `www/images/affiliations/<affilid>.png` and `country` must match a (flag) picture in `www/images/countries/<country>.png`. All country flags are present there, named with their 2-character ISO codes. See also `www/images/countries/README`. If either file is not present the respective ID string will be printed instead.

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 5.2.1 (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 `md5sum_input`, `md5sum_output` their respective md5 hashes to check for up-to-date-ness of cached versions by the judgehosts. `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 3.4 (providing testdata).

3.2 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 (potentially replacing a previous contest that was displayed before). 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 be stored but not processed; unjudged submissions received before endtime will still be judged.

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 5.2.3 (Scoreboard: freezing and defrosting).

The scoreboard, results and clarifications will remain to be displayed to team and public after a contest, until an activatetime of a later contest passes.

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). No two contests may have overlap: there's always at most one active contest at any time.

3.3 Team authentication

The authentication system lets domserver know which team it is dealing with. This system is modular, allowing flexible addition of new methods, if required. The following methods are available by default for team authentication.

3.3.1 PHP session with passwords (default)

Each team receives a password and PHP's session management is used to keep track of which team is logged in. This method is easiest to setup. It does require the administrator to generate 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 `authtoken` field in database (team table).

3.3.2 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 team with that IP listed in the team 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. Furthermore, with this method the command line submitclient can be used next to the web interface submit.

There are three 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 team's entry in the `authtoken` 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 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 team. Furthermore, a team must explicitly connect to the team interface, because with an unknown IP, the root DOMjudge website will redirect to the public interface.

Set IP upon first submission

This is only possible with the D (Dolstra protocol). The advantage is that no prior mapping needs to be configured, but the disadvantage is that the team interface cannot be viewed until at least one submission was made; there are also more constraints on the system. See the section on the Dolstra protocol for details.

The `authtoken` field in the database contains either the IP-address, or an MD5 hash of the one-time password if this was set and the team has not authenticated yet.

3.3.3 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 `authtoken` field in the database must contain the LDAP username of the DOMjudge team. 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.

3.3.4 Fixed team authentication

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

3.3.5 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.team.php`. Adding a new method amounts to editing the functions in that file to handle your specific case.

3.4 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. If they match exactly, the problem is judged to be correct. For problems with a special compare script, testdata should still be provided in the same way, but the correctness depends on the output of the custom compare script. Please check the documentation in `judge/compare_wrapper` when using this feature.

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 zip-bundle on each problem webpage. Each pair of files `<path-to-file>/<filename>.in` and corresponding `*.out` found in the zip-bundle will be added as testdata. Furthermore, when the file `domjudge-problem.ini` exists, then problem properties are read from that file in INI-syntax. All keys from the problem table are supported, so an example contents could be:

```
probid = hello

name = Hello world!
allow_submit=false
color=blue
```

Testcases will be added to those already present and imported properties will overwrite those in the database. A completely new problem can also be imported from a zip-bundle on the problems overview webpage; in that case, note that if the file `domjudge-problem.ini` is not present, a default value is chosen for the unmodifiable primary key `probid` (as well as for the other keys).

3.5 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;
- Optionally the balloon notification daemon.

3.6 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://www.your-domjudge-location/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 also a script `check-judgings`; 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.

3.7 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.

There is no special feature for testing their solutions under DOMjudge. The simplest approach is to submit these solutions as a special team. This method requires a few steps and some carefulness to prevent a possible information leak of the problemset. It is assumed that you have completely configured the system and contest and that all testdata is provided. To submit the jury solutions the following steps have to be taken:

- change the contest time to make the contest currently active;
- setup a special team at a local computer;
- submit the jury solutions as that team;
- check that all solutions are judged as expected in the jury interface;
- revert the contest to the original times.

Note that while the contest time is changed to the current time, anyone might be able to access the public or team web-interface: there's not too much there, but on the scoreboard the number of problems and their titles can be read. To prevent this information leak, one could disconnect the DOMjudge server, judgehosts and the computer used for submitting from the rest of the network.

Furthermore, you should make sure that the team you submit the solutions as, is in a category which is set to invisible, so that it doesn't show up on the public and team scoreboard. The sample team "DOMjudge" could be used, as it is in the "Organisation" category, which is not visible by default.

4 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.
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.

5 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.

5.1 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. Which to show is decided by using the HTTP authentication login used to access the web interface; you can list which HTTP users are admin with the variable `DOMJUDGE_ADMINS` in `etc/domserver-config.php`.

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.

5.2 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.

5.2.1 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'.

5.2.2 Starting and ending

The displayed scoreboard will always be that of the most recently started contest. The scoreboard is never displayed for a contest that still has to start. In other words, the scores will become visible on the first second of a contest start time.

When the contest ends, the scores will remain to be displayed, until a next contest starts.

5.2.3 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 automatically '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 scores now' button in the jury web interface, under contests.

5.2.4 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.

5.2.5 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.

5.2.6 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.

5.3 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 `bin/balloons` 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 continue even after the scoreboard is frozen, although a warning is printed on the notification. Stop the balloons daemon when you don't want balloons to be handed out anymore.

6 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 jury computers, 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.

6.1 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 jury computers.

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 will probably 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.

6.2 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: files on the jury account and in the SQL database. Files should be protected by preventing permission to the relevant directories. Furthermore, the (jury) web interface offers a view and allows modification of a lot of sensitive data.

Database access is protected by passwords. The default permissions allow connections from *all* hosts, so make sure you restrict this appropriately or choose strong enough passwords.

Note: database passwords are 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` which defaults to `www-data`.

The jury web interface is protected by HTTP Authentication. These credentials are essentially sent plain-text, so we advise to setup HTTPS at least for the jury interface, but preferably for all web interfaces. By default the `domjudge_jury` user will be given full access. You can choose to add more users to the file `etc/htpasswd-jury`. In `etc/domserver-config.php` you can add these users to the list `DOMJUDGE_ADMINS`. Most data-modification functions are restricted to only users in this list. See also the judge manual for some more details.

Secondly, the submitted sources should not be interceptable by other teams (even though that, if these would be sent clear-text, a team would normally need to be root/administrator on their computer to intercept this).

This can be accomplished by using HTTPS for the web interface. The D (Dolstra submission method) by default uses SSH to send files over the network.

There are multiple authentication methods for teams, each having its own issues to check for.

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.

When using PHP sessions or LDAP, authentication data is sent via HTTP, so we strongly advise to use HTTPS in that case.

6.3 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. For this, root privileges on the judging computers and statically compiled programs are needed. By also limiting all kinds of system resources (memory, processes, time, unprivileged user) we protect the system from programs which try to hack or could crash the system. However, a chroot environment does not restrict network access, so there lies a possible security risk that has to be handled separately.

6.4 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.

6.4.1 Permissions for the web server

Make sure that the web server serving the DOMjudge web interface pages has correct permissions to the `www, lib, etc` directory trees. The `www` and `lib` trees can safely set to be readable and accessible. Care should be taken with the `etc` dir: the `domserver-{config,static}.php`, `htpasswd-*` and `dbpasswords.secret` files should all be readable, but `dbpasswords.secret` and the htpasswd files 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 2.10 (submission methods).

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.

6.5 External security

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

Network traffic between team- and jury-computers 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) environment on the judge computers. This however does not restrict network access, so a team could try to send in a solution that tries to send input testdata back to them, access the internet, etc... A solution to this problem is to disallow all network traffic for the test user on the judge computers. On Linux, this can be accomplished by modifying the iptables, adding a rule like:

```
iptables -I OUTPUT -o <network_interface> -m owner --uid-owner <testuser_uid> -j REJECT
```

A Common problems and their solutions

A.1 Java compilers and the chroot

Java is difficult to deal with in an automatic way. It is probably most preferable to use Oracle (previously Sun) 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.

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.

For most languages compilers support this, but for Java, this is a bit problematic. The Oracle (Sun) Java compiler 'javac' is not a real compiler: a bytecode interpreter 'java' is needed to run the binaries and thus this cannot simply run in a chroot environment.

There are some options to support Java as a language:

1. One can disable the chroot environment in `etc/judgehost-config.php` by disabling `USE_CHROOT`. Disabling the chroot environment removes this extra layer of security against submissions that attempt to cheat, but it is a simple solution to getting Java to work.

2. Next, one can build a bigger chroot environment which contains all necessary ingredients to let Oracle (Sun) Java work within it. DOMjudge supports this with some manual setup.

   First of all, a chroot tree with Java support must be created. The script `bin/dj_make_chroot` creates one from Debian GNU/Linux sources; run that script without arguments for basic usage information. Next, edit the script `lib/judge/chroot-startstop.sh` and adapt it to work with your local system and uncomment the script in `etc/judgehost-config.php`.

3. As an alternative the `gcj` compiler from GNU can be used instead of Oracle's (Sun'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.

A.2 The Oracle (Sun) 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 Oracle (Sun) jvm 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 `judge/compile_java_javac.sh` 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 `judge/compile_java.sh` and check that the total memory limit `MEMLIMIT` in `etc/judgehost-config.php` is larger than `MEMRESERVED`.

A.3 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 `compile_java_javac.sh` 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 `compile_java_javac_detect.sh`, 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 and two similar scripts `compile_java_gcj.sh` and `compile_java_gcj_detect.sh` are available.

A.4 GCJ compiler warnings

When using the GNU GCJ compiler 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).

To filter these warnings, take a look at `judge/compile_java_gcjmod.sh` and replace or symlink `judge/compile_java.sh` by/to this file.

A.5 Error: 'submit_copy.sh failed with exitcode XX'

This error can have various causes. First of all: check the `submit.log` file for more complete error messages.

Assuming the default configuration where submit_copy.sh uses 'scp', we have found that shell initialisation scripts might contain statements which generate errors: scp runs the user's default shell when copying submission files and when the shell dies (e.g. because of not having a terminal), the copying fails.

Another cause might be that you do not have automatic access to the team's account (e.g. using ssh keys).

A.6 C#/mono support

Using the mono compiler and runtime for C# gives rise to similar problems as with Java. Although the C# language has been added to DOMjudge, there's no support yet to run it within a chroot environment. So in that case, `USE_CHROOT` must be disabled.

A.7 Memory limit errors in the web interface

E.g. 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 under the jury directory or by directly editing the global Apache or `php.ini` configuration.

A.8 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.

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 travelling 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 rankings are not currently possible.

C DOMjudge and the ICPC validator interface standard

DOMjudge supports the ICPC validator interface standard, which can be found at: <http://www.ecs.csus.edu/pc2/doc/valistandard.html>

As short summary, this interface standard consists of two parts: the invocation and the result interface. The invocation interface specifies that a validator must be called as a separate executable with at least four command line parameters:

```
/path/to/validator <input_data> <program_output> <reference_output> <result_file> [<extra_options>..
```

The result interface specifies that `result_file` should be a valid XML document containing a root element

```
<result outcome="string1"> string2 </result>
```

where `string1` is the result reported to the judging system and a value "accepted" indicates a correct result.

The invocation code (`judge/testcase_run.sh`) adheres to the invocation interface. It passes as a 5th optional parameter to the validator program the filename in which it expects a difference output between the program and jury output (parameters 2 and 3 respectively).

Parsing of the result XML file (in the result interface) is done with the 'xsltproc' program, which is part of the

GNOME libxslt package <http://www.xmlsoft.org/XSLT/> . *The exitcode of the validator program should be zero, otherwise an internal error is generated.*

DOMjudge currently has two validator scripts: `judge/compare` and `judge/compare_wrapper`. The first does a compare with a plain diff, the second script calls an external program for checking (e.g. `judge/check_float` for comparison of floating point results). When passed a 5th parameter, this is interpreted as a filename to which these scripts will write a comparison of the program and jury output. Both scripts also generate XML compliant output, which is written to the result file specified in parameter 4 and fully complies with the validator standard.

D Submitdaemon and the Dolstra protocol

In the default situation, teams can submit their solutions either via browsing to the web interface, or by using the command line submit client, which behind the scenes employs the same web interface to actually make the submission. This setup suffices for many environments.

The Dolstra protocol is different in that it uses a submitdaemon running on the domserver. One advantage is that submissions can be made before the IP address of the team is known. This authentication is fortified by the following process. When a client connects, it does not send the submission file, but only a reference to a randomised and not publicly visible file. This file is then copied from server side with the submit_copy script. This makes it impossible for teams to spoof a submission for a different team: the server 'calls back' the team the submitter identified himself as and checks for existence of the advertised file. Because filenames are randomised and invisible (within the `$HOME/.domjudge` directory by default), it is also impossible for someone to guess another team's filename and submit it for them.

The figure below is a graphical representation of the flow of a submission. Arrows with filled lines indicate the flow of the submission file, while dot-dash lines indicate flow of metadata about the submission. Each line where no protocol of data transfer is given, are just file system operations. Squares are programs and rounded squares are storage locations.

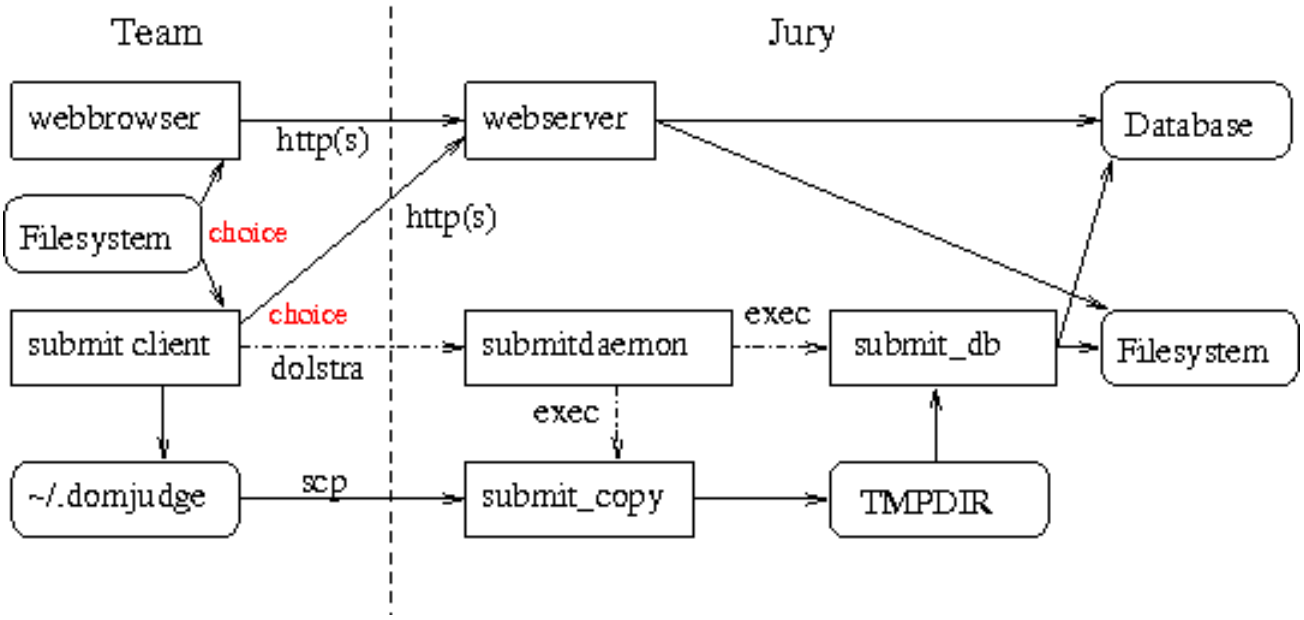

Figure D.1: Submission flow diagram including Dolstra protocol.

To have DOMjudge configure the IP upon first submission in this way, set option `STRICTIPCHECK` to 0. In that case, we start out without IP's (and the web interface will not be accessible), but as soon as a team connects with the command line submit client *to the submitdaemon*, they are authenticated by correctly submitting a file and the IP is registered and everything works as normal.

The connect can happen during the test session, so during the real contest everything is fully available. This is a secure way of authenticating teams, which requires no passwords or IP configuration, but teams must submit via the command line submit client to the command line daemon before they can access their teampage.

D.1 Dolstra protocol requirements

If you want to use the Dolstra submit method (next to / instead of the HTTP functionality) you need to satisfy the following requirements.

The submitdaemon needs to run at the domserver, and receive connections on a configurable TCP port, default 9147.

Team accounts need to be accessible via SSH on the jury computers (a SSH public key of the jury account should be installed on all team accounts to provide key-based access), and a shared filesystem (e.g. NFS) is needed between the team computers and the domserver. Alternatively, another means of providing access from the server can be configured, see the file `submit/submit_copy.sh` for more details.

To build the command line client under Windows, you need to have at least Windows XP and cygwin version 1.7 for support of the complete `netdb.h` headers.