OKUSON Manual

Frank Lübeck
(earlier versions also Max Neunhöffer)

Version 1.5.2

April 8, 2022

Contents

Copyright and License

Chapter 1

Overview

The OKUSON package provides tools for offering exercise sheets via the web. It grew out of programs written and used by the authors since several semesters, when we had to organize exercise sessions accompanying beginners courses in mathematics. Some of these courses had more than 1000 students.

The main purposes of this package are

- to enable the use of exercises which allow a mechanical check (multiple choice, yes-no questions, questions with easy to parse answers like numbers).
- to allow giving **individual** exercise sheets to participants by providing variants of questions and letting the system choose an individual selection depending on the participant by a pseudo-random process.
- to automatize the management of the participants: registration for the exercise course and exams, delivery of the exercise sheets via the web, collecting the solutions of exercises as mentioned above via the web, grading of these solutions, etc.

A more detailed account of our motivation and the description of some experiences can be found in the article:

> Frank Lübeck und Max Neunhöffer, Übungsbetrieb über Webservice, Computer Algebra Rundbrief **31**, Oktober 2002

Our technical goal was to provide a system which is based on reliable and easy to install software on the server side and which only assumes any computer with internet access, any web browser and any program for printing PDF-files on the side of the participating students.

1.1 Contents of the OKUSON Package

In the archive of this package you find:

- A collection of program modules for the scripting language Python; some are generic tools, for example for reading and writing certain kinds of data files, handling of different types of template files, a built-in web server, etc., and some specialized code for this package.
- A copy of the XML parser `RXP` by Richard Tobin at the Language Technology Group, Edinburgh and its wrapper pyRXP for use with Python.
- A complete set of sample web pages (in German and in English) which are probably not difficult to customize for other courses in other places. (The text on these pages is the only language dependent part, such that an adjustment of this package to other languages would be straightforward.)
- A detailed documentation and user guide.
- Example exercises and exercise sheets.
- Checking utilities for OKUSON exercises and sheets.

1.2 How to Use this Manual

- Get started by reading and following chapter 2, then you already have a running server.
- Look at some of the examples of exercises and browse chapters 4 and 5 to learn how to create the exercises and sheets for the service.
- Chapter 6 describes the customization of the web pages, so read this if you need to adjust some details.
- Consider some of the remarks in 2.8 to make your installation more secure and robust. Then you are ready to publish the web site for your course.
- The next few chapters describe some administrative tasks and the corresponding tools in OKUSON when the course is running.
- In the end there are some chapters on more techical aspects of the package. These are mainly interesting if you want to add some functionality to the package.

1.3 Feedback

We are interested in all comments, suggestions, extensions concerning this package. Please, tell us if you are using the package for some course. If you create exercises and sheets in OKUSON format we would be grateful if we could get a copy.

1.4 Acknowledgements

The authors of OKUSON want to thank the following people and organizations, because they have helped to make OKUSON possible:

- Guido van Rossum, his coworkers, and the Python Software Foundation for creating Python.
- Donald E. Knuth for creating TEX and lots of others who have helped to build the whole TEX/LATEX/PdfLATEX system.
- The World Wide Web Consortium (W3C) for standardizing the web, XML, CSS, and lots of other technologies.
- All the people who have contributed to `ghostscript`.
- All the people who have contributed to the `netpbm` tools.
- Richard Tobin for implementing RXP, a validating XML parser.
- Robin Becker and his colleagues at ReportLab Inc. for providing Python bindings for RXP in form of the Python extension module `pyRXP`.
- Thorsten Heck from Lehrstuhl A für Mathematik (RWTH Aachen) for contributing statistic functions for the administrator menu.
- Ingo Klöcker from Lehrstuhl A für Mathematik (RWTH Aachen) for contributing patches and bugfixes.
- Volker Dietrich, Thorsten Heck, Ingo Klöcker, and Axel Marschner for using (and testing) OKUSON even before the first official release and for the valuable hints and suggestions during Wintersemester 2003/04.
- Marc Ensenbach from Lehrstuhl A für Mathematik (RWTH Aachen) for contributing the code for the free form homework input page.
- Ingo Klöcker for contributing the extension framework plus the first few plugins.
- All the others who have been forgotten in this list.

Chapter 2

Installation

2.1 Quick Installation Overview

The following describes briefly how to get the OKUSON package installed on your system. Probably you want to use this section only to get an overview or as a reminder for later installations when you already know the process.

All steps are explained in more detail in the next few sections. When you are actually using OKUSON for a course, you may also want to consider the tips and tricks in 2.8.

1. Make sure you have Python version 2.3 or later installed.
   (WARNING for *Suse Linux 9.0* users: Some users experienced that OKUSON has problems with the non-official version of Python, called '2.3+', which is shipped with *Suse 9.0*. If you have *Suse 9.0* we suggest to get `python` directly from the web site `http://www.python.org`; after unpacking the archive `python` is easily installed by a standard `configure; make; make install` sequence.)
2. Download `okuson-XXX.tar.gz` from:
   `http://www.math.rwth-aachen.de/~OKUSON`
3. Extract archive with:
   ```
   gzip -dc okuson-XXX.tar.gz | tar xvf -
   ```
4. Compile `pyRXP` with:
   ```
   cd okuson/server ; ./makepyRXP
   ```
5. Leave the `server` directory with:
   ```
   cd ..
   ```
6. Copy `Config.xml.sample` to `Config.xml` by doing
   ```
   cp Config.xml.sample Config.xml
   ```
   and edit it to adjust at least the `AdministratorPassword` (use `cryptpasswd` to encrypt your password) and maybe some other obvious entries in the course section of this configuration file.

7. Copy the sample web pages to the document root:
   `cp -r html.sample/* html`
   (or use `html.english/*` for the english version)
8. Copy empty data files to places where they belong:
   `cd data ; cp empty/* . ; cd ..`
9. Start server with:
   `start`
   (If you get a message about 'port in use': another progamm is using that port (e.g., another OKUSON server), change the `Port` entry of the `Config.xml` file and try again.)
10. Test server by pointing your browser to
    `http://localhost:8000/index.html`
    (substitute the `8000` if you have chosen another number in the `Port` entry of the `Config.xml` file).
11. If things look alright, start customizing for your application: have another look in the course section of the `Config.xml` file, adjust details of the web pages (see 6), add information on tutoring groups (if applicable, see 8.2), prepare the first exercise sheets (see 5). A more detailed account of tasks occurring during a semester is given in chapter 3.
12. **Note**, that during the semester the OKUSON server collects and stores personal data of the participating students. So you **must** be careful with the access to these data.

2.2 Prerequisites

Personal

To use the OKUSON package as an administrator you need a working knowledge of the following things:

- Editing ASCII files with a text editor.
- Installing and starting programs under UNIX.
- Writing HTML.
- Writing LaTeX code for exercises.

In addition you should be willing to learn a few facts about XML (see chapter 4) and XHTML 1.0. And, if you want to know in some detail how things work or if you want to add some functionality to the package, you need knowledge about the programming language Python.

Technical

To run the OKUSON server you need a UNIX system that is accessible from the internet (so, you may need help from an administrator if you are behind a firewall). The following software must be installed:

- **Python** version 2.3 or later. You can test your Python installation by just typing `python` in your shell. If it is installed, a short banner with the version appears. You can leave the interpreter again by typing Control-D.

  If you are using Linux, there are probably packages available for your distribution. However, you can also download the source from `http://www.python.org` and install it without having root privileges.

- A **C-compiler**. OKUSON uses an XML parser coming as a Python extension module, which is implemented in C. We include this module in the OKUSON distribution. However, it has to be built with your version of Python being available. Note that your Python installation must be fairly complete, in some Linux distributions you must install a package with a name like python-dev or similar.

- **LaTeX**. The texts for questions are processed by TeX, therefore one does need a full featured installation of TeX/LaTeX with PDF support. For example the teTeX-distribution provides everything necessary.

- **ghostscript**. Exercise texts are translated to images for use on the web pages delivered by OKUSON.

- **netpbm**. A package for image conversions and manipulations, the program `pnmtopng` leads to particularly small files for the exercise text images. All Linux distributions probably have a `netpbm` package, for other systems see http://netpbm.sourceforge.net.

- A **text editor**. Input for exercises, sheets and web pages is generated with an ASCII text editor, which is also used to edit the configuration.

- **iconv** and **pandoc** are needed if you want to provide the MathJax version of the exercise sheets. This option is available since version 1.4, see 6.5.

If you want to use all features of OKUSON, the following additional software packages are needed:

- A **web browser**. Not only the user access but also a great deal of administrative tasks are done via the web interface, using OKUSON's built-in web server. OKUSON is tested with a wide range of web browsers, including Mozilla and friends, Opera, Netscape, Lynx, w3m, and even Internet Explorer.

- A **PDF viewer** like `xpdf` or `acroread`. Previews of exercise sheets are generated as PDF files.
- `xloadimage` (or some other viewer for images in PNG format) is needed to display test versions of exercises.
- `less` is used to conveniently browse log files.

The students participating in the courses organized with OKUSON only need a web browser on any operating system and a PDF viewer to print out their exercise sheets.

The following is **not** needed to run an OKUSON server:

- Root privileges. A normal user can install and run an OKUSON server.
- Web server. No web server apart from the one built into OKUSON is needed.
- Data base. OKUSON uses a simple and efficient way of storing data in and reading from human legible files. No external data base system is needed.

Needed Resources

As an example, we give here some facts and numbers on our use of the OKUSON system for a big course during one semester (these are estimates, based on experience with previous versions of the OKUSON programs):

Server Hardware: We use a PC with two Pentium III (1GHz) processors, 2 GB of memory, 40 GB IDE hard disc. However, any recent PC with say at least 256 MB of memory and 100 MB of free disc space and a reasonable connection to the internet could do the job as well. The workload of the OKUSON server, even for a course with many participants, does not have a detectable impact on other work done on our machine. This holds as well for CPU time as for memory usage.

Number of participants: 1200.

Number of submissions of solutions: more than 20000 (this leads to the by far largest file in the `data` subdirectory, we expect a file of about 3 megabyte, where our sheets contain 20 interactive questions each. Restarting the server with rereading all these data takes about 5 seconds.

Memory needed by the server process to hold all data in memory: a few hundred kilobytes, hence neglectible.

Number of page/file accesses: a few million, even with (artificially caused) several thousand accesses per minute, the machine is still usable for other tasks as well.

Size of `log/server.log` file: could grow to a few hundred megabytes. If you are short of disc space, you can compress the files (with `bzip2` they can be shrinked to 3–4% of their original size), or even throw away older entries.

Number of PDF sheets produced on the fly: in average each sheet is requested between one and two times in PDF format. That means that during the semester up to 30000 PDF sheets are requested. The server machine needs about 4 to 5 hours of CPU time for the corresponding calls to `pdflatex` — distributed over the whole semester.

2.3 Download

You can download the OKUSON distribution from the following web address:

`http://www.math.rwth-aachen.de/~OKUSON`

There you find this manual as a PDF file and a gzip'ed tar archive containing the source code. You can extract the contents in any place in the filesystem that is convenient by the following command:

```
gzip -dc okuson-XXX.tar.gz | tar xvf -
```

This command creates a directory `okuson` in the current working directory, under which the full distribution resides. This directory will be called "the OKUSON home directory" or short "`$OKUSONHOME`" in the sequel.

The next step is to compile the `pyRXP` extension module.

2.4 Compilation

As OKUSON uses the Python extension module `pyRXP` which builts on the XML parser `RXP` written in C, some compilation is necessary. Note that we have patched the parser code in one small place to allow compilation with the GNU `gcc` compiler versions 4.0 and above. Assuming that you have extracted the distribution as described in the previous section, you can compile `pyRXP` with the following commands:

```
cd okuson/server ; ./makepyRXP
```

After the compilation you should find two files `pyRXP.so` and `pyRXPU.so` in this directory. Otherwise, please look for error messages in the output.

You can now leave the `server` directory again with

```
cd ..
```

The next step is to edit the configuration.

2.5 Configuration

In the OKUSON home directory (the directory `okuson` which was created during extraction of the archive) there is a file named `Config.xml.sample`. It is a sample file for the file `Config.xml`, which is the central place for configuration of the OKUSON server. In the beginning, you should copy the sample file to the real one with the command:

```
cp Config.xml.sample Config.xml
```

This file is an XML file (see section 4 for a short introduction) that can be edited conveniently with an ASCII text editor. The entries of this file are explained in detail via embedded comments.

For a start and to get an impression adjust the first few entries in this file, say `CourseName`, `AdministratorPassword`, `Semester`, `Lecturer`, `Feedback` and maybe `Port`.

Note that if you have several courses running at the same time and want to use the same machine for all those OKUSON servers, you need a different port for each instance of the server.

Note that the administrator password is stored encrypted, use the script `cryptpasswd` in the OKUSON home directory to find an encryption of your password.

2.6 Getting started

OKUSON has a built-in web server and does nearly all its user interaction via a web interface.

The package contains a set of sample web pages (in German and in English) and to get started just copy them into the subdirectory `html` (which is the document root of the built-in web server in the default configuration). This is done with the following command, issued from the OKUSON home directory:

```
cp -r html.sample/* html
```

(or `cp -r html.english/* html`)

The stuff that is copied with the abovementioned command consists of

- Style sheets with the extension `.css`.
- Web page templates with the extension `.tpl`. These are processed by the OKUSON web server during startup and then delivered to the web browser.
- A few images with the extension `.png`.
- A default address line icon `favicon.ico`.

Note that most of the functionality of the OKUSON server is driven by these web pages. Their structure and content lead the user through the menu system, organizes input and output from the server and is therefore essential.

In chapter 6 we explain how to customize these web pages for your course.

An OKUSON server collects data about participants of your course, submitted solutions of exercises, tutoring groups, etc. Such data are stored in the subdirectory `data`. For a start you need empty versions of the data files. This can be achieved by the following commands:

```
cd data ; cp empty/* . ; cd ..
```

This copies the empty versions of the data files to the real versions. Please do not do this later when your server already has accumulated data!

2.7 Starting and Stopping the Server

You can now try out the OKUSON server by typing

```
start
```

in the OKUSON home directory. (Be patient, the first time you call this some images of exercise texts are generated before the server can start serving.) If everything went well, you see some log messages, concluding with

[TIMESTAMP] `Ready to start service ...`

Otherwise, you have to read through the log messages and identify error messages, which start with "`Error:`". You also find the complete log file under `log/server.log` in the OKUSON home directory.

If the server was started up successfully, you can test it by pointing your browser to the following URL:

```
http://localhost:8000/index.html
```

where the main menu for students should appear. Note that the "`:8000`" in this URL tells the browser to contact the OKUSON server on port 8000. Therefore this of course has to be changed according to the network port you chose during configuration.

If you want to stop the server, you can issue the command

```
stop
```

in the OKUSON home directory. The server finishes the ongoing requests and then terminates showing a few log messages.

There is also a script

```
restart
```

that stops the server and restarts it.

In general, whenever you change something in the servers setup (say, you add or change exercises or sheets or add some data in the `data` directory, change the `Config.xml` file, ...) you must restart the server. The advantage of this approach is that you can play around with your changes before they are published to the outside world by the server (and the server is more efficient, because it caches and preparses everything it needs during startup).

2.8 Tips and Tricks: Security, Backup, etc.

In this section we mention some details of our own use of this package. Maybe you find some of these remarks useful.

Setting the Locale

We set the locale for the server correctly, such that date and time strings are displayed according to the local standards. This is achieved by setting the `LC_ALL` environment variable to appropriately (for example `de_DE` for German, see the `man` page of `locale` on your system, usually "`locale -a`" lists the available settings).

Using a Pseudo User

We always create a pseudo user as administrator of the package. This makes it easy to share the administration between several people without fiddling around with strange file access settings in private directories.

We close completely (`chmod 700 ~`) the access to the pseudo users home directory. **Note** that during the semester the OKUSON server collects and stores personal data of the participating students. So, you **must** be careful with the access to this data.

Securing the Administrator Access

We always choose a very restricted setting for the machines from which administrator access to an OKUSON server is allowed, see the entry `AdministrationAccessList` in the configuration file `Config.xml`.

Note that the administrator of an OKUSON server is quite powerful: in all web forms of the system where a password (of a participant, a tutor or the administrator) is required, the administrator password is considered valid *provided* the request is coming from a machine with administrator access. As administrator you can see sheets which are not yet open, submit solutions to closed sheets, change a user password, get any user's sheets and results and use the `/adminmenu.html` functions.

Make Sure Your Server Uses the Correct Time

Since an OKUSON server is very strict about accepting submitted solutions as long as a sheet is open and showing the grading after a sheet is closed, it is very important that the server uses the correct local time. Otherwise expect unpleasant discussions with your students, some of them tend to hand in their solutions in the last minute.

We use the network time protocol (NTP) service on our machines, using the `ntpdate` program.

Encrypted access via `https`

We have not built the https-access into OKUSON because that would mean that you have to generate new SSL-certificates for every course. Instead we use `nginx` as a https-proxy server. With such a proxy server you can either still allow the http-access with the configured port number, or you can restrict the access to the OKUSON server to the machine running the proxy server (and maybe some local machines), see `<AccessList>` in the config file `Config.xml`.

Backup

Since the data collected by an OKUSON server can be relevant for grades it is important to have a good backup system for them.

We use a machine different from the one running the OKUSON server for this. It has Paul Vixie's version of the `cron` program installed and we define (as the OKUSON pseudo user) `crontab` entries like:

```
MAILTO=""
*/5 * * * * ~/okusonsync
10 * * * * ~/okusonarchive
```

Here, the scripts `okusonsync` and `okusonarchive` are in the pseudo users home directory and look as in the following examples (using the `rsync`, `tar`, `gzip` and `ssh` programs, assuming that the pseudo user can login from the backup machine to the server machine with ssh without password).

```
#!/bin/sh
##  okusonsync    syncronizes an OKUSON inst. using rsync

BACKUPDIR="..."
ORIGSERVER="..."
ORIGDIR="..."  # no final '/'

cd $BACKUPDIR
rsync -a --delete -e ssh $ORIGSERVER:$ORIGDIR .
```

and

```
#!/bin/sh
##  okusonarchive     packs whole backup into an archive

BACKUPDIR="..."
DIRNAME="..."
```

```
cd $BACKUPDIR
fname=`date +"BACKUP-%Y-%m-%d_%H_%M_%S.tgz"`
tar czf $fname $DIRNAME
```

In this setup, every five minutes all files of an OKUSON installation are synchronised to the backup machine (this takes a split second after the first time). And once an hour all files are packed and compressed, archiving the precise status of the whole system.

(Furthermore we have a daily backup of all data on all machines in our institute.)

These precautions allow a quick recovery of most or all data after a possible crash or data corruption of the original server.

Restarting Server after Reboot

Usually, our server machines are never rebooted. But in case that happens for some reason (e.g., a power failure) it is desirable that the downtime of the OKUSON service is as short as possible. For this we use an entry in the `crontab` of the pseudo user on the server machine like

```
@reboot  /MYOKUSONDIR/server/Server.py &
```

where `MYOKUSONDIR` has to be replaced by the full absolute path to the OKUSON home directory.

Alias for Server Machine

We create an alias for the machine running the OKUSON server. This means putting an extra entry with an additional name in the domain name server (DNS). One usually needs root privileges to do this.

The advantage of the alias name is that it allows a quick move of the whole service to another machine, in case the server crashes for some reason.

Paperless Courses

With the exception of exam exercise sheets we do not print any paper for courses with OKUSON exercises. Registration, exercise sheets, announcements, exam results, final grades are *only* available via the OKUSON server web interface. Our experience was that no students complain about this approach.

Keeping the log file data small

During a course the log file `log/server.log` grows substantially. Therefore we provide a small script, which automatically keeps some older versions of the log file and deletes very old ones. It is called `logrotate.py` and resides in the `scripts` directory of the OKUSON distribution. Usually one will copy it to the log file directory and install it in a crontab like

```
PATH=$HOME/bin:/usr/local/bin:/usr/bin:/bin
17 3 * * * $HOME/okuson/log/logrotate.py $HOME/okuson/log/server.log
```

You can configure the number of versions to be kept and the number of uncompressed versions at the top of the script. It will shift the log files up and delete the oldest.

Chapter 3

Schedule of Events for One Course

In this chapter we go through the complete schedule for one course as we imagine it and describe the actions one has to perform during the various stages with respect to OKUSON. Of course, you will have to adapt this to your specific situation. We hope however, that this chapter also gives a good overview over and a reference for the facilities of the OKUSON system, because it also contains a lot of references to various sections of this manual.

We proceed chronologically and divide roughly into the following stages: before the semester, at the begin of the semester, during the semester and after the semester.

3.1 Before the Semester

- **Installing OKUSON.** See chapter 2. Here one typically adjusts the configuration options for the course. After this step, the pages are ready for registration. One can already create the pages with bibliography and general information about the course.

- **Creating exercises.** It can be reasonable to prepare some exercises already in this early stage. One should not underestimate the time that is needed to invent decent multiple choice exercises and enough variants. If no exercises are available from earlier courses we calculate one day per week for two people to create the exercises for one sheet.

  It also cannot hurt to get used to the way exercises and sheets are entered into OKUSON.

  It has proven sensible to have a sheet number 0 which does not count in the end for the grade and which is available already at registration time, such that the participants can practice to use the system before the actual exercises begin.

3.2 At the Beginning of the Semester

- **Registration of participants.** Once you tell the students the URL of the OKUSON server, they can register. Obviously, you want to give them a few days for this purpose, before you distribute the registered participants into small tutoring groups.
- **Distribution of participants into tutoring groups.** See chapter 8.
- **Setup of group information.** About at the same time than the previous step you want to edit `data/groupinfo.txt` to set up the information about your small tutoring groups. You can enter the names of the tutors and the time and place of their sessions. Also every tutor gets a password at this stage.
- **Publishing of distribution into tutoring groups.** From the moment you enter the distribution data into `data/groups.txt` and restart the OKUSON server, the distribution is automatically published via the web pages. Note that the information you entered in the previous step is shown on the automatically generated pages.

3.3 During the Semester

- **Distribute late-comers into groups.** Especially in big courses with hundreds of participants, there always will be some sleepers who only register late. These people automatically appear in group number 0. One has to distribute them by hand into tutoring groups by appending lines to the file `data/groups.txt`.
- **Create new exercise sheets.** Of course, you will create further exercise sheets during the semester. Note that you can use the attributes `openfrom` and `opento` to prepare the sheets beforehand and let them appear automatically at a given time on the web pages and let them be closed at another time.
- **Let participants query their results.** The participants will query their results on a regular basis. Please note that there is the possibility to put up individual messages on the result pages via the administrator menu (see section 7.1.4) and to delete them later.
- **Tutors enter homework results.** There is a page especially for the tutors to type in the results of the participants in their tutoring group, such that they appear on the result page. See section 8.4.
- **Exams during the course.** Especially for beginner's courses it seems to be a good practice to organize a written exam at some stage during the semester. The OKUSON server can help you to organize this by handling the registration of participants for such exams and by producing the messages for the exams on the result page automatically, once the result of the exam is imported into the server (see 9).

3.4 After the Semester

- **Exam at the end of the semester.** The OKUSON server can help to organize the registration of participants of exams at the end of the semester, see 9.
- **Automatic grading.** At this stage one can activate the automatic grading function to produce output for the result pages of the participants, for example stating whether they get a certificate about successful participation in the course or not. Note that to use this one has to learn a little bit of Python programming to customize the automatic grading function to your needs. One also has to learn a little bit about the internal data structures of OKUSON, see chapter 12. You can also choose not to use this feature, see the next step.
- **Export of results for further processing.** Now that all data about results is collected one can export these via the administrator menu (see 7.1.16). The idea of this step is to be able to do statistical evaluations on this data. However, it is also possible to use external tools or scripts on this exported data for example to decide about grading by your own criteria. You can then import the result of these external tools in form of private messages to participants via the file `data/messages.txt` (see 11.5).
- **Producing certificates.** One can use the export of results to produce certificates of successful participation. We use a script called `schein.py` which can be found in the `scripts` directory of the OKUSON distribution to produce certificates. The script is quite self-explanatory (just start it up without arguments).

3.5 At all Times

- You can at all times use the OKUSON system to publish current information about the course. Our experience shows that usually no other means of publishing such information is needed any more.

Chapter 4

Short Introduction to XML

XML is a standard by the W3C consortium (`http://www.w3c.org`) that allows to define *markup languages* for text documents and data such that documents using such a language can be checked and processed systematically with generic tools.

The OKUSON package uses XML in several places:

- its configuration file `Config.xml`
- the files specifying exercise sheets (`xxx.bla`)
- the files containing the interactive exercises (`yyy.auf`) - but the exercise texts are written in LaTeX, a much more complicated markup language
- the delivered web pages are written with XHTML markup and the templates for these pages are written with an XML markup

The markup languages in each of these cases are different, but they are almost self explaining and we provide sample files for all of them.

In this chapter we give a basic introduction to the XML concept. This knowledge may be useful to avoid syntax errors and to understand the messages of programs parsing an XML file containing errors and to correct such errors.

4.1 An Example XML Document

Here is a complete XML document. Below we give a short glossary and explain the main concepts of the markup.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE MainElt SYSTEM "nowhere.dtd">

<!-- This is a comment.          -->

<MainElt>

<A xattr="val of xattr">abc<B yattr='contains " quotes'></B></A>

<EmptyElt />

<Long><![CDATA[
Protected <A> <B> XXX </A> </B>
]]></Long>

<Short>With &lt; entities &gt; and ampersand &amp; </Short>

</MainElt>
```

An XML document usually starts with some meta-information.

The first line `<?xml ...` just tells that this is a document with an XML defined markup.

The `<!DOCTYPE MainElt SYSTEM "nowhere.dtd">` line specifies the language to which the markup belongs by pointing to the file name of a so called *document type definition* (DTD). It also tells that the actual content of the document is enclosed by `<MainElt ...>` and `</MainElt>`.

The actual content of an XML document is structured by so called *elements*. They basically have the form `<A>something</A>`. Here `<A>` is called the *begin tag* and `</A>` the *end tag* of the element and `something` is the *content* of the element.

The *name* of an element, here `A`, is case sensitive, that is `<A></A>` and `<a></a>` are different elements.

There always must be a begin and end tag. A begin tag can contain further information as in `<A xattr="val of xattr">` above. This extra information is called an *attribute*, where `xattr` is the *name* of the attribute and "`val of xattr`" is the *value* of the attribute. An attribute must always have a value and the specification of the value must be enclosed either in double quotes (`"`) or in single quotes (`'`). The enclosing quotes are not allowed within the value.

Contents of elements can themselves contain strings and/or other elements. But elements must always be properly nested, as in the line `<A xattr=...` above, for example `<A> <B> XXX </A> </B>` is *not allowed*.

Anywhere before or after all elements or in the content of an element there can be a *comment*. This is specified by enclosing it in `<!--` and `-->`, the only character sequence not allowed in a comment is `--`. A comment is *not* part of the content of an element.

There can be elements whose content is always empty and which can be written with a combined begin and end tag as in `<EmptyElt />`, these are called *empty elements*.

Sometimes the content of an element is a string which looks like containing XML markup. There are two possibilities to avoid an interpretation as markup.

The first is demonstrated in the `Long` element above, its content is enclosed in `<![CDATA[` and `]]>`. Everything between these markers is considered as content as it is. The only string not allowed within such a CDATA section is `]]>`. (Note that the line `Protected...` is just element content and that there is no syntax error.)

The other possibility are so called *entities*. They are demonstrated in the `Short` element of our example. They start with an ampersand `&` and close with a semicolon `;`. Entities are placeholders for which a substitution text is somewhere defined. For example, the given entities `<`, `>` and `&` specify the special characters `<>&` used for XML markup.

A document type definition (DTD) defines a markup language in the sense that it says which elements are allowed in a document, which attributes are allowed for each element and how elements can or must be nested. We don't explain this in more detail here. The OKUSON package has a directory `dtd` containing the definitions of the markup used in this package.

Summary

XML documents contain elements which must be empty or have begin and end tag and which must be properly nested. Element names are case sensitive. Elements can have attributes which must have a value and the value must be enclosed in double or single quotes. There are CDATA sections and entities for specifying content containing the special characters used for XML markup.

Reference

A detailed specification of the XML standard with useful annotations is given in
`http://www.xml.com/axml/axml.html`.

4.2 Parsing and Validating XML Documents

The main advantage of using XML for new document formats (like the exercise sheets and exercises in OKUSON) is that one can use standard tools to read in and process the content of such documents. Programs which read XML documents and translate it to some internal data structure are called *XML-parser*.

XML documents that follow the formal rules explained in 4.1 are called *well-formed XML documents*. This means, that they can be parsed successfully, however, this does not imply that they make any sense.

As mentioned above the `DOCTYPE` entry of an XML file can point to a document type definition. This defines element names and gives restrictions how elements can or must be nested, gives some information on the type of content of certain elements, and it defines for each element which attributes are allowed and maybe some restriction on their values.

An XML document which is consistent with the definitions in its document type is called a *valid XML document*. Some XML parsers, like the one we use in the OKUSON programs, can check if an XML document is valid. Such a parser is called a *validating parser*. It is useful for the user who wants to check if a newly created document has the correct XML structure, and it is useful for the programmer as a general tool for checking the formal correctness of XML input.

As administrative user of OKUSON you can come across the parsing of XML files when you use the testscripts for exercise and exercise sheet documents (see 5.3 and 5.4) and during startup of the OKUSON server when the template files for the web pages are cached.

Validating and Checking Well Formedness

In the OKUSON home directory, there is a script `xmlvalidate` which can be used to validate XML documents. (With `-w` argument it is only checked if a document is well-formed.) In case of an error, a sensible message for removing the problem is printed.

Chapter 5

Creating Exercises and Sheets

The OKUSON package delivers exercises via the web. There are two types of exercises, *text exercises* for which written solutions are necessary and *interactive exercises* whose solutions are submitted via a web interface and which are automatically graded by the OKUSON server. The interactive exercises depend on the participant, they consist of a (pseudo-)random ordering and choice of variants of some questions. Such exercises cannot be specified in a single static LaTeX-file, but some additional information is needed.

While the actual exercise texts must be written in LaTeX, the additional structure is added via some simple XML markup, see 4. We provide for both types of documents templates which you can just copy and fill in. Although these are almost self explaining we describe the details in this chapter.

5.1 Scoring the solutions

The scores for the interactive exercises which are graded by the OKUSON server, are by default computed as follows:

- each correct answer gives $+1$ point
- no answer to a question gives 0 points
- each wrong answer gives -1 points
- each exercise gives at least 0 points (in particular, negative points are not merged with positive points from other exercises)

(With these rules participants are encouraged to give no answer instead of a wrong one whenever they are not sure.)

If you combine interactive exercise with other homework exercises, then adjust the scores for the latter such that you get the intended relation between the two types of exercises. Partial points (in decimal notation for OKUSON) can be given.

The scores for a correct or wrong answer respectively can be configured by using the `scorecorrect` and `scorewrong` attributes of the `QUESTION` element, see 5.3, and by using the configuration options `MCScoreCorrectDefault` and `MCScoreWrongDefault` in the `Config.xml` file. The minimal score for a complete exercise can be configured using the `mcscorelowerlimit` attribute of the `EXERCISE` element and the configuration option `MCScoreExerciseLowerLimitDefault`.

Note however, that it is the responsibility of the user that in the case that the `nrquestions` attribute in the `EXERCISE` element of a sheet file is used and makes OKUSON choose a subset of questions every possible such subset has the same maximal score!

5.2 Generalities on the Exercise and Sheet Files

The directories containing exercises and sheets for an OKUSON server must be given in the configuration file `$OKUSONHOME/Config.xml`, see the `<ExerciseDirectories>` and `<SheetDirectories>` elements. Each directory is given as a `DIR` element whose content has the form `path/to/dir` or `path/to/dir|prefix`, the prefix is used to distinguish exercises with the same name from different directories, see 5.4.

The default directories are `$OKUSONHOME/exercises` and `$OKUSONHOME/sheets`, but you can change these and also use several directories for each or merge them into a single directory.

There are three types of files recognized by OKUSON which are distinguished by their file extensions as follows:

`.tex` this is for text exercises, such a file contains just the text of the exercise in LaTeX-format.

`.auf` this is for files containing one or several interactive exercises, these are XML files with the actual exercise texts written in LaTeX.

`.bla` this if for files which specify one or several exercise sheets. They contain some meta information like sheet number or submission date and refer to the exercises to include, some intermediate texts can also be given (in LaTeX-format).

Publishing and Changing Sheets

Whenever you have added (or changed) an exercise sheet you must restart the server. (There is no automatism for this to avoid that participants get incomplete or messed up sheets.)

After a sheet is published only small changes are allowed: Do **not** add or remove exercises, questions or variants of questions. Whenever you change an exercise text be aware that participants may have already downloaded the old version.

Whenever you need to correct the given solutions of interactive exercises (see 5.3) you must reevaluate the solutions submitted so far, see 7.1.6.

Which LaTeX Macros?

In the standard setup of OKUSON the following packages and macros are loaded when `latex` is called:

- `inputenc`, with parameter `latin1` which allows to use German umlauts and other western European accented characters in the LaTeX code
- `graphicx` for including images in exercises
- `amssymb` which provides many mathematical symbols, and finally
- macros `\Z`, `\N`, `\Q`, `\R`, `\C`, `\F` as abbreviations for `\mathbb{Z}`, and so on.

The LaTeX templates used for producing the images of exercise texts for the web pages and for producing the exercise sheets in PDF-format can be found (and adjusted, if you know what you are doing) in the OKUSON configuration file `$OKUSONHOME/Config.xml`, see the elements `<LaTeXTemplate>` and `<PDFTemplate>` (or `<PDFTemplateNoTable>` without interactive exercises).

If you just want to add the loading of some further packages or to read in some of your favourite personal macros, use the `<ExtraLaTeXHeader>` element of the configuration file.

Note that if the LaTeX runs need further input files like for example images, you have to make sure that LaTeX finds them in your filesystem. This can for example be achieved by setting the environment variable `TEXINPUTS`. Note that with the standard UNIX version of LaTeX you have to append a colon to the value of `TEXINPUTS` such that the usual files for LaTeX are still be found, i.e. one has to use a command like this before starting the server:

```
export TEXINPUTS='/path/to/additional/TeX/files:'    (for bash)
```

or

```
setenv TEXINPUTS '/path/to/additional/TeX/files:'    (for tcsh)
```

Comments in LaTeX-Code

The LaTeX-code for exercises is used in the `alt`-attributes of the images in the HTML version of the exercise sheets. More precisely, a simple striping of comments is applied: lines starting with a `%`-character are deleted, and in other lines everything after the first `%`-character which is not preceded by a backslash is removed. The images are generated with the non-striped input.

This behaviour allows to include references for solutions or solutions in comments to the exercise.

Remark on the PDF-version of the Exercise Sheets

The PDF-version of an exercise sheet shows exercises and questions in a table with border lines between the entries. This should make it look similar to the HTML-version of the sheet, such that the transfer of the solutions of interactive exercises from the printed version to the web form is easier.

This LaTeX-input for the PDF-version uses the `longtable` environment and the table entries are packed in `minipages`. This seems to be quite robust with the setup mentioned above. We had reports that sheets could not be compiled by LaTeX with exercises using certain environments from additionally loaded packages. While these are probably bugs in one or some of the involved packages, it is difficult to find a general strategy to avoid them. As a general rule, when something goes wrong, try to enclose critical blocks by additional markup, e.g., enclose with { . . . } or put text in some boxes.

5.3 Writing Exercises

A text exercise used by OKUSON is written in a separate file with the extension `.tex`. Put only the LaTeX-code of the exercise text itself in that file.

If the file contains a line that consists exactly of the string "`% SOLUTION`", then everything after that is considered to be a solution to the exercise. Before the closure time of the sheet, only the part before the `SOLUTION` line is displayed, after the closure time of the sheet, the full text is shown. This is a mechanism to publish example solutions to written exercises.

We explain the format of an `.auf` file by an example that contains all possible constructs. In practice one may copy the template file

```
$OKUSONHOME/exercises/empty.auf.template
```

and fill in the exercise texts.

An Example File `example.auf`

This example demonstrates all types of questions which can be posed in interactive OKUSON exercises. The example probably almost explains itself, but read the comments below on the types of questions and how the possible answers are specified.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE EXERCISE SYSTEM "exercise.dtd">

<EXERCISE key="Example1" keywords="">

<ANSWERS type="r">Yes | No</ANSWERS>

<TEXT>
Let $a = \frac{2}{3}$ and $b = \frac{14}{7} \in \Q$ and
set $c = 3a + 5$ and $d  = a + 7b$.
</TEXT>

<QUESTION>
<VARIANT solution="Yes">
The numerator of $a$ is $2$.
</VARIANT>
<VARIANT solution="No">
The denominator of $a$ is $2$.
</VARIANT>
</QUESTION>

<QUESTION>
<ANSWERS type="c">b | c | d </ANSWERS>
<VARIANT solution="b|c">
Mark the number(s) which are integers.
</VARIANT>
<VARIANT solution=" d">
Mark the number(s) which are not integers.
</VARIANT>
</QUESTION>

<QUESTION>
<ANSWERS type="s"></ANSWERS>
<VARIANT solution="30">
What is $3d+c$ (canceled down)?
</VARIANT>
<VARIANT solutionregexp="^4$|^9$|^14$">
```

```
Give an integer which is the sum of two of the
numbers $a, b, c, d$.
</VARIANT>
</QUESTION>

</EXERCISE>
```

As usual for XML documents the first two lines say that this is an XML encoded document whose document type is described in a file `exercise.dtd` and whose content consists of an `EXERCISE` element.

The document type definition is contained in OKUSONs `dtd` subdirectory. We now explain all elements in this example, these explanations include the rules given in the document type definition.

The `EXERCISE` element marks one exercise. Its content is a sequence of `ANSWERS`, `TEXT` and `QUESTION` elements.

An `EXERCISE` element can have the optional attribute `mcscorelowerlimit` which changes the minimal score a participant can get (default is 0, this default can be overwritten with the configuration option `MCScoreExerciseLowerLimitDefault`).

`TEXT` elements are used for text appearing before the actual questions of the exercise. Their content consists of LaTeX code. `TEXT` elements are considered only, if they are in the first or last position within the `EXERCISE` element. A possible first `TEXT` element is used as prefix before all questions of this exercise and a possible last `TEXT` element is used as a postfix after all questions of this exercise. All other `TEXT` elements are silently ignored. This behaviour might be changed in future versions.

There can be at most one `ANSWERS` element. If it is there it describes a default for the answers to all questions of this exercise. This element has the form `<ANSWERS type="?">? | ? </ANSWERS>` with three possibilities for the `type` attribute:

`"r"` (radio button) this is for questions where several possible answers are given and exactly one of them is correct. In this case the content of the `ANSWERS` element is a list of strings for the possible answers which are separated by `|`-characters. Leading and trailing whitespace in each possibility is removed (so that `>Yes|No<` and `> Yes | No <` are interpreted the same).

`"c"` (multiple choice) this is for questions where several possible answers are given, and some subset of them is correct. This means that one has to select **exactly** the right subset to score a point. The syntax for the content in this case is the same as for `"r"`.

`"s"` (string) this is for questions where the answer is expected as a string (that has to be typed into an input field). In this case the content of the `ANSWERS` element should be empty.

The QUESTION elements are for the actual questions. Such an element can have the two optional attributes scorecorrect and scorewrong determining the score for a correct answer (default is $+1$, this default can be overwritten using the configuration option MCScoreCorrectDefault) and a wrong answer (default is -1, this default can be overwritten using the configuration option MCScoreWrongDefault) respectively. The content of a QUESTION element is a sequence of at most one ANSWERS element and at least one VARIANT element. In case there is an ANSWERS element, this must have the same syntax as just described and it is taken as description of the answers to all variants of the current question. If there is no ANSWERS element, the default given in the content of the EXERCISE element is taken.

The content of each VARIANTS element is the LaTeX code of a question. The correct solution to this question is specified in one of the attributes solution or solutionregexp, more precisely the syntax depends on the answer type of the question:

"r" the solution must be given in the solution attribute and be exactly one of the answers given in the relevant ANSWERS element (again, up to leading or trailing whitespace).

"c" the solution must be given in the solution attribute and be a subset of the answers given in the relevant ANSWERS element, separated by |-characters.

"s" the solution can be given either in the solution attribute as list of possible answer strings separated by |-characters (usually just one string), or it can be given in the solutionregexp attribute as a regular expression such that an answer is correct if and only if it matches this regular expression. To be precise: the given expression must follow the syntax documented in the re-module of python, the matching is done with the search method of regular expression objects in that module. In any case in the solutions and submitted answers leading and trailing whitespace will be deleted before the comparison.

Several Exercises in One Document

It is possible to put several exercises as above into one document. In that case the document should look like

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE EXERCISES SYSTEM "exercise.dtd">

<EXERCISES>

<EXERCISE>
   ...
</EXERCISE>

...
```

```
</EXERCISES>
```

(note the entry `EXERCISES` in the `DOCTYPE` declaration). There can be an arbitrary number of `EXERCISE` elements as explained above.

Checking Exercises

The OKUSON package contains a utility script for checking files with interactive exercises as well as text exercises. The script assumes that you have the pager program `less` and the image viewer program `xloadimage` (or some other image viewer which you specify with an optional argument) installed on your computer. The script is called as follows:

```
$OKUSONHOME/testexercise [-v viewer] file1 file2 ...
```

where `file1`, `file2` are exercise files with either `.tex` or `.auf` extension.

For each `.tex` file the LaTeX code is processed as configured in your OKUSON installation and the result is converted to an image. If there was a problem with the call of `latex`, the pager program `less` is called with the log file describing the error.

Otherwise, if an image could be successfully generated, this image is shown with the help of the image viewer `xloadimage`. Except for the pixel resolution this is the image used to display the corresponding exercise text to the course participants in the HTML version of the exercise sheets.

For each given name of an `.auf` file this utility first parses the file and checks if it is a valid document. If not, some error message is printed giving the exact position in the file where a problem occured.

After a successful parsing of the file the utility considers each piece of LaTeX code given in some `TEXT` or `VARIANT` element. Each such piece of text is checked as described above for an exercise text from a `.tex` file.

5.4 Specifying Sheets

For exercise sheets in OKUSON there is another XML document type definition which we will also explain by looking at an example. Files of this type must have `.bla` as their extension. In these files, interactive exercises are referred to by the value of their `key` attribute, and text exercises are identified by their file names.

Example of an Exercise Sheet document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE SHEET SYSTEM "sheet.dtd">

<SHEET counts="1"
       magic="2003"
       nr="0"
       name="0"
       first="1"
       openfrom="12:00_01.09.2003"
       opento="14:00_20.10.2003">

<TEXT>The first exercise is interactive. You must send the
solution via the web interface.</TEXT>

<EXERCISE key="Example1" prefix="" nrquestions="2" order="p" />

<TEXT>For the next exercise hand in a written
solution to your tutor. </TEXT>

<INCLUDE file="fractions.tex"/>

<TEXT>Solutions for this sheet are accepted until
Monday, Oct. 20, 2 pm.</TEXT>

</SHEET>
```

The first two lines tell that the document is an XML document with a document type specified in a file `sheet.dtd`. The content of the file consists of a `SHEET` element.

A `SHEET` element has several attributes:

`nr` must be the string of a non-negative integer. All sheets must have different values for this attribute, which is used to order all sheets numerically.

`name` (optional, default value is value of `nr` attribute) this is used as a name of the sheet that appears for example in the header of the sheet, for example, it could be `name="Test 1"`.

`first` (optional, default value is `1`) fixes the number of the first exercise on this sheet.

`counts` (optional, default value is `1`) determines if this sheet is relevant for the grading of the course, value `1` means *yes* and value `0` means *no*.

magic string of some positive integer number smaller than 2^{32}. This is used in the pseudo-random choice of questions and variants in the exercise sheets. Vary this value if you reuse a sheet or exercise to avoid that people with the same identity get the same choices again.

opento this entry determines how long solutions for this sheet are accepted. The value must be in the format `hh:mm_dd.MM.yyyy`, with `hh` the hour in the range 00..23, `mm` the minutes in the range 00..23, `dd` the day of the month in the range 1..31, `MM` the month in the range 1..12 and `yyyy` the year. This time is interpreted as local time on the machine running the OKUSON server.

openfrom (optional, default is that the sheet is open immediately) entry with the same syntax as `opento`. If it is set to a time in the future then participants of the course cannot get the sheet or submit solutions for it (But the administrator can, so you can check and test a sheet before the course participants can see it.)

maxhomescore (optional, default empty) entry specifying the maximal number of points a participant can achieve in his written homework exercises on this sheet. This is used only for statistical purposes.

starhomescore (optional, default empty) entry specifying the number of `optional` points among all points (see `maxhomescore`). "Optional" points are not considered not mandatory and might be given for particularly difficult exercises. This is used only for statistical purposes.

starmcscore (optional, default empty) same as `starhomescore`, but for online exercises.

The content of a `SHEET` element is a sequence of any number of `TEXT`, `EXERCISE` and `INCLUDE` elements.

The `EXERCISE` elements are empty elements with some attributes:

key the value must be the same as for a `key` attribute of some exercise read before by the OKUSON server.

prefix (optional, default value is the empty string) only matches an exercise with the correct `key` which was loaded with the given prefix. (This allows the use of several exercise directories, which may contain exercises with the same keys.)

nrquestions (optional, the default is the number of questions available in the exercise) the number of questions from this exercise which should be pseudo-randomly chosen for each sheet (there can be more questions in the exercise).

order (optional, default value is `"p"`) the value can be either `"p"` which means that the pseudo-random choice for the exercise sheets will permute the given questions; or the value can be `"f"` which means that the ordering of questions is fixed such that the questions will appear in the given order on each sheet.

The INCLUDE elements are also empty and have an attribute file whose value is interpreted as name of a file containing a text exercise. There is also the optional attribute prefix with the same meaning and default as for EXERCISE elements.

The TEXT elements contain LaTeX code of text which is included in the exercise sheets before, between or after the specified exercises.

In rare cases a page break for sheets with several pages occurs in an ugly looking position. Then you can use a

```
<TEXT>\mbox{}\newpage</TEXT>
```

to force a page break between certain exercises.

Specifying Several Sheets in one File

On can put several SHEET elements in one .bla file, but these must be enclosed in a SHEETS begin and end tag, and the DOCTYPE declaration must specify that a SHEETS element is the top level element in the document.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE SHEETS SYSTEM "sheet.dtd">

<SHEETS>

<SHEET>
  ...
</SHEET>
...

</SHEETS>
```

Checking Sheets

The OKUSON package contains a utility script testsheet for checking sheet files. It produces a LaTeX file and a PDF file of the given sheets containing all questions in all variants, and containing a list of solutions of all interactive exercises. The layout of the exercises is exactly as it should look in the PDF versions of the exercise sheets of the course participants.

It may be useful to check the individual exercises first with testexercise, see 5.3.

The syntax for this utility is

```
$OKUSONHOME/testsheet file1.bla file2.bla ...
```

For a sheet in <file>.bla with name <name> a LaTeX file <file>_<name>.tex and a PDF file <file>_<name>.pdf are generated, if no error occurs. These contain all variants of all questions and on an extra page the solutions for the interactive exercises.

Chapter 6

The Web Pages

We hope that the customization of the OKUSON package for your application is possible by changing the basic entries in the `Config.xml` file and by adjusting the template files for the web pages delivered by the OKUSON server. In this chapter we describe how the web pages work. If you need further changes, please tell the OKUSON authors - maybe there are sensible generic extensions of the package which fulfill your wishes.

6.1 General Format of the Web Pages

The web pages delivered by the OKUSON server should be valid with respect to a specification from the W3C consortium which is called `XHTML 1.0 Strict`. These are combined with `CSS 2.0` style sheets which are also specified by that consortium, details can be found in `http://www.w3c.org`.

Furthermore, we avoid the use of client side programs like `java` or `javascript`, plugins or other features of web browsers which some people tend to switch off.

This may sound quite tedious and restrictive on a first glance, but we see several advantages:

- Participants of the course need very little prerequisites, which are nowadays available on essentially any computer.
- Strictly following some official standard should guarantee a predictable and good appearance of the pages on all current web browsers and all operating systems.
- Having the page layout information (fonts, font-sizes, colors, border sizes, ...) in style sheet files makes it easy to give all pages a uniform appearance and to change this appearance for all pages at once.
- Using the XML variant of the released HTML specifications allows to use standard tools for processing and checking such files (see chapter 4). Actually, most pages delivered by

an OKUSON server are dynamically created from templates on demand. The templates are also well-formed XML documents, which allow an efficient preparsing and processing.

- There are developments like MathML which will allow in the future to include mathematical formulae directly in web pages (currently, we use images for that). Also, a systematic processing of XML documents on the client side is already possible in new web browsers. Such extensions will only work in combination with HTML pages which strictly conform to some official specification.

- We can include in our built-in web server an automatic check of the validity of all delivered web pages. (Non valid pages are still delivered, but they are saved in temporary files for debugging and the validator icon is removed.)

The OKUSON package comes with a complete set of sample web pages, they are in the subdirectory `html.sample` (German) and `html.english` (English). Of course, the text of web pages is language dependent. It should be straightforward to produce versions in other languages.

If you want German or English pages for your course, just copy all the sample pages to the web directory given in your configuration file. Probably only minor adjustments will be needed for your course, see the next section for more details.

All sample pages are provided as template files with extension `.tpl`. These are well-formed XML documents and almost `XHTML`, except for some extra tags.

The interface between the web pages and the OKUSON server is described by the meaning of these extra tags and by the names and meaning of input fields which are used in input forms for the course participants and for the administrator.

6.2 Delivering Static Files

You can deliver other types of files via the OKUSON server, for example PDF- or postscript-files, static HTML-files and so on. To achieve this just copy them somewhere below your web page directory (the subdirectory `html` of your OKUSON directory in the standard configuration). The only condition is that the OKUSON server knows the document type corresponding to the extension of the file name. (You can find the predefined file extensions in `server/fmTools/BuiltinWebServer.py`, search for `TypeDict`.)

Example: If you use the standard configuration and have a PDF-file `html/Solutions1.pdf` then OKUSON will deliver this file under the address

`http://my.computer.org:8000/Solutions1.pdf` and tell the browser

that this is a PDF-file.

6.3 Password Protected Files

You can restrict access to certain files to registered users. For this mention the protected files, one per line, in `Config.xml` within the element `ProtectedFiles`. These protected files can be accessed via a form which is demonstrated in the file `html/protected.tpl`.

6.4 How to Customize the Web Pages

All web pages delivered by the OKUSON server are created by filling in template files (with extension `.tpl`) dynamically.

The placeholders in the `.tpl`-files are easy to detect: While all element names in XHTML are written with lower case letters, the OKUSON placeholders are XML elements whose names start with capital letters.

There are placeholder elements with a simple global substitution text, and others which denote some text that depends on a sheet number or on personal data of a course participant. If you browse through the template files distributed with OKUSON you can probably often guess the meaning of such an element from its name and location. So, we are not going to describe everything in too much detail here, but try to give hints how to achieve likely adjustments of the pages.

6.4.1 What You Can Change Without Problems

- The style sheet files with `.css` extension to change colors, fonts, font sizes, margins, ...
- All texts in the pages.
- In particular the top page `index.html` could be redesigned, but you clearly want to include links to the main functions of the server: registration, change of registration data, getting the exercise sheets and result information.
- The `favicon.ico` icon can be substituted by another one.

6.4.2 What You Should Not Change

- The component names in the HTML forms, these names explain the meaning of the input data to the OKUSON server. But you can take out or add some components in these forms as explained below.

- The names of many template files. In particular those in the `errors` and `messages` directories, these files are used by the OKUSON server for error and success messages, as well as some others which are used for the output of internal functions in the server (`sheet`, `result`, `reg...`, `group...`, `admin...`, `exquery`). If you change other file names make sure that you change links to these pages accordingly.

6.4.3 Globally Defined Elements for Use in All Web Pages

The following elements appear in almost all `.tpl` template files. They are substituted by the content of the elements of the same name in your configuration file:

`<CourseName/>`, `<Semester/>`, `<Lecturer/>`, and `<Feedback/>`.

For customization you can use elements of form `<ConfigData key="...">`, in the configuration file `$OKUSONHOME/Config.xml` and include their content in the web pages via tags `<ConfigData key="..." />`.

Furthermore the following elements are generally useful:

`<ValidatorIcon/>` which includes an image in the `images` directory with a link to the validating service of the W3C consortium (it is only included if a page *is* actually valid).

`<CurrentTime/>` which is substituted by a string describing the local time of the delivery of a page. To get this string in a local layout set the environment variable `LC_ALL` of the user starting the OKUSON server appropriately (say `de_DE` for German, see the `man` page of `locale` on your system, often `locale -a` lists the available settings). It is possible to further customize the printing format via the element `<DateTimeFormat>` in your configuration file (by specifying an argument for the Python function `time.strftime`).

`<IfTime attr="spec"> ...some content...</IfTime>` The content of this element is only included in the page depending on the time specified by the given attributes. Possible values for `attr` are `from`, `to`, `after`, `before`, `on` with the obvious meanings, also several attributes can be given. The format of `spec` is a sequence of digits in the order YYYYMMDDhhmmss and arbitrary non-digits in between which are ignored; it is possible to leave out trailing digits. (Example: to include a paragraph on a page only between 6am on 12 Jan 2019 and 5pm on 14 Jan 2019 use:

```
<IfTime from="2019011206" to="2019-01-14, 17:00:00">
<p>Temporary paragraph</p>
</IfTime>
```

`<IfFileExists path="some/file"> ...some content...</IfFileExists>` The content of this element is only included in the page if the file given in the `path` attribute exists (relativ to the document root of the server).

A few other elements are available on all pages, but they are only sensible in a special context, we list them shortly, see the sample web pages how they are used. The elements with `Group` as part of their name correspond to a distribution of the course participants into smaller groups for tutoring. If you do not have such groups, ignore these elements. See 8.2 for more details.

`<GroupSize number="??"/>` number of participants in group with given number.

`<GroupDistribution/>` sequence of HTML table rows (ID, number of group), sorted numerically or alphabetically by IDs.

`<GroupsOverview components="..." nodisplay="..."/>` sequence of HTML table rows, one for each group.

See the comment in the sample file `groupoverview.tpl` and 8.2 for possible `components`. The optional attribute `nodisplay` must be a comma separated list of group numbers. The given groups are left out of the display. For examples `nodisplay="0"` can be used to suppress the default group `0` in the overview.

`<MembersOfGroup number="??"/>` comma separated list of IDs of participants in group with given number.

`<AvailableSheetsAsButtons/>` row of buttons for all available sheets, for use on query page for sheets.

`<AvailableResolutions/>` a `select` environment showing the resolutions configured in entry `<Resolutions>` of the `Config.xml` file.

`<IfIndividualSheets>...</IfIndividualSheets>` content is only typeset if the option `<IndividualSheets>` in the `Config.xml` file is set to `1`.

`<IfNoIndividualSheets>...</IfNoIndividualSheets>` content is only typeset if the option `<IndividualSheets>` in the `Config.xml` file is set to `0`.

`<IfExamRegistered nr="X">...</IfExamRegistered>` content is only typeset if the participant has registered for the exam with number X.

`<IfNotExamRegistered nr="X">...</IfNotExamRegistered>` content is only typeset if the participant has not registered for the exam with number X.

`<IfHTML>...</IfHTML>` and

`<IfMathJax>...</IfMathJax>` only useful in `html/sheet.tpl`; the first is for the display of sheets with images for the exercise texts and the second for sheets with exercises in HTML coding with MathJax.

6.4.4 Special Elements in Pages Containing Personal Data

There are two pages for registration, one for the actual registration and one for changing the personal data, see the sample pages `registration.tpl` and `regchange2.tpl`. Another page with personal data is the results page, see the sample page `results.tpl`.

Registration and Change of Data

The following registration data for each participant are stored by the OKUSON server: a user id, last name, first name, the number of semesters the participant has studied, (main) topic of studies, password (has to be given two times when it is first specified or changed), email address, wishlist of other ids for distribution into tutoring groups and a collection of further data for customization.

Each of these fields is connected with three names used in the web pages. The first is a name of the input field in the registration form, the second is an XML element name of a placeholder for the input field code with the known value as default (for changing the registration data) and the third is an XML element name of a placeholder for the value itself.

Here is a table giving these names.

description	form input name	input element placeholder	value placeholder
user id	id	`<HiddenIdField/>`	`<IdOfPerson/>`
last name	lname	`<LastNameField/>`	`<LastName/>`
first name	fname	`<FirstNameField/>`	`<FirstName/>`
semester	sem	`<SemesterField/>`	`<Sem/>`
group number	groupnr	`<GroupField/>`	`<Group/>`
group number	groupnr	`<GroupSelection/>`	`<Group/>`
main topic	stud	`<TopicField/>`	`<Topic/>`
password	passwd and passwd2	-	-
email	email	`<EmailField/>`	`<Email/>`
wishlist	wishes	`<WishesField/>`	`<Wishes/>`
custom data	persondata.xxx	`<PersonDataField key="xxx"/>`	`<PersonData key="xxx"/>`

In the last row the `xxx` can be substituted by any string. If a web request contains several values of a key `persondata.xxx` then these values will be collected as semicolon separated strings.

Concerning the main topic of studies, one can suggest some common entries which are specified in the configuration file, see the element `PossibleStudies` in the config file and the sample registration web pages.

There are further elements which can be used with the customization data components in case these should take only a fixed set of values:

```
<PersonDataRadioButton name="persondata.xxx" value="myvalue" />,
<PersonDataCheckBox name="persondata.xxx" value="myvalue" /> and
<PersonDataSelectOption name="persondata.xxx" value="myvalue"
      content="mycontent"/>,
```

The first two produce an HTML `<input>` field of type radio button or check box, respectively, which has the `checked` attribute set if the currently stored value of `persondata.xxx` is `myvalue`. Note that OKUSON stores only one value for each component name. Instead of using a component with several input elements use several components which can be switched on and off independently.

Similarly, `PersonDataSelectOption` produces an `<option>` element for use with `<select>`. Its content is `mycontent`, its `value` attribute is `myvalue` and its `selected` attribute is set if the currently stored value is `myvalue`. In this case the `content` attribute is optional and defaults to being the same as `value`.

Results Page

On the results page with template `html/results.tpl` there are also the following elements which produce personal data:

`<Results/>` this is substituted by a sequence of HTML table rows, one for each exercise sheet. The default entries of each row are the sheet name, the number of points in the interactive exercises and the number of points in the homework exercises. This can be customized by an attribute `components`, for example if you don't have one of the exercise types on your sheets. The default corresponds to

```
<Results components="interactive,homework" />,
```

the order of results points could be exchanged, or one of the exercise types can be left out. (Adjust the table header in the template appropriately.)

There are two more options `withMaxMCScore` and `withMaxHomeScore` which lead to the display of the maximal possible score after the score in brackets.

`<TotalScore/>` the total sum of all points in the interactive and homework exercises.

`<MaxTotalScore/>` the total sum of all maximal possible points in the interactive and homework exercises.

`<TotalMCScore/>` the total sum of all points in the interactive exercises.

`<MaxTotalMCScore/>` the total sum of all maximal possible points in the interactive exercises.

`<TotalHomeScore/>` the total sum of all points in the homework exercises.

`<MaxTotalHomeScore/>` the total sum of all maximal possible points in the homework exercises.

`<ExamRegStatus nr="??"/>` the information, whether a participant has registered for exam number ??.

<ExamGrade nr="??"/>, <ExamGrades/> the first is substituted by a result string of a function given in the entry `ExamGradingFunction`, which is activated by `ExamGradingActive`, in `Config.xml`, the function is called with the given exam number as argument. The second element is equivalent to a sequence of `ExamGrade` elements for all exams. See chapter 9 for more details.

<Grade/> this works similar to `ExamGrade` but is for the final grading of the course. See the configuration entries `GradingFunction` and `GradingActive` in the `Config.xml` file and chapter 10 for more details.

<GeneralMessages/> this is substituted by the content of the file configured in the entry `<GeneralMessageFile>` of `Config.xml` (`data/generalmessage.txt` by default).

<PrivateMessages/> this is substituted by message lines collected for the specific participant in the file given by the entry `MessageFile` in `Config.xml` (default is `data/messages.txt`). The format of this file is described in 11.5. You can add and remove individual messages via the administrator menu, see 7.1.4.

6.4.5 Special Elements for Sheet Specific Data

There are some special elements used with the sheets in HTML format.

<SheetNumber/> number of sheet.

<SheetName/> name of sheet.

<IfOpen>...</IfOpen> content only included if sheet is still open (or if the administrator overruled with his password).

<IfClosed>...</IfClosed> content only included if sheet is no longer open (and administrator did not overrule).

<HiddenIdOfPerson/> and <HiddenNameOfSheet/> hidden input fields for submission form.

<WebSheetTable/> the actual table rows for the sheet, entries are images with exercise texts, includes input fields for submission of interactive exercises.

<OpenTo/> and <OpenFrom/> closing and opening times of sheet in readable format (same format as explained for `<CurrentTime/>`).

6.4.6 Special Elements for Tutoring Group Specific Pages

All information about tutoring groups is also exported for use on web pages related to one such group. In 8.3 it is explained how this information can be given to the OKUSON server.

<GroupNumber/> the number of the group.

<GroupTutor/> the name of the tutor of the group.

<GroupPlace/> place where the group meets.

<GroupTime/> time when the group meets.

<GroupEmailTutor/> the email address of the tutor of the group.

<GroupIDs/> a comma separated sorted list (numerically or alphabetically) of the IDs of the participants.

<GroupData key="..."/> further unspecified data for customization.

6.4.7 Special Elements for Administration Pages

The administration and tutor pages use templates similar to the other pages described so far. But since there is little need to adjust the administration pages (explained in chapter 7), we do not document the helper elements here. They are almost self explaining or else look in the code. If you want more functions in the admin menu, please, tell us about them. Maybe they are of general interest and we want to add them to the OKUSON package.

6.5 MathJax support

It is possible to deliver exercise sheets where the exercise texts are encoded in HTML and such that the formulae are rendered by MathJax (`https://www.mathjax.org/`). To enable this feature you just need to offer it to the participants in the file `html/exquery.tpl`, search for the string "MathJax" and uncomment as you wish.

This feature can only be used with graphical browsers with JavaScript enabled.

To make it work you need the programs `iconv` and `pandoc` on the machine that runs the OKUSON server. (The first is probably there, the second can probably be installed via the package manager of your linux distribution.)

The advantage of this alternative to display the exercise sheets is that the exercise texts can flow according to the current display, in particular this is a first step to make the sheet pages more readable on mobile devices with small screen.

The disadvantage is that you can no longer use almost arbitrary LaTeX in your exercise texts. You need to check carefully for all exercises if `pandoc` translates it to sensible HTML code, and if not you need to change your exercise texts.

For the OKUSON author's exercises this doesn't seem difficult because except for maths formulae only a few LaTeX constructs are used (simple `itemize`, `enumerate` and `description` environment, simple macros like `\emph`, `\textrm`, umlauts are written in latin1 encoding).

You can use your own macros, but have to define them in the `html/sheet.tpl` template, there are some examples given.

Feedback on this feature is welcome, also hints to a more flexible LaTeX to HTML converter program.

Chapter 7

Administration via the Web Interface

The configuration of the OKUSON system and the creation of exercises and sheets works with an ASCII text editor. For some other administrative tasks however, the OKUSON server offers a convenient web interface. You can reach this by pointing your browser to the following URL, when the OKUSON server is running:

`http://localhost:8000/adminmenu.html`

This assumes, that the OKUSON server is running on the same machine than your web browser. If this is not the case, you have to substitute the name of the server machine for "`localhost`". The same applies for the port number `8000`, if you have configured your OKUSON server to listen to another port.

Note that there is an extra configuration option `AdministrationAccessList` to limit the IP range of machines, from where the administration pages can be accessed (see the comment in the `Config.xml` sample file). The default setting is to allow administrator access only from the local machine (`localhost`).

In addition you have to authenticate yourself to the OKUSON server for every administrator operation. You can do this by entering the administrator password every time you start an operation. To make live a bit easier we offer a cookie-based login procedure for this purpose to avoid repeated password input. This works as follows:

To log in you have to visit the URL

`http://localhost:8000/adminlogin.html`

and type the administrator password. After login you are led to the administrator menu, however, a secret number for this login session is stored in your browser in a cookie. As long as the OKUSON server is not terminated and your browser still has this cookie value and you do not login anew, you can perform administrative tasks without further authentication. There is an option at the top of the administrator menu to log out again.

Note that the communication between your browser and the OKUSON server is not encrypted, such that everybody listening along the way could possibly get administrator access. Therefore

you should limit the administrator access via the abovementioned IP ranges and log out after you are done with administrative tasks.

7.1 Administrative Tasks in the Administrator Menu

You have the following options in the administrator menu:

7.1.1 Restart server

This stops the OKUSON server gracefully and restarts it immediately. This is the same as launching the `restart` script in the OKUSON home directory. Note that you have to log in again for further administrative tasks, as the login session is terminated automatically.

7.1.2 Shutdown server

This stops the OKUSON server gracefully. This is the same as launching the `stop` script in the OKUSON home directory.

7.1.3 Display available and future sheets

This option gives the administrator access to a page to display sheets exactly as the page for regular participants, except, that also future sheets (which have not yet reached their `openfrom` date) are accessible.

7.1.4 Send message

With this option one can send an individual message to a certain participant. This message will appear on the page where the participant queries his results. More than one message is possible. All messages are stored in the file `data/messages.txt`. The format of this file is described in section 11.5.

This messaging system can also be used to automatically produce individual messages for all participants. In this case it is probably better not to type in the messages via the web interface, but to append the messages directly to the file `data/messages.txt`.

Remember the possibility to display a general message on the result pages of all participants via the file `data/generalmessage.txt` (see section 11.6).

7.1.5 Delete messages of

With this option you can delete a subset of the private messages of one participant. To this purpose one gets a display of all messages and can choose the subset to delete. Note that this deletion is not really a deletion but more a "revocation". A "deleted" message is just repeated as a new message with a dollar sign $ prepended in the internal data format. The participant of course does no longer see revocated messages. In this way, no message is ever lost.

7.1.6 Reevaluate participants' answers for sheet

This options comes in handy if the unfortunate case happens (and it will happen eventually, believe it or not!) that a "correct" solution was entered incorrectly in the exercise description. One has to know that the evaluation of the submission of a participant happens at the time of submission and is then cached in memory (and on disk). Therefore later changes in the correct solutions in the exercise source files are **not** considered without manual intervention!

To this end, one can select this option and reevaluate all submissions of all participants for a certain sheet.

7.1.7 Show Exercise Statistics for sheet

The code for this and the following four statistic functions was kindly contributed by Thorsten Heck and Ingo Klöcker from Lehrstuhl A für Mathematik, RWTH Aachen.

With this option you can display statistics about numbers of participants. For every variant of questions the number of participants who have got this variant, the number of participants who answered the question, and the number of participants who answered the question correctly is shown with a nice graphical display. Color coding is used to mark extreme cases.

7.1.8 Show Global Statistics

With this option and the next one you can display statistics about the distribution of scores, separated by tutoring group and sheet, and by homework and multiple choice. The distributions are shown as total numbers, as percentages, and as histograms.

7.1.9 Show Global Statistics, separated per Group, for sheet

See above.

7.1.10 Show Cumulated Score Statistics

With this option one can display statistics about cumulated scores of participants. This is interesting to track the success of participants during the semester. It can be restricted to one tutoring group or not. The distributions are shown as total numbers, as percentages, and as histograms.

7.1.11 Show Detailed Score Table

With this option one can display an overview over all participants in a tutoring group with all their scores.

Comment: The following options are for data exports. They all send a file with content type "`text/okuson`" and your browser probably will ask you to save the file, because it does not know what to do with this content type. In all cases we send sensible default names along.

Attention: If you choose in your browser some helper application to view these exports, it may very well be that the exported files are written into some global temporary directory like `/tmp`. It is of course your responsibility to ensure proper data protection for this at least potentially critical personal data!

7.1.12 Format string

This is a generic export function. Above this input field there is a description of what can be exported. Basically the format string is a prototype of the lines to be exported (one line for each participant) and percentage signs followed by single letters indicate data fields to be exported. Between these percentage expressions arbitrary text can be entered and will be exported exactly as entered.

This new generic export function probably makes all the following ones unnecessary.

Note that there is a generic sorting and selecting script (`sortselect.py`) in the `scripts` directory of the OKUSON distribution. It explains itself if it is called without arguments. Together with the UNIX standard utility `uniq` the generic export function and that script it should be possible to conduct nearly every statistical analysis necessary.

7.1.13 Export people for tutoring group distribution

With this option you can export data about all registered participants for the purpose of distributing them into smaller tutoring groups. Usually this will be a semi-automatic process that is described in chapter 8 below.

7.1.14 Export people

With this option you can export the personal data of all registered participants into a single ASCII file.

In the file exported there is — after some comment lines beginning with a #-character — one line for every participant. This line comes in the same format as the file `data/people.txt` (see 11.1), **except** that there is an additional field, namely the number of the group in which the person is.

This means, that there are three reasons, why you should **not** copy the output of this export directly to the file `data/people.txt`. Apart from making no sense at all, you have to remove the last data field with the group, and, what is probably worse, all colons and newlines that may be contained in personal data fields are deleted or replaced by a space respectively, whereas in `data/people.txt` special precautions are taken to store such data (see chapter 11).

So the fields in `peoplelist.txt` are separated by colons and arranged in the following order:

ID
last name
first name
semester number
studiengang
encrypted password
email address (possibly empty)
wishlist as typed in
custom personal data
group number (0 if not in any group)

You can choose among a number of orderings. There seems to be no immediate application of this export facility, but it offers easy access to the available personal data for private scripts and standard UNIX text tools.

Our intention for this export is to be able to access the current data in the server easily with private scripts.

7.1.15 Export participants of exam

This export facility is to get information about the registration situation for exams. Therefore the exported data is restricted to ID, name, first name and the time stamp of the registration. The file format is as follows: For every participant there is one line that contains the abovementioned fields, separated by colons. See chapter 9 for a description of how to organize exams conveniently.

7.1.16 Export results

This export facility produces information about all the results of the participants, including multiple choice exercises, homework exercises, and exams. The data format is as follows: There is one line for each participants with fields separated by colons. The fields and their order can be read off the following table:

id
last name
first name
group number
total score in multiple choice exercises
total score in written homework exercises
total score
generated message from automatic grading function
grade from automatic grading function
a string of all exam results and grades, separated by semicolons
for every sheet one more field, see below

Note that the fields generated by automatic grading are there but empty, if automatic grading is switched off. For each exam the participant has a data field, there are two parts of information exported (separated by semicolons): the score in the exam and the grade, as calculated by the automatic grading function for exams. If a participant has a data field for some exam (for example if he has registered for this exam), but did not take part, the string "`-;0`" is exported as his result for that exam. If automatic grading of exams is switched off, the grade exported is always 0. So the exams data field is an alternating list of entries, all separated by semicolons, for each exam an entry for the score, followed by an entry for the grade. At the end of the line, there is for every closed sheet a field with sheetname, multiple choice score and homework score, separated by semicolons.

The obvious application of this export is at the end of the semester to produce certificates for successful participants.

Chapter 8

Managing Participants

8.1 Registration of Participants

With respect to registration there are two entries in the `Config.xml` configuration file. If the content of `<RegistrationPossible>` is `1` then participants can register themselves. If it is `0` further participants can only be registered by the administrator. By default, participants can later change all the data given during registration (except for their ID). Using the element `<KeptData>` the change of certain data can be disallowed, see the comments in `Config.xml` for details. This can be useful if data were imported from other source (e.g., the university administration).

Once you start the server and registration is allowed, participants can register via the OKUSON web interface without any further administrational interaction.

Among the sample web pages coming with OKUSON the template files `registration.tpl` and `regchange2.tpl` are used for the initial registration and change of personal data, respectively.

Before you start the service check if the personal data collected on these pages are what you need in case of your course. It is advisable to ask for all data you may need later, e.g., for the grading and a possible certificate in the end of the course. In chapter 6 we have explained the customization of the web pages, in particular in section 6.4.4 we mentioned some customization variables you can use for additional data. On the other hand you can just delete input fields in the registration pages which you do not need.

The most important datum of a registration is the ID of the participant, which cannot be changed later. All data concerning a specific course participant are stored by OKUSON together with the corresponding ID.

IDs of Participants

In our university there is a natural ID for each student (the "Matrikel" number) which is often used. If you want to use other types of IDs, e.g., login names of the students choice, you can customize the configuration option `IdCheckRegExp` in `Config.xml`.

Visitor IDs

Sometimes it is useful to have some visitor IDs which can be used with arbitrary password and which do not appear in the export of user data or result statistics. OKUSON has the configuration option `GuestIdRegExp` for specifying such IDs.

Registration using external data

The `scripts` subdirectory of OKUSON contains a script `RWTHCampus2people.py`. This does the registration of participants using data which come from the university administration (the Bachelor students have to officially sign in for all their courses, we want to use the data and avoid a double registration). We use the `<KeptData>` entry in `Config.xml` to disallow the change of imported data.

While this is a specific script for usage at RWTH Aachen slight variations of it may be useful in other places.

Registration with validation

If you have problems with spam registrations there is a possibility to add a validation step to the registration. In that case users have to submit a valid email address during registration. To activate this

- change the value of the `<ValidateRegistration>` element in the `Config.xml` file to 1,
- adjust the value of the `<ValidateRegistrationMail>` element in `Config.xml` (name of lecture and URL to the OKUSON server),
- change the text in `html/registration.tpl` to make clear that a valid email-address is needed for validation of the registration.
- maybe further restrict the email addresses allowed for validated registration by specifying a regular expression in the `<ValidEmailAddresses>` element of `Config.xml`.

After restart of the server, users can only pre-register via the registration form. They then get an email with a validation link. If they click that link or copy it to their browser, the registration will be finished.

A corresponding temporary file (by default `data/people.txt.tmp`) can be removed from time to time if it was not accessed for a while.

8.2 Distributing Participants into Tutoring Groups

We always accompany our exercises with regular meetings in a number of smaller groups with an assigned tutor. (The tutors check and grade the written solutions of non-interactive exercises, and they discuss all exercises with their group.)

You can tell the OKUSON server about the membership of a course participant in a tutoring group by numbering the groups (starting from 1, the number 0 is a default number for people not (yet) put in one of those groups), and by appending a line of the form

```
someid:groupnumber
```

to the file `data/groups.txt` (more precisely, the file configured by `GroupFile`).

But, as you may know, for bigger courses, this distribution of the participants into tutoring groups is not easy, at least if you want to fulfill wishes of participants to be together with some others. OKUSON helps you to complete this task in the following ways:

First (already at registration time), participants can enter a wishlist of other students' IDs with whom they want to be in the same group. OKUSON offers one input field, where participants can basically type in IDs of other participants, usually separated by commas or whitespace.

Secondly you can export all registered people via the administrator pages (see 7) using the button "Export people for tutoring group distribution". You have the choice of exporting all participants together (option "all together") or separated by their course of studies. Also you can choose the data fields by which the output is sorted. The result of this export is an ASCII file in a certain format (see 8.2.1), that contains basically the IDs, the first and last names, the semester number, the course of studies, the wishlist and possible additional data you may collect of each participant.

This file is sent to your web browser with the content type `text/okuson`. This usually means that your browser — not knowing this type — asks you, where it should store this file. It should offer the default file name `peoplelistforgroups.txt`. Once you have saved this file on your hard disk, you can use the following two scripts in the `scripts` directory of your OKUSON home directory: `distribute.py` and `numbergroups.py`. The first does a distribution of participants into groups thereby fulfilling the wishes of participants as far as possible and forming as many groups as you order. The second script just takes the result and brings it in a format suitable to reimport it into OKUSON. The result of `numbergroups.py` can be appended to `data/groups.txt`. After the next server restart the new information is available to the OKUSON server.

8.2.1 Usage of `distribute.py`

The input to `distribute.py` is an ASCII file with one line for each participant (plus extra lines, see below). Each such line has to be in the following format:

```
id:last name:first name:semester:studiengang:wishlist:persondata
```

where `wishlist` is a comma separated list of valid IDs and the `persondata` in the end means a string describing the customization data (as comma separated strings of concatenated key-value pairs `xxx,xxxval`). Lines beginning with a hash # character are ignored as comments. Empty lines play a special role, they separate so called "parts" of the input. If for example you want to form groups for participants of some course of studies separately, you can just separate the lists of participants by empty lines, which is also done by the OKUSON server, if you select the option "by course of studies" during export. Via the sorting options it is also easy to split into parts by other criteria.

The output of `distribute.py` has the same format, except that empty lines now separate the groups into which the script has divided the participants. Some comment lines have been added.

What does `distribute.py` do?

First it calculates for each part the finest equivalence relation with the property, that all wishes (within the part) are fulfilled in the sense, that any two participants where one had the other on his wishlist, are in the same equivalence class.

Then it distributes these equivalence classes into a number of groups you have specified on the command line (see below). To this end it uses the following simple algorithm: It always puts the biggest equivalence class left into the smallest group available. This algorithm works amazingly well, because usually you will have lots of people without wishes, which help to fill the groups in the end. Occasionally there will be one equivalence class which is too big, thereby making one exercises class which is bigger than all the others. See below how to overcome this problem.

`distribute.py` is called with the following command line arguments:

```
scripts/distribute.py INPUTFILE OUTPUTFILE GROUPS {GROUPS}
```

where `INPUTFILE` is replaced with the name of the input file, `OUTPUTFILE` is replaced by the name of the output file and these two file names are followed by as many numbers as there are parts (separated by empty lines, see above) in the input. Each number specifies, how many groups should be formed out of the corresponding part.

Note that the output file is overwritten and therefore should be different from the input file.

The information about the equivalence classes is preserved in the following way: between any two equivalence classes within the same group a comment line is added, indicating the delimiter between equivalence classes.

8.2.2 Usage of `numbergroups.py`

The usage of `numbergroups.py` is even simpler. It is called in the following way:

```
scripts/numbergroups.py INPUTFILE OUTPUTFILE [FIRSTGROUP]
```

where `INPUTFILE` is replaced with the name of the input file, `OUTPUTFILE` is replaced by the name of the output file, and `FIRSTGROUP` is an optional argument which selects the number of the first group and defaults to 1.

The script `numbergroups.py` just reads in the input file, reads empty lines as group delimiters, ignores comments and writes out the group distribution in the format the OKUSON server needs it in the file `data/groups.txt`. The import works just by appending the result of `numbergroups.py` to the file `data/groups.txt` and restarting the server.

8.2.3 Strategies for Distribution

We found that often the following strategy was good enough:

1. Export people "all together".
2. Run `distribute.py` once with the number of groups you want to have.
3. Look at the output on the screen and decide whether the group sizes are suitable.
4. Run `numbergroups.py` to prepare the input for OKUSON.
5. Append it to `data/groups.txt` and restart the server.

Because of the nature of this procedure with many intermediate steps you can do manual interventions at all stages. For example you can divide the output of the OKUSON server by hand in a number of parts which are handled separatedly. Or you can play around with the number of groups (note that using 1 for the number of groups basically gives you statistics about the equivalence classes). In this way, you can overcome problems with too big equivalence classes by manually separating them into different parts. At last you also can look at the output of `distribute.py` and change the distribution before importing it into the OKUSON server via `numbergroups.py`.

Note finally that you can also separate different parts of the input into different files manually and run the scripts on these files separately. In the end you can put everything together by choosing the `FIRSTGROUP` argument of `numbergroups.py` accordingly.

We hope that this whole procedure is flexible enough for all situations.

8.3 Importing Information About the Tutoring Groups

If you are using the distribution of the course participants into tutoring groups, there are pages among the OKUSON sample web pages for publishing information on each group: number, meeting place and time, tutor, contact address and also further infos that can be freely customized, see the templates `groupoverview.tpl` and `groupinfo.tpl`. There is also an encrypted password for each group. This can be used (and changed) by the groups tutor for sending grading results of non-interactive homework exercises to the OKUSON server, see below. The placeholder elements in these template files are described in 6.4.6.

The import of these data for the existing groups is by appending one line per group to the text file `data/groupinfo.txt` (more precisely, the file configured in entry `GroupInfoFile`). Each such line is a sequence of entries separated by colons `:`. The entries are interpreted in the following ordering:

number number of the group, a positive number.

passwd encrypted password for the tutors input access.

tutor name of tutor.

place place of group meeting.

time time of group meeting.

emailtutor email address of tutor.

maxsize maximal number of participants in group.

groupdata additional entries for customization.

Note that you could give pieces of HTML code containing appropriate links as some of these entries. Note also that entries are separated by colons, within the entries you can encode a colon by typing `\d` instead.

The second last entry comes into play, if one uses the feature, that participants can choose the number of their tutoring group during registration (see the explanations in the `Config.xml` file).

The last entry *groupdata* must be specified as comma separated list of concatenated key-value pairs. Example: you can store that the lecture hall for a group has a blackboard and no beamer by an entry like:

```
...:blackboard,yes (2 big ones),beamer,no
```

Such information can be used in the template web pages via tags

`<GroupData key="blackboard"/>` and `<GroupData key="beamer"/>`.

8.4 Input of Homework Results by Tutors

There is a web page `/tutors.html` where tutors can request input forms for homework exercise results, either for a particular member of the group and all sheets, or for a particular sheet and all members of the group. The group's password is needed for accessing these forms. This password can also be changed via this request page.

Each result must be given in form of a total score and (maybe optionally) a comma separated list of partial results. You may specify the optional `maxhomescore` attribute for your exercise sheets for later use with grading or statistics.

The above mentioned input page allows tutors only to enter points for participants in their own group. If this is too hard a restriction for your purposes, you should consider using the free input form which is available under `/HomeworkFree`. It allows not only a less redundant input in case you have several participants handing in one sheet as teamwork but may also be configured such that every tutor is able to enter points for any participant of the course. In order to keep the system "safe", this feature is not enabled by default; if you want to use it, be sure to change the `RestrictToOwnGroup` value to `no` in the main configuration file.

The free input form presents several rows of input fields. The fields in the first column accept comma-separated lists of ID's (in case one sheet has been handed in as teamwork) or one single ID, the following two columns for entering points are the same as in all other such input forms.

After submitting the form, the entered data is checked. If any error occurs (unknown sheet number, ID, etc.), the wrong entry is marked by a red star and the tutor is asked to correct it. Otherwise, the form is displayed once again (without any editing possibilities) and the tutor is asked to confirm the data. Following the confirmation, the data is stored in the system.

Chapter 9

Managing Exams

Unfortunately, we have no idea how OKUSON can help with checking and grading written exam exercises.

Nevertheless, there is functionality for the administrational aspect of exams, this is explained in this chapter: Registration for exams, importing exam results in the OKUSON server, displaying the results.

OKUSON handles some information about exams for each participant. Exams are numbered from 0 to a certain limit, which is 23 at the moment. The information stored is basically the fact, whether a certain person wants to take part in the exam, and if so, the result in the form of one non-negative integer score as the final result and an additional string, which can be anything, for example intermediate results. The OKUSON system will only store this string and will not further process it, except the user supplies functions to do so (see below).

The result of exams can enter the automatic grading decision at the end of the course (see 10).

9.1 Registration for Exams

There is already a web page prepared to organize the registration of participants for exercises. It is stored in the template "`examregistration.tpl`" in the root directory of the web pages (usually "`html`" in the OKUSON home directory).

In the default setting this page is not linked from anywhere else, such that participants will not visit this site accidentally. Note however, that one can reach this page by typing in the correct URL right from the beginning of the course. Therefore the form for submission of the registration is commented out, such that it does not appear on the page. Of course, a participant knowing the details can submit registrations "by hand" anyway.

To activate the registration, one has to do the following:

- "Comment in" the form in `examregistration.tpl`.

- Edit the attribute value for "nr" in the input field in the line marked "***please edit***" to the exam number, for which you want registration.
- Delete the string "***please edit***" in the same line.
- Edit the text around this to your needs.
- Activate the menu entry "Klausuranmeldung" in the main menu in the file index.tpl.
- Restart the server.

From the moment you do the last step, participants can register (and deregister) for the exam in question.

The collected information is stored in the file examregistrations.txt in the data directory. However, you do not need to know its format, because there is export functionality for these registration information via the administrator menu (see section 7.1.15).

Please note again that even if you take the registration page from the web pages, people knowing the details can still submit registrations and deregistrations "by hand".

9.2 Importing Exam Results into the Server

The results of exams are put directly into the data file exams.txt in the data directory. See section 11.4 for the file format. This file will never be written by the OKUSON server. Therefore it is save to append data while the server is running. After a restart, everything is available within the OKUSON system.

Note that without further intervention, no result is displayed publicly. However, in the export option for results (see 7.1.16) the data appears.

9.3 Displaying Results of Exams automatically

To display the exam results on the usual result pages, there are two special configuration options. The basic idea is, that the administrator of the OKUSON server supplies a function that produces the output that should appear on the web page.

This works as follows: There is a configuration option ExamGradingActive. The automatic exam grading is active if and only if the value of this integer is non-zero. For this case, there is the configuration option ExamGradingFunction, which contains a Python function with the name "Grade" that is called with two arguments, the first being an object of type Person (see chapter 12), containing all the personal data of one participant, including information about exams (in the exams entry). The second argument is the number of the exam in question. This function is called when the result page for a participant is generated for all exams that participant took part in.

The `Grade` function has read access to the personal and exam data and has to return a pair. The first entry of this pair has to be a string, which is put in a "`<p> </p>`" environment on the web page, if you use the `<ExamGrades/>` element. The second entry is a grade, which is later exported in the result export. This grade has to be a number (int or float) and is converted into a string during export.

Note that of course you can refrain from using this feature altogether, produce the output for the results display externally with your own tools and import this into standard private messages in the file `message.txt` (compare sections 7.1.4 and 11.5).

Chapter 10

Automatic Grading

At the end of the semester there will usually be some kind of grading procedure. This can be as simple as deciding which participant passes the course and gets a certificate of participation ("Schein") or can be more involved like coming up with a "grade" of some sort.

Usually the data going into such decisions is known to the OKUSON system (scores from multiple choice exercises, scores in non-interactive exercises, results of exams). Therefore it seems natural to give the task of grading to the OKUSON system as well as giving out the information about their results to the participants.

There are two ways to do this: The first is using the automatic grading facilities of OKUSON described in this chapter and the second works by exporting all the result data as described in section 7.1.16, using external tools for the decision and reimporting the information back into the OKUSON system as private messages (see sections 7.1.4 and 11.5).

Due to the big variety of possible grading algorithms, one has to learn to write a little Python function to use automatic grading.

There are two configuration options for automatic grading: `GradingActive` and `GradingFunction`. The first is integer valued and the automatic grading functionality is switched on if and only if its integer value is non-zero.

Once automatic grading is switched on, the entry `GradingFunction` is used (the sample configuration file `Config.xml.sample` in the OKUSON distribution has an example function, which is shown below at the end of the chapter). It has to contain a Python function `Grade`, which takes 5 arguments: An object of type `Person` (see 12), then a list `sl` for the available sheets, the total score of the person in all multiple choice exercises, the total score in all written homework exercises, and finally a list of integers of length 24, where each number corresponds to one exam, numbered from 0 to 23. A value of 0 stands for either 0 points or for the fact, that this person did not take part in the exam. This list is for programmer's convenience only, as this information is contained in the personal data of the person anyway. There one can also read off, whether the person actually did take part in a certain exam or not.

The function has to return a pair, where the first entry is a string which is put between a "`<p>` `</p>`" on the results page, if the element `<Grade />` is present in the template file. The second

entry is a number (float or integer), that is stored as the grade. It is later exported along with the other results. You can for example use the grade as a flag, whether somebody passed the course or not.

With the automatic grading facility you can devise nearly arbitarily complicated conditions for the grading decision. You have read access to all personal data and the whole OKUSON system.

Note however that it is strictly forbidden to change any global data in the OKUSON server. The reason for this is that the server is multi-threaded, every web access has its own thread and any change in global data has to be protected by certain locking mechanisms to avoid possible internal data corruption. Bugs coming from such misuse will show up only occasionally and will be extremely hard to find!

Here is the example function from the example configuration file:

```
def Grade(p,sl,mcscore,homescore,exams):
    '''This function decides about the grade of person p.
       sl is a list of sheets as returned by the function
       SheetList in the Exercises module. mcscore is the
       total sum of points in all closed sheets with counts
       value equal to 1. homescore is the total sum of
       points in the homework and exams is a list of length
       at least 24 with one score corresponding to each of
       24 exams (indexed from 0 to 23). If the participant
       did not take part in an exam, there is a zero in the
       corresponding position.'''
    if (mcscore+homescore >= 240 and
        (exams[0]+exams[1] >= 50 or
         exams[2] >= 50 or
         exams[0]+exams[2] >= 50 or
         exams[1]+exams[2] >= 50)):
        return ('Sie bekommen den Schein.',1)
    else:
        return ('Sie bekommen leider keinen Schein.',0)
```

Chapter 11

File Formats

The data files of the OKUSON system all reside in the `data` directory in the standard configuration. They are all extended ASCII files (usually with the ISO-8859-1 8 bit encoding) and have the following standard format:

- Lines are separated by newline characters (ASCII code 10), i.e. UNIX line ends.
- Empty lines are ignored.
- Lines where the first non-whitespace character is a hash character "#" are treated as comments and are ignored.
- Every line that is not ignored stands for a data record.
- For each file there is a field separating character (usually a colon ":") and one can access all the fields in a line simply by splitting the line at all occurances of the separating character. If the value of a data field actually contains the separating character it is protected as described below.
- After reading an input line, the following procedure is applied to every data field (after the split):

replace	`\d`	by the delimiter
replace	`\n`	by a newline
replace	`\r`	by a carriage return
replace	`\c`	by a hash mark #
replace	`\e`	by a backslash

- Before writing an output line, the following procedure is applied to every data field (before joining fields with the separator in between):

replace	every backslash	by `\e`
replace	every delimiter	by `\d`
replace	every newline	by `\n`
replace	every hash character #	by `\c`
replace	every carriage return	by `\r`

- The last two points ensure that data lines contain delimiter characters only between fields and that backslashes are followed only by one of the characters "cdenr". Therefore, arbitrary string values can be put into data fields within one line.
- There is one exception for the empty list: It is stored as the special value "`\0`" (a backslash followed by a zero) to distinguish it from the list containing exactly one empty string.
- In every file one of the fields is the ID that identifies the person to which the data belongs.
- There is the following principle: New data is **always appended** to the file and **never deleted**. Therefore, at any given time, **the last value with a certain ID** is the one that counts. The reason for this is that appending is more efficient than writing the file anew and that it is guaranteed, that every piece of information in the memory of the server is also saved to disc at all times.

Whereas the format of the file makes it easy to use external standard UNIX tools to access all the information, the last principle somehow ruins this, because one always has to sort out the last valid value. We strongly recommend to use the export facilities described in sections 7.1.13 to 7.1.16 to access data, because they do not impose this difficulty.

However, we see certain applications (mainly data import) to access the files described in this chapter, therefore it is worthwhile to document the format.

Note that the fundamental idea of data management in the OKUSON system is, that data is read in at server startup from the files in the `data` directory. The data is then stored in internal data structures. As mentioned above, when more than one line in a file corresponds to the same ID, the last one counts. During the runtime of the server the data is kept in memory and on disc. Every single change is first appended to the data files on disc and then entered into the data structures in memory, such that consistency even after a server breakdown is guaranteed.

11.1 `data/people.txt`

The general comments about file formats at the beginning of this chapter apply also to `data/people.txt`. The delimiter character for this file is a colon ":".

Each line of the file corresponds to the personal data of one participant and contains the following data fields (in the third column some comments about the data type are displayed):

Name of field	Description	Data type
id	Identification	string
lname	Last name	string
fname	First name	string
semester	Number of semester	non-negative integer
studies	Studiengang	string
passwd	encrypted password	string as from `crypt`
email	email address	string
wishlist	ids of other people	string
persondata	custom personal data	string

11.2 data/groups.txt

The general comments about file formats at the beginning of this chapter apply also to `data/groups.txt`. The delimiter character for this file is a colon ":".

Each line of the file corresponds to the group membership data of one participant and contains the following data fields (in the third column some comments about the data type are displayed):

Name of field	Description	Data type
id	Identification	string
nr	Number of group the person is in	non-negative integer

11.3 data/groupinfo.txt

Note that usually you will have to edit this file by hand at the beginning of your course to set up the information about your tutoring groups. Therefore it is particularly important to document the format of this file.

The general comments about file formats at the beginning of this chapter apply also to `data/groupinfo.txt`. The delimiter character for this file is a colon ":". Remember to quote possible colons ":" by "`\d`" for example in time values like "17:30"!

Each line of the file corresponds to the group data of one tutoring group and contains the following data fields (in the third column some comments about the data type are displayed):

Name of field	Description	Data type
nr	Number of group	non-negative integer
passwd	Password for this group	string
tutor	Name of tutor	string
place	Place of group session	string
time	Time of group session (remember "\d" for ":"!)	string
email	Email address of tutor	string
maxsize	Maximal number of participants	non-negative integer
groupdata	Additional customization data	string

The format of the *groupdata* entry which encodes key-value pairs is explained in 8.3.

Note that in the case of this file the group number is the identifying field such that if more than one line with the same group number appears, only the latest counts.

11.4 `data/exams.txt`

Note that after an exam you will have to supply the content of this file from outside the OKUSON system. Usually you will have to type in the data from the external grading procedure of the exam. Therefore it is particularly important to document the format of this file.

The general comments about file formats at the beginning of this chapter apply also to `data/exams.txt`. The delimiter character for this file is a colon ":".

Each line of the file corresponds to the exam information of one participant for one exam and contains the following data fields (in the third column some comments about the data type are displayed):

Name of field	Description	Data type
id	Identification	string
examnr	Number of exam (zero based)	non-negative integer
totalscore	Total score of participant	non-negative integer
maxscore	Maximal score in this exam	non-negative integer
separatescore	A string to store scores from parts of the exam	string

The last field can be used for arbitrary purposes. Especially the automatic grading facilities of course have access to this information. The second last field is not used within the OKUSON system as of this writing.

11.5 data/messages.txt

The general comments about file formats at the beginning of this chapter apply also to `data/messages.txt`. There is however one exception, namely that all lines corresponding to a certain ID count, not only the last. Messages are therefore never deleted but only "revoked": If a message is repeated with a dollar sign "$" prepended, it will no longer be displayed on the result page of the participant.

The delimiter character for this file is a colon ":".

Each line of the file corresponds to one personal message for one participant and contains the following data fields (in the third column some comments about the data type are displayed):

Name of field	Description	Data type
id	Identification	string
msg	Personal message	string

11.6 data/generalmessage.txt

The format of this file is an exception. Its content is put into the result page instead of a `<GeneralMessages />` element in the template for the result page. The content of this file is copied one to one without any change. Therefore you have to put a valid XHTML 1.0 subtree into this file using the standard ISO-8859-1 ASCII encoding.

Chapter 12

Internal Data Structures

12.1 Overview and Introduction

This chapter is written to be read along with the files `Data.py` and `Exercises.py` of the OKUSON server's source code. Knowing the internal data structures of the OKUSON server requires a certain knowledge of the Python language, which we will assume throughout this chapter.

Having said this, we no longer have to explain certain things: For example, the names and types of components of objects in a certain class can readily be read off the well-documented source code. Also the behaviour and syntax for the basic Python objects like strings, integers, floats, lists, and dictionaries will not be explained here. Therefore we concentrate on the general structure and on the questions like "What is where in the source code?".

All data structures regarding people and tutoring groups are collected in the server's `Data` module and reside therefore in the file `server/Data.py`. In this file one also finds the implementation of the reading process from files into the memory data structures. This process will not be explained here, as it uses a very generic tool from the `fmTools` library, which will be documented elsewhere.

All data structures regarding exercises and sheets are collected in the `Exercises` module and reside therefore in the file `server/Exercises.py`. In this file one also finds the implementation of the reading and parsing process from files into the memory data structures. This process will also not be explained here, as it basically boils down to using the XML parser `pyRXP` to get a memory representation of the XML tree, which is then recursively translated into the in-memory structures described here. Additionally, the file `server/Exercises.py` contains methods for `Exercise` and `Sheet` objects to offer services via the web.

For this chapter here, we will not describe the latter processes but only document the data structures. The main idea of this chapter is to help the user to write the automatic grading functions he has to supply, because they need conventient read access to the internal data structures.

12.2 Data of Participants

The data structures for the personal data of participants are organized as follows: Every participant has an ID, which can be an arbitrary string. There is a dictionary "`people`" in the `Data` module, which stores for any known participant under his ID one object from the class `Person`.

Every `Person` object has an entry `mcresults`, which is a dictionary where under each sheet name (see section 12.4) there can be stored an object from the class `MCResult`. Note that it is not an inconsistency, if there is nothing bound in `mcresults` under a valid sheet name! Therefore one always has to check whether there is some data before accessing it!

Further, every `Person` object has an entry `homework`, which is a dictionary where for each sheet name (see section 12.4) there can be stored an object from the class `Homework`. Note that also here it is not an inconsistency, if there is nothing bound in `homework` for a valid sheet name!

Every `Person` object has an entry `exams`, which is a list. This list has a certain length and at each position (zero based!) there may either be the value `None` or an object from the class `Exam`. Note that lists in Python must not have holes, such that it is necessary to have the possibility of the value `None`. Also the length of the list is variable, therefore one cannot be sure that it always has the same length for all participants, so please check in your code!

Note that the object of type `Exam` is created at the time the participant registers for the first time for the exam in question and is never destroyed afterwards. Even if the participant deregisters, this only means that the `registration` component of the `Exam` object is set to zero. A `totalscore` of -1 indicates, that the participant did not (yet) participate in the exam.

The personal messages of a person are just stored as a list of strings in the `messages` component of the `Person` object. Revoked messages are no longer stored in memory.

12.3 Data of Groups

The identification of groups works via their number. There is a global dictionary `groups` in the `Data` module, where for each group there is stored an object from the class `GroupInfo` under the (string) value of its number. Numbers are non-negative integers.

The group number 0 plays a special rôle, because it is the group where people show up automatically, if the administrator does not distribute them explicitly. This can be handy, if there is only one group for a small course. The component `people` of the `GroupsInfo` object is updated automatically, when a person registers with the system for the first time.

12.4 Data of Exercises and Sheets

All data for exercises and sheets is collected in the `Exercises` module in the OKUSON server and resides therefore in the file `server/Exercises.py`.

There are the classes `Sheet`, `Exercise`, `Question`, and `TeXText`. A sheet consists of exercises, an exercises of questions, and the "innermost" parts of the whole system are simple strings and TEX-texts, which are texts or formulae, which are typed in TEX input and can be rendered into a PDF file or as image in a web page.

There is a global list `AllSheets` where all known sheet objects are stored in some random order. One should not access this list directly, but use instead the function `SheetList` in the same module, which returns a properly sorted list of triples, where each triple corresponds to one sheet and consists of the sheet number, the sheet name, and the sheet object itself.

The basic incredients of a sheet are stored in the `list` component, which is a list containing `TeXTexts` for the texts between exercises and for non-interactive exercises, and `Exercise` objects for the multiple choice exercises.

All exercises (multiple choice and non-interactive ones) are stored under their name in the global dictionary `AllExercises`. Usually one will not have to access this variable, as one usually will access the exercises via the sheets they are contained in.

The basic incredients of an interactive exercise are objects of type `Question`, which in turn contain probably several variants with possibly different correct solutions.

The fundamental method for the personalization of sheets is the method `ChooserFunction` of the class `Sheet`. It uses a numerical seed as seed for a pseudo random process and returns the choice of actual variants the corresponding participant gets. Note that the submissions and marks stored in the `Data` module only make sense together with this information and the information about the available sheets.

Chapter 13

Using OKUSON sheets with Moodle

There is a script `scripts/ExportToMoodle.py` which can be used to export variants of an OKUSON sheet to a collection of Moodle (`https://moodle.org/`) exercises (of *cloze* question type).

Here is a short description of the workflow.

- install OKUSON as usual, but there is no need to adjust the web pages
- generate exercises and sheets as usual (test with `testexercise` and `testsheet`)
- now we produce a number of variants of a sheet and translate each of them to a Moodle question (so for OKUSON it is a *sheet*, for Moodle it is just one *question*. Say, we want 200 variants of sheet number 1 in a file `moodlesheet1.xml`, then use:

  ```
  scripts/ExportToMoodle.py catsh1 1 100000 50 moodlesh1.xml
  ```

  (calling the script without arguments shows the usage.)
- in your Moodle course import the questions from `moodlesh1.xml` which is in Moodle XML format; these questions are in the question category `catsh1`
- to use these questions in a Moodle *test*, configure a test with only one random question from the category `catsh1` and adjust the maximal number of points accordingly

If you want to provide pdf-versions of the exported variants of a sheet you need some web server for the files (Moodle doesn't provide standard links to uploaded files, e.g. you could use your OKUSON instance). Then call `ExportToMoodle.py` with the base URL for these pdf-files as extra argument:

```
scripts/ExportToMoodle.py catsh1 1 100000 50 moodlesh1.xml \
                                        https://my.server/sheet1
```

Then copy the files inside the created subdirectory `pdfs` to the specified web location. Each generated Moodle exercise contains a link to the corresponding pdf-file.

Appendix A

Customization Examples

A.1 Using IDs of Different Type

In Aachen all students have an associated unique 6-digit number (the *Matrikelnummer*) which is used all over the place. Therefore, we use it for registrations and as ID's within an OKUSON server.

If you want to use other types of IDs, do the following:

- Adjust the two entries `<IdCheckRegExp>` and `<GuestIdRegExp>` in your configuration file `Config.xml`.
- Adjust all places where the word *Matrikelnummer* appears in templates. There is one such line in the `<PDFTemplate>` entry of `Config.xml`. And there are quite a few files in `html/*.tpl`. Here, change the word *Matrikelnummer* to something more appropriate and maybe increase the values of the `size` and `maxlength` attributes of corresponding `input` elements.

That's it.

A.2 A Course Without Non-Interactive Homework Exercises

If you are not going to give non-interactive exercises which are not graded automatically by OKUSON, do the following:

- Just ignore all remarks about *homework points* in this manual and in input forms.
- Adjust the `<Results>` tag in the template file `html/results.tpl` to exclude the homework points.

A.3 A Course Without Interactive Exercises

We also use OKUSON for courses without interactive exercises, just to take advantage of the other utilities: registration, tutoring- and exam- management, HTML versions of exercise sheets, etc.

This can be achieved by setting the option `<IndividualSheets>` of the configuration file `Config.xml` to `0`.

By this switch certain things vanish for the participants, for example it is no longer necessary to type ID and password to get the exercise sheets, on the results page the interactive exercises are not mentioned, and the PDF-sheets don't use a table for the exercises.

The two cases are handled by the same template files, via the tags `<IfIndividualSheets>` and `<IfNoIndividualSheets>`.

The different templates for the TEX-files of the PDF-sheets are marked by `<PDFTemplate>` and `<PDFTemplateNoTable>` in the `Config.xml` file.

A.4 Managing Additional Personal Registration Data

In this section we give a few examples of using the special tags explained in section 6.4.4. The first shows the use of a free-form string, the next three are about asking for additional information concerning tutoring groups using radio buttons, check boxes or select options, respectively.

Example: Birthplace

There are universities which use the place of birth as part of the identification of a person in course certificates. To get and store that information you can use the `persondata` field. In the template `html/registration.tpl` you could add a block:

```
<tr>
  <td>Place of Birth:</td>
  <td> <input size="30" maxlength="30"
              name="persondata.birthplace" value="" /> </td>
</tr>
```

And in the template `html/regchange2.tpl` you could add the following block, in which the value from the registration (or last change) is preset:

```
<tr>
  <td>Place of Birth</td>
  <td> <PersonDataField key="birthplace" /> </td>
</tr>
```

Example: Group Choice with Radio Buttons

Say, you have three different tutoring groups, on Monday, Tuesday and Wednesday. Here is a way to let the participant choose exactly one of these.

In `html/registration.tpl`:

```
<tr>
  <td>Preferred Tutoring Group:</td>
  <td>
    <input type="radio" name="persondata.prefgroup"
           value="Mon" checked="checked" />Monday,
    <input type="radio" name="persondata.prefgroup"
           value="Tue" />Tuesday,
    <input type="radio" name="persondata.prefgroup"
           value="Wed" />Wednesday
  </td>
</tr>
```

To get an appropriate block in the `html/regchange2.tpl` page with the stored choice pre-selected use:

```
<tr>
  <td>Preferred Tutoring Group:</td>
  <td>
     <PersonDataRadioButton name="persondata.prefgroup"
                            value="Mon" />Monday,
     <PersonDataRadioButton name="persondata.prefgroup"
                            value="Tue" />Tuesday,
     <PersonDataRadioButton name="persondata.prefgroup"
                            value="Wed" />Wednesday
  </td>
</tr>
```

Example: Group Choice with Select Options

An alternative for the last example is to use the `select` element, in particular if there are much more than three choices.

You could use the following in `html/registration.tpl`:

```
<tr>
  <td>Preferred Tutoring Group:</td>
  <td> <select name="persondata.prefgroup">
```

```
    <option value="Mon" selected="selected">Monday</option>
    <option value="Tue">Tuesday</option>
    <option value="Wed">Wednesday</option>
       </select>
  </td>
</tr>
```

Getting this with the stored value pre-selected in `html/regchange2.tpl` you could use:

```
<tr>
  <td>Preferred Tutoring Group:</td>
  <td> <select name="persondata.prefgroup">
   <PersonDataSelectOption name="persondata.prefgroup"
                    value="Mon"  content="Monday" />
   <PersonDataSelectOption name="persondata.prefgroup"
                    value="Tue"  content="Tuesday" />
   <PersonDataSelectOption name="persondata.prefgroup"
                    value="Wed"  content="Wednesday" />
   </select>
  </td>
</tr>
```

Example: Group Choice with Check Boxes

As a third variant of the previous two examples, assume you want to allow several choices by each participant. This could be done with check box input elements. You can use the following code snippet in both files, `html/registration.tpl` and `html/regchange2.tpl`:

```
<tr>
  <td>Possible Tutoring Groups:</td>
  <td>
    <input type="hidden" value="None"
           name="persondata.prefgroups" />
    <label><PersonDataCheckBox value="Mo"
           name="persondata.prefgroups" /> Monday</label>
    <label><PersonDataCheckBox value="Tu"
           name="persondata.prefgroups" /> Tuesday</label>
    <label><PersonDataCheckBox value="We"
           name="persondata.prefgroups" /> Wednesday</label>
  </td>
</tr>
```

Then OKUSON stores under the key `persondata.prefgroups` the value `None` and all values of checked variants. separated by semicolons. The `hidden` element is needed such that every request contains at least one value for this key (otherwise one could not deselect all choices from a previous submission).

A.5 Customizing the Look and Feel of the Web Pages

The most obvious thing to play around with may be the setting of colors or a background image. You may try the effect of changing the following line in `html/OKUSON.css`:

```
background-color:#B0E0E6;
```

to

```
background-color:#FF0000;
color:#00FF00;
```

You can be sure that some people cannot read anything on pages with this setting, so this demonstrates that you should be quite careful with choosing sensible colors.

Or you can add a background image with a setting like:

```
background-color:#CCCCCC;
background-image:
  url(http://www.math.rwth-aachen.de/~OKUSON/badback.gif);
```

Of course, as you can see here: Only use background images which don't disturb the legibility of the pages!

More subtle changes would affect table layout, fonts, relative font sizes and many more details. See the http://www.w3c.org pages or some book on CSS if you do not know how to do this. The W3C pages also have an online validation service for CSS style sheets.

As a general rule: Use only basic CSS features which are reasonably well implemented in essentially all current browsers (some CSS settings may shorten the list of successful browser tests, as given in `html/techinfo.tpl`).

There are two CSS style sheet files in the OKUSON sample pages, `OKUSON.css` and `OKUSONSheet.css`. You must be more careful with changes to the latter one. In particular, the background of the table with the exercise texts as images should be white or very bright for good legibility.

If you want to add some icon or some standard information on all course web pages, you must edit all template files accordingly. You can use the customization entries `<ConfigData key="...">` in the `Config.xml` file together with the tags `<ConfigData key="..."`

/> in the template .tpl files. You can include HTML markup in the content of these elements, but note that you must use a CDATA environment or escape the markup with entities in your Config.xml (see 4).

A.6 Using OKUSON with Another Language

As mentioned in the web page chapter 6 the OKUSON system can be adjusted to another language by translating the visible texts in all sample web pages (the html.sample/...tpl files).

If you intend to do this, please contact us. Of course, we would be happy to distribute OKUSON with sets of sample pages in several languages.

Appendix B

Differences Between XHTML and Other Variants of HTML

If you know some HTML and compare it with the basics on XML as explained in 4 you see that the syntax rules for XHTML must be more restrictive than for other (non-XML) variants of HTML. In particular note the following details of XHTML:

- All tags must be written with lower case letters in the element names.
- All non-empty elements must have a start- and end-tag, in particular enclose paragraphs in `<p>` and `</p>` or list entries in `<li>` and `</li>`.
- Attributes always must have an assigned value and the value must be enclosed by either double or single quotes.
- Write empty elements like `<br />`, the space before the `/>` is not necessary according to the specification but it helps some old browsers to interpret it correctly.
- Do not put information on colors or fonts in the XHTML file. Instead use the `.css` style sheet file. (For complicated cases use the `class` attribute to mark elements for which you want to give special formating rules in the style sheet.

Using the W3C HTML 4.01 specification (`http://www.w3.org/TR/html401/`) – this includes a nice *elements* overview – together with the above rules and the general rule to avoid complicated looking constructs when possible, we found it not too difficult to produce sets of valid web pages.

In the OKUSON home directory you find the tool `xmlvalidate`, which can be used for validating (or finding problems with) XHTML documents. But actually, all of our OKUSON web pages are generated dynamically from `.tpl` template files, which cannot be validated directly. Therefore, we have included an on-the-fly XHTML-validation of HTML pages in the OKUSON built-in web server. It does not refuse the delivery of non-valid pages, but removes in that case the validator icon which may have been put in the document via the `<ValidatorIcon/>` tag.

Appendix C

GPL

Here is the text of the license under which the OKUSON package is distributed. Instead of this you may also take any newer version as published in
http://www.gnu.org/licenses/gpl.html.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.