PUPIL

System Documentation

Contents

1. Overview....4
   1.1. The system components....4
2. Backend....5
   2.1. Overview....5
   2.2. File hierarchy....6
   2.3. Database....6
      2.3.1. Teacher....6
      2.3.2. Course....7
      2.3.3. Project....7
      2.3.4. Pattern....9
      2.3.5. Scenetype....10
      2.3.6. Scene....10
      2.3.7. Category....11
      2.3.8. Image....11
      2.3.9. Student....12
      2.3.10. Belonging....12
      2.3.11. Permission....12
      2.3.12. Testcase....13
      2.3.13. Input....13
      2.3.14. SSI....14
      2.3.15. SRI....14
      2.3.16. SCRI....15
      2.3.17. SCRIScene....15
      2.3.18. SOSIImg....16
      2.3.19. SOSIOpt....16
      2.3.20. SORICat....16
      2.3.21. SORIOpt....17
      2.3.22. Block....17
      2.3.23. Sceneblock....18
   2.4. XML RPC....19
      2.4.1. addblock....19
      2.4.2. addcategory....19
      2.4.3. addcodepermission....20
      2.4.4. addpermission....20
      2.4.5. addscenetoblock....20
      2.4.6. addstudent....20
      2.4.7. blocksettings....21
      2.4.8. cloneproject....21
      2.4.9. clonescene....21
      2.4.10. createproject....21
      2.4.11. createscri....22
      2.4.12. createsori....22
      2.4.13. createsosi....23
      2.4.14. createssi....24
      2.4.15. createtext....24
      2.4.16. deleteblock....24
      2.4.17. deletecategory....25
      2.4.18. deleteimage....25
      2.4.19. deleteproject....25

2.4.20. deletescene 25
2.4.21. deletesql 25
2.4.22. deletestudent 26
2.4.23. getallimages 26
2.4.24. getblocklist 26
2.4.25. getblockscenelist 26
2.4.26. getblocksettings 27
2.4.27. getcategorylist 27
2.4.28. getimagesforcategory 27
2.4.29. getprojectdetails 28
2.4.30. getsceneinfo 29
2.4.31. getscenelist 29
2.4.32. getsqlfile 29
2.4.33. getwithoutpermission 29
2.4.34. getwithpermission 30
2.4.35. importlist 30
2.4.36. importstudents 30
2.4.37. listuniqueimages 30
2.4.38. registerinput 31
2.4.39. remcodepermission 31
2.4.40. removescenefromblock 31
2.4.41. rempermission 32
2.4.42. replacecategory 32
2.4.43. replaceimage 32
2.4.44. resetdata 32
2.4.45. savesql 33
2.4.46. scenesinblock 33
2.4.47. scenesnotinblock 33
2.4.48. updateproject 34
2.4.49. updatescri 36
2.4.50. updatesori 37
2.4.51. updatesosi 38
2.4.52. updatessi 39
2.4.53. updatestudent 39
2.4.54. updatetext 39
3. Frontend....40
3.1. File hierarchy....40
4. Installation....41
4.1. Dependencies....41
4.2. Binary installation....41
4.2.1. Setting up database....42
4.2.2. Setting up apache....42
4.3. Building from source....43
4.3.1. Build dependencies....43
4.3.2. Compiling backend....43
4.3.3. Compiling frontend....44
5. Maintenance....44
5.1. Backups....44
5.2. Adding or removing teachers....45

1. Overview

The PUPIL (“Pedagogiskt UtvecklingsProjekt I Labbet”... approximately “Pedagogical developmental project in the lab”) project implements a web based system for completely web based visual experiments.

The documentation for the project consists of three parts:

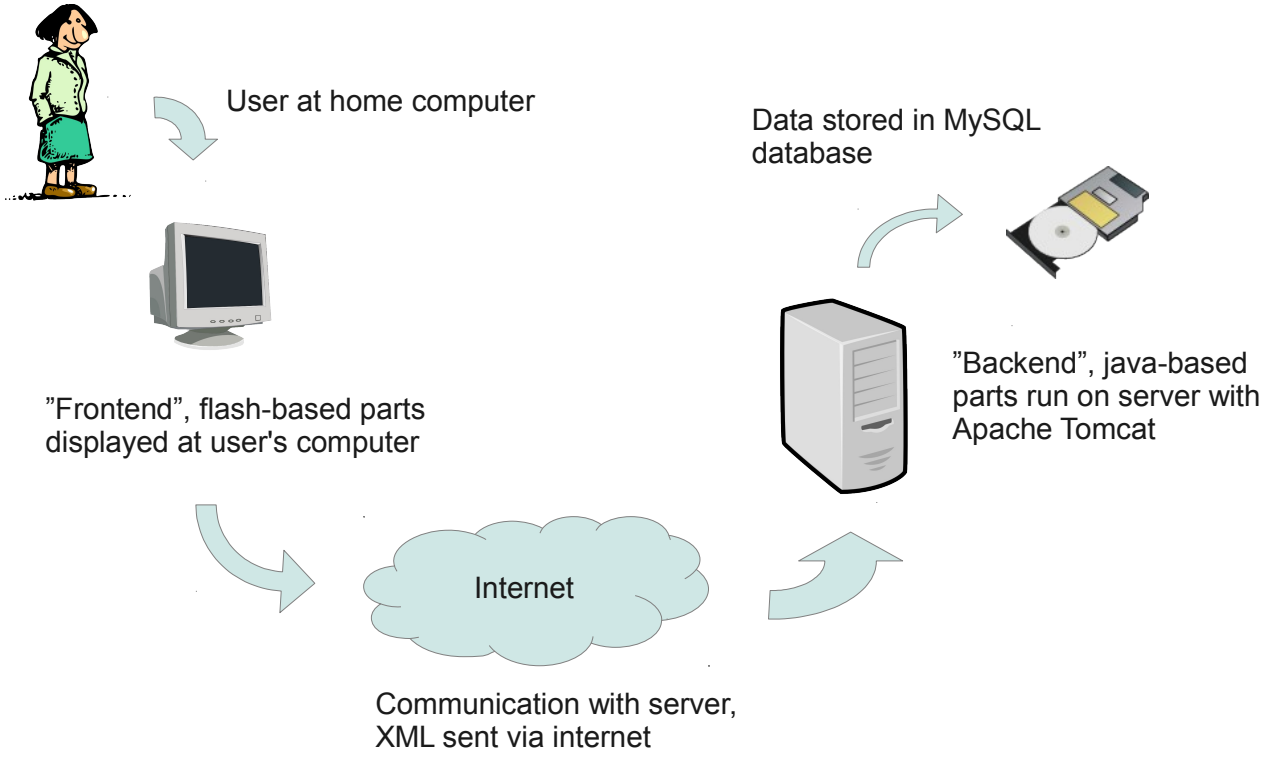
- The user documentation
- The code documentation
- The system documentation (this document)

For basic instructions on how to use the system once it is installed, please refer to the user documentation. For more in-depth information about the code, please refer to the code documentation.

This document provides a systems overview of the system: How it is structured, how the different parts relate to each other and how to install/manage the system.

The document presupposes basic knowledge of Linux systems. Things such as web server, servlet engine and databases will not be explained. It is assumed the reader already knows how to install and manage such things.

1.1. The system components



The system conceptually consists of three parts, the “Frontend” which is what users see, the “Backend” which handles the logic, and the database which stores the data.

The user uses a web browser to access the server. The browser starts up a flash application (“Front end”) once the server has determined that the user is authorized to use the service.

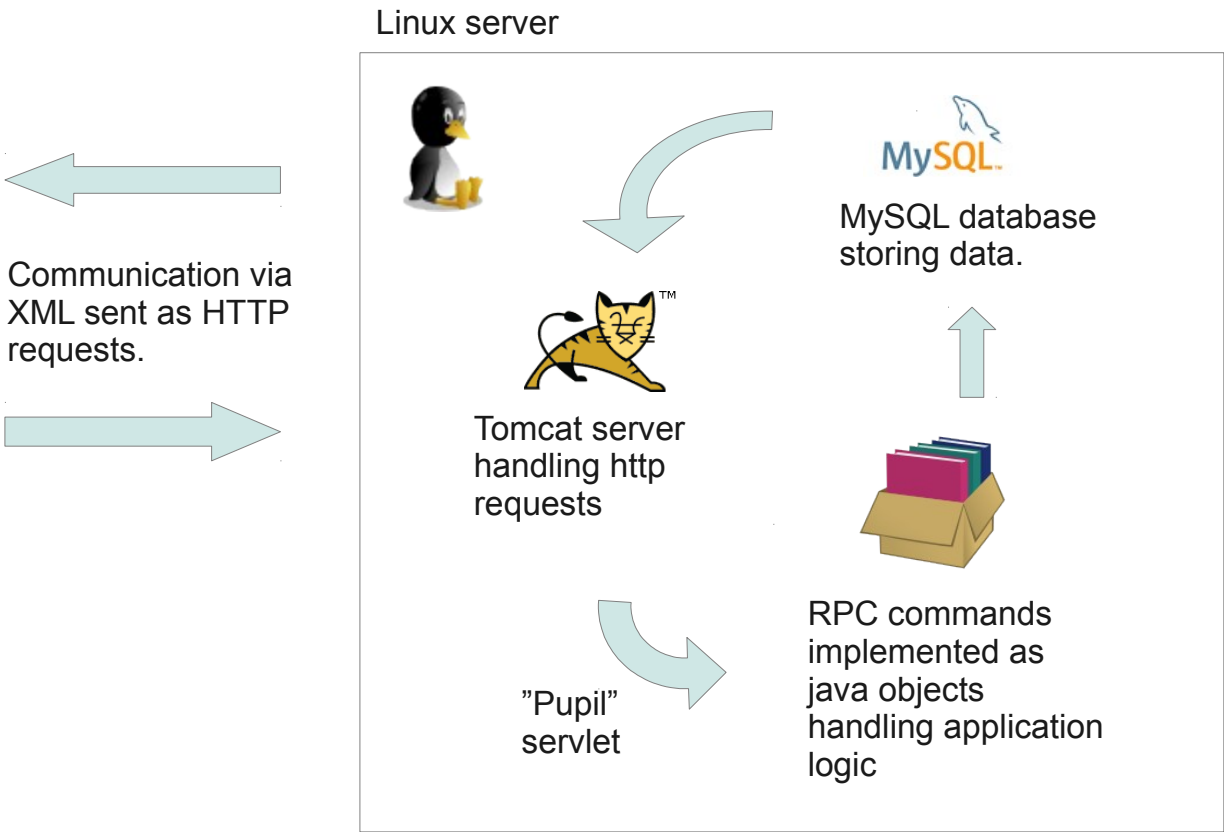
The flash application requests information and sends data to the server using XML RPC. In practice, these are text documents sent by form submits via normal HTTP.

The server (“Back end”) receives the XML requests, parses them and acts upon them. If data is requested, an XML document is sent back containing the data. If data was sent from the frontend, the data is stored in the database.

2. Backend

The backend is the server parts of the system. These are usually run on a Linux server.

2.1. Overview



The linux server runs Tomcat as servlet engine. Tomcat calls the “Pupil” servlet, which in essence consists of a set of RPC commands, each implemented as a java object. The objects react to the incoming communication (always an XML request) passed on by Tomcat, and performs an operation in the database, storing and/or retrieving data. The end result is passed back to tomcat as an XML text fragment which is sent back as answer to the initial request.

2.2. File hierarchy

On the server, the files are stored in this hierarchy (inside the "pupil" directory). Directories irrelevant for the discussion has been excluded:

Path	Contents
survey	Mod_survey files related to running projects
WEB-INF	The root of the logic for the pupil servlet
WEB-INF/src	The source code for the pupil servlet
WEB-INF/src/pupil/command	The XML RPC commands making up most of the logic
WEB-INF/src/sql	The SQL commands used to construct the database
WEB-INF/lib	The dependencies for the pupil servlet
WEB-INF/classes	The compiled pupil servlet
flash	The root of the "frontend" parts
flash/pupil	The source code of the "frontend" parts
flash/images	Images used in projects
sql	Data export statements for projects

Below, the database and the XML RPC will be discussed. Dependencies and the frontend parts are discussed in a later section.

2.3. Database

The database presupposes MySQL. It is defined through a set of SQL-script files (one per table) available in WEB-INF/src/sql. A script "setupdb.pl" is used to run the SQL script files to construct a new empty database. This is discussed more in detail in the sections about installation later in the document.

The following sub-sections contains a detailed specification of the tables used in the pupil database. The specification is ordered along their interdependencies (ie, a table that other tables depend on must be created before those other tables can be created). For a developer studying this reference, it is suggested that "Project" (2.3.3) is the table to start reading at.

2.3.1. Teacher

```
CREATE TABLE teacher (
  teacher_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  login VARCHAR(80) NOT NULL,
  pass VARCHAR(80) NOT NULL
) ENGINE=INNODB;
```

The "teacher" table consists information about users allowed to create and manage experiments. Each teacher has a teacher_id, which is a numerical reference never shown to the user, a "login" which is a username and a "pass" which is a plain text password.

2.3.2. Course

```
CREATE TABLE course (
  course_id INTEGER PRIMARY KEY AUTO_INCREMENT,
  teacher_id INTEGER NOT NULL,
  code VARCHAR(80),
  name VARCHAR(80),
  description VARCHAR(80),
  FOREIGN KEY (teacher_id) REFERENCES teacher(teacher_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The "course" table used to be around to manage the connection between students and experiments. It has since become deprecated, as students are instead directly tied to each experiment. The table remains to maintain database consistency, but can otherwise be ignored.

2.3.3. Project

```
CREATE TABLE project (
  project_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  teacher_id INTEGER NOT NULL,
  name VARCHAR(80) UNIQUE,
  description VARCHAR(80),
  displaywelcome BOOLEAN DEFAULT 0,
  displaythanks BOOLEAN DEFAULT 0,
  welcometop VARCHAR(200) DEFAULT "",
  welcomemid VARCHAR(200) DEFAULT "",
  welcomebottom VARCHAR(200) DEFAULT "",
  thankstop VARCHAR(200) DEFAULT "",
  thanksmid VARCHAR(200) DEFAULT "",
  thanksbottom VARCHAR(200) DEFAULT "",
  urlredirect VARCHAR(500) DEFAULT "",
  maxwidth INTEGER DEFAULT 0,
  maxheight INTEGER DEFAULT 0,
  flashright BOOLEAN DEFAULT 0,
  flashwrong BOOLEAN DEFAULT 0,
  flashwhite BOOLEAN DEFAULT 0,
  whitemin INTEGER DEFAULT 800,
  whitemax INTEGER DEFAULT 1600,
  hideopts BOOLEAN DEFAULT 0,
  displaypolicy INTEGER DEFAULT 1,
  splicearray BOOLEAN DEFAULT 0,
```

```
 subsetsize INTEGER DEFAULT 0,
 pauseimage_id INTEGER DEFAULT NULL,
 wrongimage_id INTEGER DEFAULT NULL,
 rightimage_id INTEGER DEFAULT NULL,
 blockrandom BOOLEAN DEFAULT 0,
 FOREIGN KEY (teacher_id) REFERENCES teacher(teacher_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The “project” table is in some sense the centre of the database. Each line in it describes the basic information for an experiment. The columns describe general settings for the project. Data, students and scenes refer to this table. In the following “experiment” and “project” are used as synonyms.

Each project has a “project_id” which is used in other tables to refer to the project, but which is never shown to the user. The project has a reference to a teacher (2.3.1), which is the person who created the project.

The project has a “name” which is what users use when they refer to the experiment, and which is also what is used by all XML RPC commands that need to refer to a project. This name must be unique.

The project has a “description” which is an arbitrary string shown to the user in the list of projects.

The two booleans “displaythanks” and displaywelcome” determine whether text screens should be shown to the test person before and/or after an experiment is run. If the booleans are set, the “welcometop”, “welcomemid”, “welcomebottom”, “thankstop”, “thanksmid”, and “thanksbottom” columns are text strings put on the respective text screens.

If “urlredirect” is set, it is an URL that the web browser should be redirected to once the experiment is finished. If not set, a messagebox notifying the user that the experiment is done is instead shown.

The columns “maxwidth” and “maxheight” are measurements in pixels determining if the project should be limited in screen space rather than expanding over the available screen. If set to 0 these settings are ignored.

“flashright”, “flashwrong” and “flashwhite” are booleans determining if extra pauses should be made between showing scenes in the project. White is here a pause with no information on the screen, while right and wrong are short pulses shown depending on the previous user input.

If “flashwhite” is set, the “whitemin” and “whitemax” are settings in milliseconds determining the timespan for how long the pause between scenes should be. The end pause is randomized between min and max.

If “hideopts” is set, option texts are always hidden even in scenes that would normally show option texts.

The “splicearray” determines if an image is allowed to show up twice within the same scene. If set to 1, this will be avoided, but may lead to problems if there are too few images in a category.

If “subsetsize” is set to any value above 0, that many scenes will be picked and shown. If set to 0, all scenes will be shown.

Instead of colored full screen flashes, the right/wrong/pause events can contain an image. The “pauseimage_id”, “wrongimage_id” and “rightimage_id” then contains which image to show. These integers are references to the image (2.3.8) table, but since this would cause a circular dependency, the ACID reference integrity setting has been excluded here.

Finally “blockrandom” determines whether blocks should be shown in random order. If set to 0, blocks are shown in alphabetical order.

2.3.4. Pattern

```
CREATE TABLE pattern (
  pattern_id INTEGER PRIMARY KEY NOT NULL,
  images INTEGER NOT NULL,
  mnemonic VARCHAR(30) NOT NULL,
  description VARCHAR(120) NOT NULL
) ENGINE=INNODB;
```

The “pattern” table contains pattern definitions. A pattern is a layout of images on the screen in a scene, for example three by two images shown. The list of available patterns is static and never changes.

The “pattern_id” is used for scenes to refer to the pattern, it is never show to the user. “images” determines how many images the pattern contains (for example “3x2” contains 6 images). “mnemonic” is the name of the patter, as seen by the user, for example “3x2”. “description” is not currently used.

For a list of the contents of this table, see the user documentation.

2.3.5. Scenetype

```
CREATE TABLE scenetype (
  scenetype_id INTEGER NOT NULL PRIMARY KEY,
  mnemonic VARCHAR(30) NOT NULL,
  description VARCHAR(250) NOT NULL
) ENGINE=INNODB;
```

The scene type is the basic type that each scene belong to. This table contains static information and is never changed. The “scenetype_id” is used by the scene table to refer to its type. The “mnemonic” is the name of the type as seen by the user. The “description” is currently not used.

The scene types is used to determine where to find extra information. Thus the scene's type leads to other tables being involved. The scene types are:

- static_image, reads from the SSI table (2.3.14)
- static_category_random_image, reads from the SCRI table (2.3.16)
- select_option_static_image, reads from SOSIImg (2.3.18) and SOSIOpt (2.3.19)
- select_option_random_image, reads from SORICat (2.3.20) and SORIOpt (2.3.21)
- text, does not contain extra information (this is just a text information scene)

2.3.6. Scene

```
CREATE TABLE scene (
  scene_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  scenetype_id INTEGER NOT NULL,
  pattern_id INTEGER NOT NULL,
  project_id INTEGER NOT NULL,
  timeout INTEGER DEFAULT 0 NOT NULL,
  description VARCHAR(80),
  lead VARCHAR(250) DEFAULT '',
  correctkey CHAR DEFAULT '~',
  UNIQUE(project_id,description),
  FOREIGN KEY (scenetype_id) REFERENCES scenetype(scenetype_id) ON DELETE CASCADE,
  FOREIGN KEY (pattern_id) REFERENCES pattern(pattern_id) ON DELETE CASCADE,
  FOREIGN KEY (project_id) REFERENCES project(project_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

Next to the project table, the scene table is the most important in the database. Each line in the scene table describe a view shown to a test person in an experiment. Depending on its type (see the Scenetype table, 2.3.5), multiple other tables may be read to compile the information needed for the scene.

The scene has a “scene_id” which is used to refer to the scene otherwise in the database. This is never shown to the user. The “scenetype_id” determines the type of the scene. The “pattern_id” determines the layout of the graphics on the scene. The “project_id” ties the scene together with the correct project.

If “timeout” is set, the next scene will be shown after that many milliseconds even if the user has not given an answer (if an answer is given, that will cause a scene switch before the timeout). “description” is the name of the scene as shown to the user designing the project.

“lead” is the text information shown on the scene. Depending on scene type this will have different function. On SSI and SCRI scenes it is ignored. On SORI and SOSI scenes it will be shown as a caption above the available options. On Text scenes it will be the only information shown at all.

“correctkey” is what key should be called “correct” when storing information about the user input. If set to “~”, any key will be considered correct.

2.3.7. Category

```
CREATE TABLE category (
  category_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  project_id INTEGER NOT NULL,
  name VARCHAR(80) UNIQUE,
  FOREIGN KEY (project_id) REFERENCES project(project_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The “category” table groups images within a project. For example, a category could be “spiders” and then connect images of individual spiders. Categories are always specific to a project. There are no cross-project categories. The “name” is the name of the category as shown to the user. The “project_id” is a reference to which project the category belongs to.

2.3.8. Image

```
CREATE TABLE image (
  image_id INTEGER PRIMARY KEY NOT NULL AUTO_INCREMENT,
  category_id INTEGER NOT NULL,
  file_name VARCHAR(80),
  FOREIGN KEY (category_id) REFERENCES category(category_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

Each line in the “image” table represents an image to be shown as part of a scene. Each image belongs to a category. “file_name” is the search path to the image, without path name. The path is deduced as “flash/images/[category.name]/”.

2.3.9. Student

```
CREATE TABLE student (
  student_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  login VARCHAR(80) NOT NULL,
  pass VARCHAR(80) NOT NULL
) ENGINE=INNODB;
```

Test persons are in the language of pupil called “student” (since pupil began as an online teaching tool). A student has a “login” which is the user name and a “pass” which is a plaintext password. The “student_id” is a numerical reference used to refer to the student and never shown to the user.

2.3.10. Belonging

```
CREATE TABLE belonging (
  course_id INTEGER NOT NULL,
  student_id INTEGER NOT NULL,
  FOREIGN KEY (course_id) REFERENCES course(course_id) ON DELETE CASCADE,
  FOREIGN KEY (student_id) REFERENCES student(student_id) ON DELETE CASCADE,
  PRIMARY KEY (course_id,student_id)
) ENGINE=INNODB;
```

“Belonging” used to tie students to courses. This table is no longer used, but is kept to keep database consistency.

2.3.11. Permission

```
CREATE TABLE permission (
  permission_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  student_id INTEGER NOT NULL,
  project_id INTEGER NOT NULL,
  FOREIGN KEY (student_id) REFERENCES student(student_id) ON DELETE CASCADE,
  FOREIGN KEY (project_id) REFERENCES project(project_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The “permission” table determines which students are allowed to run which projects. It is simply a connector table between project and student where “student_id” is the student reference and “project_id” is the project reference.

2.3.12. Testcase

```
CREATE TABLE testcase (
  testcase_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  project_id INTEGER NOT NULL,
  student_id INTEGER NOT NULL,
  FOREIGN KEY (project_id) REFERENCES project(project_id) ON DELETE CASCADE,
  FOREIGN KEY (student_id) REFERENCES student(student_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The “testcase” is a case, ie one experiment run by one test person. Each test person has one testcase per trial and per project. In most cases there is only one trial, so think of testcase as one student's collected test data within one project. The testcase has no information of its own but is simply used to tie “input” to a “student” within a “project”.

2.3.13. Input

```
CREATE TABLE input (
  input_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  testcase_id INTEGER NOT NULL,
  scene_id INTEGER NOT NULL,
  time_start BIGINT DEFAULT 0,
  time_end BIGINT DEFAULT 0,
  time_delta INTEGER,
  actual_input VARCHAR(5),
  correct_input BOOLEAN,
  FOREIGN KEY (testcase_id) REFERENCES testcase(testcase_id) ON DELETE CASCADE,
  FOREIGN KEY (scene_id) REFERENCES scene(scene_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

“Input” is an single time measurement within one test case, ie data about how a student reacted to one particular scene. “Input” lines are grouped together in a “testcase”.

The “testcase_id” identifies which test case the input belongs to. The “scene_id” identifies which scene was shown to collect the data.

“time_start” and “time_end” is milliseconds since epoch for when the input happened. “time_delta” is the calculated difference between “time_start” and “time_end”, or in other words the time between the scene was shown and the student hit a key.

“actual_input” is what key the student hit and “correct_input” is a boolean showing if this was the deemed correct input.

2.3.14. SSI

```
CREATE TABLE ssi (
  ssi_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  image_id INTEGER NOT NULL,
  scene_id INTEGER NOT NULL,
  position INTEGER NOT NULL,
  keychar CHAR NOT NULL,
  FOREIGN KEY (scene_id) REFERENCES scene(scene_id) ON DELETE CASCADE,
  FOREIGN KEY (image_id) REFERENCES image(image_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The SSI (static image, there's a superfluous “s” in the table name) ties a particular image together with a particular scene and determines on which position on the screen the image should be displayed.

“position” is value from 1 to 9 determining position within the scene's pattern. It is calculated from left to right and from top down. For example, in the “3x2” pattern, “1” would be top left and “5” would be middle bottom.

The “keychar” is which key should be pressed to choose the particular image in the scene.

2.3.15. SRI

```
CREATE TABLE sri (
  sri_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  scene_id INTEGER NOT NULL,
  category1 INTEGER NOT NULL,
  category2 INTEGER,
  FOREIGN KEY (category1) REFERENCES category(category_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The SRI table used to list images in a scene type which is no longer used. This table is deprecated and can be ignored, but is kept for consistency.

2.3.16. SCRI

```
CREATE TABLE scri (
  scri_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  scene_id INTEGER NOT NULL,
  category_id INTEGER NOT NULL,
  position INTEGER NOT NULL,
  keychar CHAR NOT NULL,
  FOREIGN KEY (scene_id) REFERENCES scene(scene_id) ON DELETE CASCADE,
  FOREIGN KEY (category_id) REFERENCES category(category_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The “SCRI” (static category, random image) list categories along with which position to display a random image from the category within a scene.

In the table the “scene_id” is the relevant scene and “category_id” is the category to pick an image from. For “position” and “keychar”, see description in “SSI” (2.3.15).

2.3.17. SCRIScene

```
CREATE TABLE scriscene (
  scriscene_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  scene_id INTEGER NOT NULL,
  noduplicates BOOLEAN,
  FOREIGN KEY (scene_id) REFERENCES scene(scene_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The “SCRIScene” contains extra scene information for scenes of the SCRI type. The only value here is “noduplicates” which determines if the same image should be allowed to be displayed twice within the same scene

2.3.18. SOSIImg

```
CREATE TABLE sosiimg (
  sosiimg_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  image_id INTEGER NOT NULL,
  scene_id INTEGER NOT NULL,
  position INTEGER NOT NULL,
  FOREIGN KEY (scene_id) REFERENCES scene(scene_id) ON DELETE CASCADE,
  FOREIGN KEY (image_id) REFERENCES image(image_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The “SOSIImg” (SOSI is Static Option, Static Image) table ties an image together with a position on a scene of the SOSI type. For the “position” column, see description in the SSI table.

2.3.19. SOSIOpt

```
CREATE TABLE sosiopt (
  sosiopt_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  optiontxt VARCHAR(250) NOT NULL,
  keychar VARCHAR(5) NOT NULL,
  scene_id INTEGER NOT NULL,
  FOREIGN KEY (scene_id) REFERENCES scene(scene_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The “SOSIOpt” describes available text options within a SOSI scene. Each option is a line of text together with a key. These are displayed under the images of the scene.

“optiontxt” is the text belonging to the option. The “keychar” is the keypress associated with the option.

2.3.20. SORICat

```
CREATE TABLE soricat (
  soricat_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  category_id INTEGER NOT NULL,
  scene_id INTEGER NOT NULL,
  position INTEGER NOT NULL,
  FOREIGN KEY (scene_id) REFERENCES scene(scene_id) ON DELETE CASCADE,
  FOREIGN KEY (category_id) REFERENCES category(category_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The “SORICat” (SORI is Static Option, Random Image) table ties categories of images together with position on a scene of the SORI type. For “position” see description in the SSI table.

2.3.21. SORIOpt

```
CREATE TABLE soriopt (
  soriopt_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  optiontxt VARCHAR(250) NOT NULL,
  keychar VARCHAR(5) NOT NULL,
  scene_id INTEGER NOT NULL,
  FOREIGN KEY (scene_id) REFERENCES scene(scene_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The “SORIOpt” describes available text options within a SORI scene. Each option is a line of text together with a key. These are displayed under the images of the scene.

“optiontxt” is the text belonging to the option. The “keychar” is the keypress associated with the option.

2.3.22. Block

```
CREATE TABLE block (
  block_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  project_id INTEGER NOT NULL,
  name VARCHAR(80) NOT NULL,
  random BOOLEAN DEFAULT 0,
  textfirst BOOLEAN DEFAULT 0,
  UNIQUE(project_id,name),
  FOREIGN KEY (project_id) REFERENCES project(project_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

A “block” groups scenes together and is used to determine the order in which scenes are shown. Only scenes belonging to a block will be shown.

The “name” of the block is what is shown to users. In the normal case blocks are shown in alphabetical order, but this can be overridden in the “project” table.

The “random” column determines if scenes should be randomized inside the block (note that this is different than whether the *blocks* should be displayed in random order). “textfirst” determines if text scenes always should be shown first even when the scenes are otherwise shown in random order.

2.3.23. Sceneblock

```
CREATE TABLE sceneblock (
  block_id INTEGER NOT NULL,
  scene_id INTEGER NOT NULL,
  FOREIGN KEY (block_id) REFERENCES block(block_id) ON DELETE CASCADE,
  FOREIGN KEY (scene_id) REFERENCES scene(scene_id) ON DELETE CASCADE
) ENGINE=INNODB;
```

The “sceneblock” table simply ties scenes together with blocks. Note that a scene can belong to multiple blocks.

2.4. XML RPC

The core setup of pupil consists of XML RPC (Remote Procedure Call). In practice this means fragments of XML are sent to the server as an instruction to do something, usually either store or retrieve data. An example of the “cloneproject” RPC could look like this:

```
<command>
    <function>cloneproject</function>
    <parameter name="name" value="mynewproject" />
    <parameter name="description" value="A copy of my old project" />
    <parameter name="oldname" value="myoldproject" />
</command>
```

The root of the XML is “<command>” and it contains “<function>” which is the name of the call and “<parameter>” which modifies the call.

All the RPC looks like this within pupil.

In the following reference, the title of the section is what would end up in “<function>” and the blue block is the list of valid parameters (which would show up in “<parameter>”).

The return values are always XML, but may look slightly different dependent on the operation.

All references to “scenes”, “project”, “block” etc use the alphanumeric name of the object. Database keys are never used in the RPC. Boolean values are always in the format “0” for false and “1” for true. Arrays are denoted as “.. [ ]”. For an array “category[ ]” the individual values would be called “category1”, “category2” and so on.

2.4.1. addblock

name
project

Addblock constructs a new (empty) block with the given name for the given project.

RETURNS: “ok” or an error string.

2.4.2. addcategory

name
project

Addcategory constructs a new (empty) category with the given name within the given project.

RETURNS: “ok” or an error string.

2.4.3. addcodepermission

code project

Addcodepermission assigns permission to the given project for all students belonging to the course represented by the “code”. “Code” is the course catalog code, normally using five digits. This function is only valid within MIUN.

RETURNS: “ok” or an error string.

2.4.4. addpermission

login project

Addpermission assigns permission to a given project for a student with the given login

RETURNS: “ok” or an error string.

2.4.5. addscenetoblock

project scene block

Addscenetoblock assigns a given scene to a given block within the given project. Scenes can be assigned to multiple blocks.

RETURNS: “ok” or an error string.

2.4.6. addstudent

login pass

Addstudent creates a new student with the given login and the given password.

RETURNS: “ok” or an error string.

2.4.7. blocksettings

project
block
random
textfirst

Blocksettings saves the two boolean settings “random” (show scenes in random order) and “textfirst” (always show text scenes first in the block disregarding randomization) for the given block within the given project.

RETURNS: “ok” or an error string.

2.4.8. cloneproject

name
description
oldname

Cloneproject creates a full clone of the project given in oldname, and assigns the clone the given name and the given description. The categories in the new project are named with the new name as prefix.

RETURNS: “ok” or an error string.

2.4.9. clonescene

project
original
clone

Clonescene makes a copy of the scene with the name given by original and assigns the new name given by clone, within the given project.

RETURNS: “ok” or an error string.

2.4.10. createproject

name
description

Createproject constructs a new empty project with the given name and description.

RETURNS: “ok” or an error string.

2.4.11. createscri

```
project
pattern
label
correct
timeout
noduplicates
category[ ]
key[ ]
```

Createscri creates a new SCRI-scene. Here “label” is the name of the new scene, “correct” is the keypress that should be considered correct, timeout is an integer representing the milliseconds after which the scene should be marked as failed if no keypress has occurred, “noduplicates” a boolean controlling whether the same image should be allowed to appear twice within the scene.

Category[ ] and key[ ] are the category and keypress representing each image on the scene.

RETURNS: “ok” or an error string.

2.4.12. createsori

```
project
pattern
label
correct
timeout
numopts
lead
category[ ]
option[ ]
key[ ]
```

Createscri creates a new SORI-scene. Here “label” is the name of the new scene, “correct” is the keypress that should be considered correct, timeout is an integer representing the milliseconds after which the scene should be marked as failed if no keypress has occurred, “noduplicates” a boolean controlling whether the same image should be allowed to appear twice within the scene. “Lead” is the text representing the question line. “Numopts” is the number of options (and should correcpond to the length of option[ ] and key[ ]).

Category[ ] are the categories representing the images on the scene. Option[ ] and key[ ] are the options and their corresponding keys.

RETURNS: “ok” or an error string.

2.4.13. createsosi

project
pattern
label
correct
timeout
numopts
lead
option[ ]
key[ ]
category[ ]
file[ ]

Createscri creates a new SOSI-scene. Here “label” is the name of the new scene, “correct” is the keypress that should be considered correct, timeout is an integer representing the milliseconds after which the scene should be marked as failed if no keypress has occurred, “noduplicates” a boolean controlling whether the same image should be allowed to appear twice within the scene. “Lead” is the text representing the question line. “Numopts” is the number of options (and should correcpond to the length of option[ ] and key[ ]).

Category[ ] and file[] are the images on the scene (file is the name of the image within the category). Option[ ] and key[ ] are the options and their corresponding keys.

RETURNS: “ok” or an error string.

2.4.14. createssi

```
project
pattern
label
correct
timeout
category[ ]
file[ ]
key[ ]
```

Createscri creates a new SI-scene. Here “label” is the name of the new scene, “correct” is the keypress that should be considered correct, timeout is an integer representing the milliseconds after which the scene should be marked as failed if no keypress has occurred, “noduplicates” a boolean controlling whether the same image should be allowed to appear twice within the scene.

Category[ ] and file[] are the images on the scene (file is the name of the image within the category). Key[ ] are the corresponding keys.

RETURNS: “ok” or an error string.

2.4.15. createtext

```
project
label
text
```

Createtext creates a text scene. Here “text” is the text of the scene. It can contain row breaks (“\n”) to delimit lines of text.

RETURNS: “ok” or an error string.

2.4.16. deleteblock

```
project
block
```

Deleteblock removes the block with named by “block” within “project”. This only removes the scene assignments. Scenes as such are left untouched.

RETURNS: “ok” or an error string.

2.4.17. deletecategory

name

Delete the category with “name”. WARNING: This cascades so that all scenes and images using the category will also be removed.

RETURNS: “ok” or an error string.

2.4.18. deleteimage

category image

Delete the image “image” within the given category. WARNING: This cascades so that all scenes using the image will also be removed.

RETURNS: “ok” or an error string.

2.4.19. deleteproject

name

Delete the project and all its data.

RETURNS: “ok” or an error string.

2.4.20. deletescene

project scene

Delete the scene within the given project.

RETURNS: “ok” or an error string.

2.4.21. deletesql

name

Delete the SQL file with the given filename.

RETURNS: “ok” or an error string.

2.4.22. deletestudent

login

Removes the student login. WARNING: This also removes all data generated with the student login.

RETURNS: “ok” or an error string.

2.4.23. getallimages

project

Retrieves a list of all images within a project.

RETURNS: An array of “row” tags within the data section of the result XML. Each row contains the attributes “file_name” and “category”.

2.4.24. getblocklist

project

Get a list of blocks declared in the project.

RETURNS: An array of “row” tags within the data section of the result XML. Each row contains the attribute “name”, which is a name of a block.

2.4.25. getblockscenelist

name

Get a list of scenes associated with a block “name”.

RETURNS: An array of “row” tags within the data section of the result XML. Eacg row describes a scene and contains the following attributes:

- block: The name of the block
- scenetype: The type of the scene
- description: The name of the scene
- images: The number of images within the scene
- pattern: The pattern of the scene

2.4.26. getblocksettings

project block

Get the current settings for a a block.

RETURNS: One row tag within the data block. The row has the booleans “random” (if scenes should be randomized within the block) and “textfirst” (if text scenes should be displayed first if randomization is enabled).

2.4.27. getcategorylist

project

Get a list of categories within a project.

RETURNS: An array of “row” tags within the data section of the result XML. Each row contains the attribute “name”, which is a name of a category.

2.4.28. getimagesforcategory

name

Get a list of images within a category.

RETURNS: An array of “row” tags within the data section of the result XML. Each row contains the attribute “file_name”, which is a name of an image within the category.

2.4.29. getprojectdetails

name

Get a list of global settings for a project.

RETURNS: One row tag within the data block. The tag contains the following attributes:

- name: The name of the project
- login: The login that created the project
- displaywelcome: Boolean for whether a text scene should be shown at the start of the project
- displaythanks: Boolean for whether a text scene should be shown at the end of the project
- welcometop: The first line of text of the welcome scene.
- welcomemid: The second line of text of the welcome scene.
- welcomebottom: The third line of text of the welcome scene.
- thankstop: The first line of text of the thanks scene.
- thanksmid: The second line of text of the thanks scene.
- thanksbottom: The third line of text of the thanks scene.
- urlredirect: URL to redirect to when project has finished
- maxwidth: Width in pixels to use for laying out a project
- maxheight: Height in pixels to use for laying out a project
- flashright: Boolean setting for flash a “correct” feedback on correct input
- flashwrong: Boolean setting for flash a “wrong” feedback on wrong input
- flashwhite: Boolean setting for showing a pause between scenes
- whitemin: Minimum time in milliseconds for pause between scenes
- whitemax: Maximum time in milliseconds for pause between scenes
- displaypolicy: Policy for displaying scenes where 1 is “all scenes in order”, 2 is “all scenes randomized and 3 “randomized subset. NOTE: this setting is deprecated and no longer used
- subsetsize: Number of scenes to pick for a subset with displaypolicy = 3. NOTE: this setting is deprecated and no longer used
- hideopts: Boolean for whether to always hide option texts in option scenes
- blockrandom: Boolean for whether blocks should be shown in random order
- rightcategory: Category to use when showing image instead of color when flashing “correct” feedback.
- rightimage: Image to use when showing image instead of color when flashing “correct” feedback.
- wrongcategory: Category to use when showing image instead of color when flashing “wrong” feedback.
- wrongimage: Image to use when showing image instead of color when flashing “wrong” feedback.
- pausecategory: Category to use when showing image instead of color when pausing between scenes.
- pauseimage: Image to use when showing image instead of color when pausing between scenes.
- splicearray: Whether to never show an image again during the rest of the project once it has been shown the first time.

2.4.30. getsceneinfo

scenename projectname

[TO BE WRITTEN]

2.4.31. getscenelist

name

Get a list of all scenes within the project “name”.

RETURNS: An array of “row” tags within the data section of the result XML. Each row describes a scene and contains the following attributes:

- description: The name of the scene
- scenetype: The type of the scene
- pattern: The pattern the scene uses

2.4.32. getsqlfile

name

Get the contents of an SQL file (ie the SQL script as such, not the data it would return.

RETURNS: SQL script in plain text. WARNING: this is not formatted as XML.

2.4.33. getwithoutpermission

project

Get a list of students who does not have access to the given project.

RETURNS: An array of “row” tags within the data section of the result XML. Each row describes a student and contains the attribute “login” which is a student login.

2.4.34. getwithpermission

project

Get a list of students who have access to the given project.

RETURNS: An array of “row” tags within the data section of the result XML. Each row describes a student and contains the attribute “login” which is a student login.

2.4.35. importlist

kod

Fetch a list of students from the central database. The “kod” is the internal MIUN course code the students should match.

RETURNS: An array of “row” tags within the data section of the result XML. Each row describes a student and contains the attribute “login” which is a student login.

2.4.36. importstudents

kod

Import a list of students from the central database and create them in the pupil database. The “kod” is the internal MIUN course code the students should match.

RETURNS: “ok” or an error string.

2.4.37. listuniqueimages

name

Get a list of images within a project. Images are only listed once even if they appear in multiple scenes.

RETURNS: An array of “row” tags within the data section of the result XML. Each row describes an image and contains the attributes “category” and “file”.

2.4.38. registerinput

project

Register the input timings for the scenes displayed during an experiment run.

This command requires a “data” tag with an array of “scene” tags. Each scene should have the following parameters:

- displayno: The sequential number for where the scene was shown
- start: When the scene was displayed as specified by time in milliseconds since epoch.
- end: When the scene was closed as specified by time in milliseconds since epoch.
- delta: Time in milliseconds the scene was visible
- keychar: Which key was pressed
- correct: Whether this was correct or not
- name: The name of the scene.

RETURNS: “ok” or an error string.

2.4.39. remcodepermission

code
project

Remove permissions to a project for all students belonging to a course with “code”.

RETURNS: “ok” or an error string.

2.4.40. removescenefromblock

project
scene
block

Unassociate a “scene” from “block” within “project”.RETURNS: “ok” or an error string.

RETURNS: “ok” or an error string.

2.4.41. rempermission

login
project

Remove permission for the student “login” within “project”.

RETURNS: “ok” or an error string.

2.4.42. replacecategory

sourcecat
destinationcat
project

Batch replace all occurrences of the “sourcecat” category with “destinationcat” category within “project”.

RETURNS: “ok” or an error string.

2.4.43. replaceimage

sourcecat
destinationcat
sourceimg
destinationimg

Batch replace all of the image sourceimg within category sourcecat, with image destinationimg within category destinationcat.

RETURNS: “ok” or an error string.

2.4.44. resetdata

name

Clear all recorded test data withinproject “name”.

RETURNS: “ok” or an error string.

2.4.45. savesql

name

Save a SQL file with filename “name” to disk.

This command expects a “data” tag containing the SQL script in plain text.

RETURNS: “ok” or an error string.

2.4.46. scenesinblock

project
block

Get a list of scenes currently belonging to “block” within “project”.

RETURNS: An array of “row” tags within the data section of the result XML. Each row describes a scene and contains the attribute “scene”, which is the name of a scene.

2.4.47. scenesnotinblock

project
block

Get a list of scenes currently not belonging to “block” within “project”.

RETURNS: An array of “row” tags within the data section of the result XML. Each row describes a scene and contains the attribute “scene”, which is the name of a scene.

2.4.48. updateproject

name
displaywelcome
displaythanks
welcometop
welcomemid
welcomebottom
thankstop
thanksmid
thanksbottom
urlredirect
maxwidth
maxheight
flashright
flashwrong
displaypolicy
subsetsize
flashwhite
whitemin
whitemax
hideopts
splicearray
blockrandom
rightimage
wrongimage
pauseimage

Saves settings for a project. The attributes are:

- name: The name of the project
- login: The login that created the project
- displaywelcome: Boolean for whether a text scene should be shown at the start of the project
- displaythanks: Boolean for whether a text scene should be shown at the end of the project
- welcometop: The first line of text of the welcome scene.
- welcomemid: The second line of text of the welcome scene.
- welcomebottom: The third line of text of the welcome scene.
- thankstop: The first line of text of the thanks scene.
- thanksmid: The second line of text of the thanks scene.
- thanksbottom: The third line of text of the thanks scene.
- urlredirect: URL to redirect to when project has finished
- maxwidth: Width in pixels to use for laying out a project
- maxheight: Height in pixels to use for laying out a project

- flashright: Boolean setting for flash a “correct” feedback on correct input
- flashwrong: Boolean setting for flash a “wrong” feedback on wrong input
- flashwhite: Boolean setting for showing a pause between scenes
- whitemin: Minimum time in milliseconds for pause between scenes
- whitemax: Maximum time in milliseconds for pause between scenes
- displaypolicy: Policy for displaying scenes where 1 is “all scenes in order”, 2 is “all scenes randomized and 3 “randomized subset. NOTE: this setting is deprecated and no longer used
- subsetsize: Number of scenes to pick for a subset with displaypolicy = 3. NOTE: this setting is deprecated and no longer used
- hideopts: Boolean for whether to always hide option texts in option scenes
- blockrandom: Boolean for whether blocks should be shown in random order
- rightcategory: Category to use when showing image instead of color when flashing “correct” feedback.
- rightimage: Image to use when showing image instead of color when flashing “correct” feedback.
- wrongcategory: Category to use when showing image instead of color when flashing “wrong” feedback.
- wrongimage: Image to use when showing image instead of color when flashing “wrong” feedback.
- pausecategory: Category to use when showing image instead of color when pausing between scenes.
- pauseimage: Image to use when showing image instead of color when pausing between scenes.
- splicearray: Whether to never show an image again during the rest of the project once it has been shown the first time.

RETURNS: “ok” or an error string.

2.4.49. updatescri

project
pattern
label
correct
timeout
noduplicates
scene
category[ ]
key[ ]

Update an existing SCRI scene with name “scene”. Here “label” is the (new) name of the new scene, “correct” is the keypress that should be considered correct, timeout is an integer representing the milliseconds after which the scene should be marked as failed if no keypress has occurred, “noduplicates” a boolean controlling whether the same image should be allowed to appear twice within the scene.

Category[ ] and key[ ] are the category and keypress representing each image on the scene.

RETURNS: “ok” or an error string.

2.4.50. updatesori

project
pattern
label
correct
timeout
numopts
lead
scene
option[ ]
key[ ]
category[ ]

Update an existing SORI scene with name “scene”. Here “label” is the (new) name of the new scene, “correct” is the keypress that should be considered correct, timeout is an integer representing the milliseconds after which the scene should be marked as failed if no keypress has occurred, “noduplicates” a boolean controlling whether the same image should be allowed to appear twice within the scene. “Lead” is the text representing the question line. “Numopts” is the number of options (and should correcpond to the length of option[ ] and key[ ]).

Category[ ] are the categories representing the images on the scene. Option[ ] and key[ ] are the options and their corresponding keys.

RETURNS: “ok” or an error string.

2.4.51. updatesosi

project
pattern
label
correct
timeout
numopts
lead
scene
option
key[ ]
category[ ]
file[ ]

Updates an existing SOSI scene with name “scene”. Here “label” is the (new) name of the new scene, “correct” is the keypress that should be considered correct, timeout is an integer representing the milliseconds after which the scene should be marked as failed if no keypress has occurred, “noduplicates” a boolean controlling whether the same image should be allowed to appear twice within the scene. “Lead” is the text representing the question line. “Numopts” is the number of options (and should correcpond to the length of option[ ] and key[ ]).

Category[ ] and file[] are the images on the scene (file is the name of the image within the category). Option[ ] and key[ ] are the options and their corresponding keys.

RETURNS: “ok” or an error string.

2.4.52. updatessi

```
project
pattern
label
correct
timeout
scene
category[ ]
file[ ]
key[ ]
```

Updates an existing SI scene with name “scene”. Here “label” is the (new) name of the new scene, “correct” is the keypress that should be considered correct, timeout is an integer representing the milliseconds after which the scene should be marked as failed if no keypress has occurred, “noduplicates” a boolean controlling whether the same image should be allowed to appear twice within the scene.

Category[ ] and file[] are the images on the scene (file is the name of the image within the category). Key[ ] are the corresponding keys.

RETURNS: “ok” or an error string.

2.4.53. updatestudent

```
login
pass
```

Set a new password “pass” for login “login”.

RETURNS: “ok” or an error string.

2.4.54. updatetext

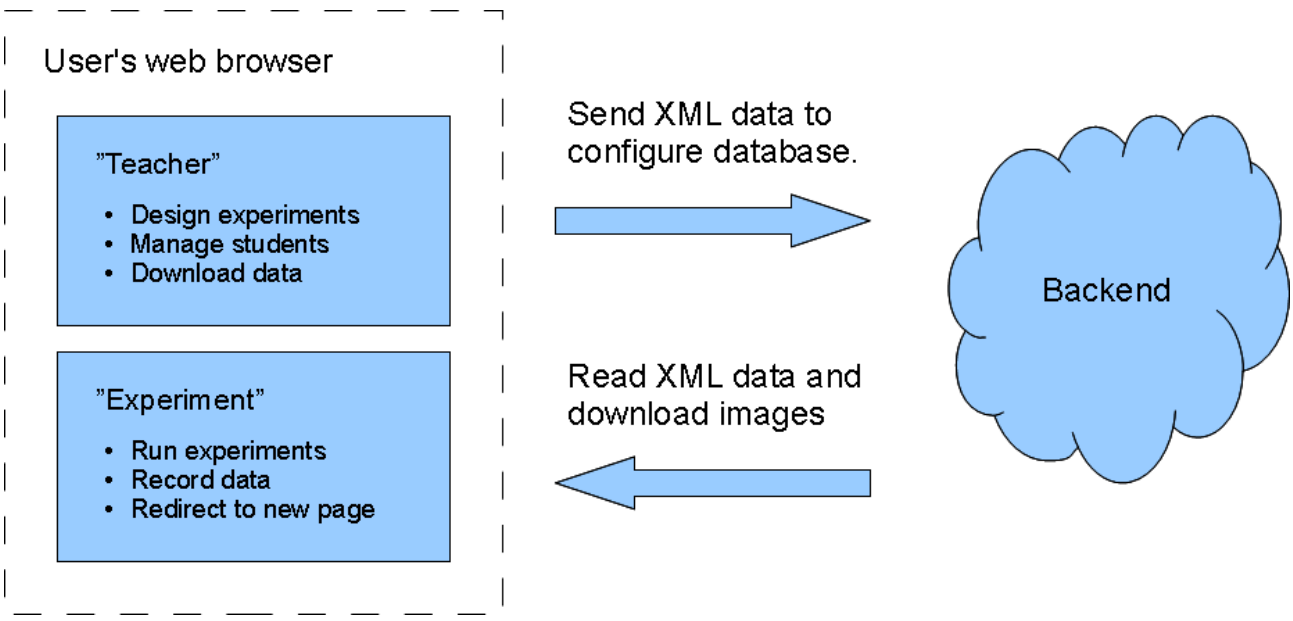
```
project
label
text
scene
```

Updates an existing text scene with name “scene”. Here “label” is the (new) name of the new scene and “text” is the new text for the scene.

RETURNS: “ok” or an error string.

3. Frontend

The frontend contains two parts, both communicating with the backend.



The frontend is run as flash applications in the user's web browser. In practice there are two applications: The “teacher” part which is used to design experiments, and the “experiment” part which is used to run experiments. Both applications communicates with the backend using mainly XML.

3.1. File hierarchy

The files are contained in the root of the PUPIL directory.

Path	**Description**
flash/images	Images belonging to projects
flash/pupil/reaction	Source code for the experiment parts of pupil
flash/pupil/common	Source code shared between experiment ant teacher
flash/pupil/teacher	Source code for the teacher parts of pupil
sql	SQL scripts for custom data exports

4. Installation

Installation of PUPIL should normally only be attempted by an experienced linux administrator with good knowledge of apache, mysql and tomcat. Depending on the base linux distribution, things may look very different in the file system hierarchy. The following is a general outline of the process.

4.1. Dependencies

PUPIL expects the following components to be installed. With other versions, things may or may not function properly.

- Apache 2.2
- Tomcat 6.0
- Sun JDK 1.6
- Mysql 5.0

4.2. Binary installation

It is here assumed that the person installing the software package has access to a zip file or similar containing the whole system with compiled binaries.

- To start, unzip the package in some reasonable place. Here we assume the package is unzipped into “/var/pupil”.
- Create an empty file “WEB-INF/servlet.log”
- Find the file WEB-INF/log.properties and check that output log file is “/var/pupil/WEB-INF/servlet.log”. If not, change the setting.
- Enable the servlet for tomcat. In debian this is done by placing an XML file in “/etc/tomcat6/Catalina/localhost”. See existing files there for format.

4.2.1. Setting up database

Before running this step, MySQL should be configured and you will need the MySQL root password handy.

Enter the directory "/var/pupil/WEB-INF/src". Run the command:

- "perl setupdb.pl"

This will create an initial empty database for PUPIL.

4.2.2. Setting up apache

Normally you would not access tomcat directly. Instead the usual approach is to have apache as a proxy frontend. If you do not want this, you can skip this section.

Make sure that proxying is enabled in apache.

Add the following to the appropriate VHOST section:

```
<Location /pupil/pupil>
  ProxyPass http://127.0.0.1:8080/pupil/pupil
  ProxyPassReverse http://127.0.0.1:8080/pupil/pupil
  Order allow,deny
  Allow from all
</Location>

<Location /pupil/sql>
  ProxyPass http://127.0.0.1:8080/pupil/sql
  ProxyPassReverse http://127.0.0.1:8080/pupil/sql
  Order allow,deny
  Allow from all
</Location>

<Location /pupil/flash>
  ProxyPass http://127.0.0.1:8080/pupil/flash
  ProxyPassReverse http://127.0.0.1:8080/pupil/flash
  Order allow,deny
  Allow from all
</Location>
```

In debian, this is normally added in /etc/apache2/sites-enabled/000-default.

4.3. Building from source

If you have access to the PUPIL source distribution the following is the general approach for building the system.

4.3.1. Build dependencies

In order to build the whole system you will need the following tools:

- Flex 3.5
- Ant (any version)

You will also need the following libraries (any reasonably modern version should do) :

- commons-fileupload
- commons-lang
- commons-io
- mysql JDBC
- log4j
- servlet-api

If you installed the binary distribution previously, chances are these files are available in /var/pupil/WEB-INF/lib

4.3.2. Compiling backend

To compile the backend enter the directory WEB-INF/src

Edit “build.xml” and check that the classpath specification is valid for your installation.

Run the following:

- ant clean
- ant

Remember to restart the servlet before trying to run the newly compiled code.

4.3.3. Compiling frontend

Compiling the frontend requires Flex 3.5 or later to be installed and that “mxmlc” is in the path. This is usually found in the “bin” folder in the flex distribution.

Enter /var/pupil/flash. To compile, run the following commands:

- mxmlc teacher.as
- mxmlc reaction.as

Note that you will have to clear your browser cache and/or restart the browser to see any changes in flash files.

5. Maintenance

Once installed, PUPIL should not require much maintenance. However, there are some things that might need doing.

5.1. Backups

Backups should take two things into account: The database and the image files.

To backup the database, use “mysqldump” to write the entire database to a file, for example:

- mysqldump --opt -u pupil -p pupil > pupilbackup.sql

To backup the image data, zip the contents of the image directory in the flash directory, for example:

- tar cfvz pupilimages.tgz /var/pupil/flash/images

5.2. Adding or removing teachers

There is currently no interface for adding or removing teachers. This has to be done via the mysql interface. Enter mysql:

- mysql -u pupil -p pupil

To insert a new teacher:

- INSERT INTO teacher(login,pass) VALUES('[login]','[password]');

To remove a teacher

- DELETE FROM teacher WHERE login = '[login]';