The demexp Book

or the internals of a voting system

David Mentré

22nd October 2006

Contents

Chapter 1

Introduction

> "La grande bataille dans la société où nous vivons n'oppose pas le bien et le mal mais l'intelligence et la connerie. Les forces de la connerie sont démentielles."
>
> Yves Michaud (Le Monde, 28th of August, 2001)

This book describes the internals of the demexp server and client.
todo: We do not handle deletion of responses in a question. TO DO

1.1 License

1.2 Credits

The idea of the Democratic Experience comes from Frédéric LEHOBEY and has been then designed and refined by Félix HENRY and Frédéric.

The demexp development team was initially made of David MENTRÉ.

Several people helped or are helping (by order of appearance):

- Isabelle OGER: first user interface;
- David DECOTIGNY;
- Serge LEBLANC: first CDuce code;
- Augustin MASQUILIER: demexp module for Drupal;

- Ketty: improvements on CGI web interface.

Of course, Frédéric and Félix helped us design the software, test it and solve programming issues.

Translations have been provided by:

- Serge LEBLANC, Emmanuelle RICHARD and Fabien TSCHUDY: Esperanto;
- Frédéric LEHOBEY: French.

Packages of demexp have been provided by:

- Thomas PETAZZONI: Debian and Ubuntu packages;
- Christophe GISQUET: Windows port and installer;
- Christophe GUILLOUX: Mandrake package;
- Thomas DE GRENIER DE LATOUR: Gentoo package.

We also would like to thank people that have indirectly contributed to this project:

- Norman Ramsey for his noweb package[1]. We hope this code is an example of good use of literate programming;
- Gerd Stolpmann for his RPC[2] and his WDialog[3] packages. Nearly all the code related to network encoding, decoding and data transfer is Gerd's one;
- Jean-Christophe Filliâtre for his Bitv[4] Bit Vector package;
- the CDuce team for CDuce[5] that allows to manipulate XML much more easily;
- the Savannah team for providing a useful development environment;
- Jacques Garrigue and the team around LablGTK2[6] for doing such a hard work;
- SooHyoung Oh for the adaptation of Gtk2 tutorials to OCaml;
- and last but not least, the people at INRIA who have developed such a wonderful language which is OCaml!

1.3 To do

Of first priority:

1.

Other various items to do on the whole system:

- user manual and tutorial;

Other various items to do on the compilation system:

- Makefile as literate file;

[1] http://www.eecs.harvard.edu/~nr/noweb/
[2] http://www.ocaml-programming.de/programming/rpc.html
[3] http://wdialog.sourceforge.net/
[4] http://www.lri.fr/~filliatr/software.en.html
[5] http://www.cduce.org/
[6] http://wwwfun.kurims.kyoto-u.ac.jp/soft/olabl/lablgtk.html

Other various items to do on the network messages:

- document use of RPC;

Other various items to do in server:

- define politics on information visibility: what should be available to a wide audience and what shouldn't?
- generate correct password hash (use a salt);
- should we deactivate accounts instead of removing them? What about votes of removed/deactivated voters?
- Anonymous should not be able to add new responses (DoS);
- increase maximum number of reponses to a question (DoS mitigation);
- save question and response creation date;
- add new delegation system;
- i18n and l10n of interface;
- i18n and l10n of position base;
- real security (at least, authentication and encryption between client and server);
- have access to database internals (see below);
- check for invariants;
- add an event system to schedule all computations, database saving, etc. Workers send to background thread the modification they have made and the background thread redo computations accordingly;
- add timout (1mn) to clients so that the whole server in never blocked;
- security audit (buffer overlow, hash uses, ...);
- make a tool that produces delegation graph suitable to `dot`;
- change literate programming tool to use a tool that start from .ml/.xdr/etc. files (and use ocamldoc for the ocaml part);
- check that stored data structure size is coherent with limits set by messages (create a module Limits that sets constants from net/Messages);
- performance measuring and related perf. issues;
- rewriting of condorcet voting to use Schulze algorithm;
- analysis of Condorcet voting;
- explanations on a vote result (like Debian's explanations);
- port to other Unix platforms;
- messages that could be added:
  - remove a participant;
  - list all delegates;

  - give size of in memory bases;
- put debug and log output into the same file. Mark debug output with [dbg];
- define criteria for question and response description validity and check for them;
- justification of our algorithms.

Other various items to do in GTK2 client:

- automatically get votes from server when .demexp/ is created from scratch;
- save timestamps of questions to avoid loosing them when the cache is cleared after a client upgrade;
- allow multiple servers to be manage graphically from the GUI;
- check link and reponse validity in client;
- add virtual tags that exist only in client, e.g. for new questions, questions on which we haven't voted yet;
- user doc;
- add progress bars;
- get several tags of a question in the same RPC call;
- support of internationalization;
- check that we cannot create wrong RPC messages: too much tags, too much reponses, etc.;
- use a real configuration file (see `Config_file` module from Cameleon2);
- i18n and l10n of interface;
- i18n and l10n of position base;
- scripting capabilities in the client.

Other various items to do on the web interface:

- better graphical layout (CSS);
- allow access to multiple servers;
- setup a configuration file (accessible servers);

Chapter 2

Client and Server configuration

Following OCaml module `Config` defines static configuration parameters used by both client and server code. This module is not in literate programming form because it is generated by the `./configure` script.

```
(* generated by ./configure --prefix /tmp --for-developer *)
(* if do_autotest is true, some automatic tests will be automatically
   done when a software is started *)
let do_autotests = true
(* if check_invariants is true, invariants on bases are checked at
   each base modification. Can be costly. *)
let check_invariants = true

(* config/config.inc.ml *)

(* copyright 2003-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

(* this file contains static definitions for configuration options of
   the demexp software. You can tweak them if needed *)

(* version number of the server *)
let server_version = "0.8.2 -- U2: Unscalable & Unsecure"

(* version number of the client *)
let client_version = "0.8.2"

(* port on which the server will listen to clients *)
(* please notice that, according to IANA port numbers assignement,
   private ports like this one should be from 49152 through 65535.
   http://www.iana.org/assignments/port-numbers *)
let default_server_port = 50000

(* Internet address (i.e. interface) on which the server listen to *)
let default_server_address = "127.0.0.1"

(* maximum number of clients the server is willing to handle
   simultaneously *)
let server_max_clients = 100

(* maximum size of an incoming message on server, to prevent denial of
   service *)
let maximum_message_size = 5 * 1024 * 1024 (* bytes *)
```

```
(* maximum number of copies of a base kept (under name .0, .1, ...,
   .maximum_file_rotation *)
let maximum_file_rotation = 2

(* default name for binary bases *)
let default_bases_name = "bases.dmxp"

(* default server name on which the client connects to *)
let default_server_name = "official.demexp.org"

(* maximum number of elements in an RSS feed *)
let maximum_size_of_rss_feed = 40

(* We print a message if autotests are available. We put this message
   here as the [[Config]] module is the first module (it is included
   by other modules) and therefore it will initialize itself first. *)
let _ =
  if do_autotests then Printf.eprintf "Autotests compiled.\n"
```

Part I

Network protocol (`net/`)

Chapter 3

Network messages in XDR format

We use RPC, Remote Procedure Call (IETF RFC 1831[1]), to define the message kind and content between DemExp client and server on the network. This method allows us to have a clear specification of each network messages, independant of any programming language. It should help maintain interoperability between clients and servers written in different programming languages, on different machines. Within RPC, messages are encoded using External Data Representation Standard (IETF RFC 1832[2]), which is portable.

Using the `ocamlrpcgen` program on `messages.xdr` file we generate the corresponding encoding and decoding OCaml routines: `Messages_aux` module contains general data type definitions as well as conversion functions, `Messages_clnt` module contains calling interface for a client and `Messages_srv` module contains callee interface for a server. `ocamlrpcgen` exists in Gerd Stolpmann RPC package.

3.1 Note on non standard `_int32` annotation

Due to its implementation in OCaml and choices made by demexp programmer, XDR `int` integers are mapped to 31 bits OCaml native integer on a 32-bits machine (see use of `ocamlrpcgen` in Makefile). Therefore, despite being marked as 32-bits XDR integer, those `int` are limited to 31 bits. As they are used as identifiers and those identifiers are created on the server, starting from zero, it will take some time to reach the maximum number $2^{30} - 1$ (about 1 million) of tags, questions or participants. This limitation is thus not an issue.

However, some XDR `int` are marked as `_int32`. Those XDR `int` are mapped to OCaml Int32 integers and are manipulated as 32-bits integer by the server.

If you use an implementation in another language that maps XDR `int` into 32-bits integer, you are on the safe side and can ignore non-standard `_int32` annotation[3].

3.2 Data types

We define in this section all messages necessary for a client to browse the position base and vote on a question.

14 ⟨*messages.xdr* 14⟩≡ 15a ▷

```
/* copyright 2003-2005 David MENTRE */
/* this software is under GNU GPL. See COPYING.GPL file for details */
```

[1] http://www.ietf.org/rfc/rfc1831.txt
[2] http://www.ietf.org/rfc/rfc1832.txt
[3] for example with a `#define _int32 /*nothing*/`.

3.2.1 General definitions

The protocol between the client and the server has a version number. This number is modified each time an incompatible change is made.

15a ⟨*messages.xdr* 14⟩+≡ ◁14 15b ▷

```
const PROTOCOL_VERSION = 1;
```

We set here size limits on all exchanged information to avoid the handling of unnecessary big datastructure at the receiving end. While XDR allows definition of variable length data structure of unknown size, it seems stupid (and dangerous from a Denial of Service risk handling perspective) to be prepare to handle the maximum size: $2^{32} - 1$. Therefore we define a reasonable size for each data structure.

15b ⟨*messages.xdr* 14⟩+≡ ◁15a 15c ▷

```
const MAX_LOGIN_SIZE = 64; /* characters */
const MAX_PASSWORD_SIZE = 64; /* characters */
const MAX_EXPLANATION_SIZE = 255; /* characters */
const MAX_QUESTION_SIZE = 1024; /* characters */
const MAX_RESPONSE_STRING_SIZE = 1024; /* characters */
const MAX_TAG_LABEL_SIZE = 255; /* characters */
const MAX_NUMBER_RESPONSES = 2048; /* number of responses */
const MAX_NUMBER_QUESTIONS = 200; /* number of questions */
const MAX_EXTERNAL_LINK_SIZE = 255; /* characters */
const MAX_NUMBER_WINNING_RESPONSES = 10; /* number of responses */
const MAX_GROUP_NAME_SIZE = 255; /* characters */
const MAX_NUMBER_GROUPS = 20; /* number of groups */
const MAX_NUMBER_IDS = 100; /* number of information fields */
const MAX_NUMBER_TAGS_PER_QUESTION = 200; /* number of tags per question */
const MAX_COMPRESSED_SIZE = 1048576; /* in bytes */
const MAX_TIMERS_STRING_SIZE = 65536; /* characters */
```

We define some basic data types used throughout following messages.

15c ⟨*messages.xdr* 14⟩+≡ ◁15b 16a ▷

```
typedef string login_t<MAX_LOGIN_SIZE>;
typedef string password_t<MAX_PASSWORD_SIZE>;

typedef int question_id_t;
typedef string question_desc_t<MAX_QUESTION_SIZE>;
typedef string response_desc_t<MAX_RESPONSE_STRING_SIZE>;
typedef hyper date_t; /* as number of seconds since since
                         00:00:00 GMT, Jan. 1, 1970 */
typedef string external_link_t<MAX_EXTERNAL_LINK_SIZE>;
typedef int vote_choice_t<MAX_NUMBER_RESPONSES>;

typedef string group_name_t<MAX_GROUP_NAME_SIZE>;
typedef group_name_t groups_t<MAX_NUMBER_GROUPS>;

typedef string tag_label_t<MAX_TAG_LABEL_SIZE>;
typedef int tag_set_t<MAX_NUMBER_TAGS_PER_QUESTION>;
typedef string server_timers_t<MAX_TIMERS_STRING_SIZE>;
```

An anonymous login is encoded as "`anonymous`" character string.

We define in `return_code_t` error codes that can be returned by remote procedure calls. The error code should be sufficient to translate error message in any foreign language.

16a ⟨*messages.xdr* 14⟩+≡ ◁15c 16b▷

```
enum return_code_t {
  rt_ok = 200,

  rt_generic_client_error = 400,/* it's client's fault */
  rt_not_enough_rights = 401, /* client hasn't the right to do that */

  rt_already_exists = 402, /* we wanted to some information but it
                              already exists on server */
  rt_not_found = 403, /* we looked for some information but it does not exist
                         on the server */
  rt_request_too_much_ids = 404, /* we request information on a number of
                                    ids which is bigger than MAX_NUMBER_IDS */
  rt_bad_status = 405, /* the provided new status to set_question_status()
                          is not tagging_only neither public */

  rt_vote_choice_not_found = 406, /* we included in our vote a response number
                                     that does not exists on the server */
  rt_duplicate_vote_choice = 407, /* we included the same response more than
                                     once in our vote */
  rt_anonymous_cannot_vote = 408, /* Anonymous user cannot vote */
  rt_bad_format = 409, /* a login, question or response is not formated
                          as expected */
  rt_bad_login = 410, /* the server could not authenticate the client */

  rt_generic_server_error = 500 /* it's server's fault */ };
```

3.2.2 RPC calls return types

In the following code fragments, the structure *method*`_return_t` defines the returned type for RPC *method*. All of those data structures begins with a field containing the status of the answer (`rt_ok` or error). For all RPC requesting information for a set of items using `base` + `number` scheme, `rt_request_too_much_ids` is returned if `number` is bigger than `MAX_NUMBER_IDS`.

todo: Looking at `/usr/include/rpcsvc/ *.x` on my Linux system, I see that returned argument to RPC methods are using a union type to discriminate between a normal case and error cases. Might be a good enhancement in the future. TO DO

3.2.3 Login

16b ⟨*messages.xdr* 14⟩+≡ ◁16a 17a▷

```
struct login_return_t {
  return_code_t login_return_code;
  int server_protocol_version;
  cookie_t login_cookie;
};
```

3.2.4 Timestamps

17a ⟨*messages.xdr* 14⟩+≡ ◁16b 17b▷

```
struct get_timestamps_return_t {
  return_code_t gt_return_code;
  opaque gt_participant<MAX_COMPRESSED_SIZE>;
  opaque gt_question<MAX_COMPRESSED_SIZE>;
  opaque gt_tag<MAX_COMPRESSED_SIZE>;
};
```

3.2.5 Questions

17b ⟨*messages.xdr* 14⟩+≡ ◁17a 17c▷

```
struct question_id_return_t {
  return_code_t question_id_return_code;
  question_id_t question_id_id;
};
```

17c ⟨*messages.xdr* 14⟩+≡ ◁17b 17d▷

```
struct max_question_id_return_t {
  return_code_t max_question_id_rc;
  question_id_t max_question_id;
};
```

17d ⟨*messages.xdr* 14⟩+≡ ◁17c 17e▷

```
struct response_t {
  response_desc_t r_info_desc;
  external_link_t r_info_link;
};

enum question_status_e { tagging_only = 1, public = 2 };

struct question_t {
  _int32 int q_timestamp;
  question_id_t q_id;
  question_desc_t q_desc;
  date_t q_info_limit_date; /* if set to zero, there is no limit date */
  question_status_e q_info_status;
  response_t q_info_responses<MAX_NUMBER_RESPONSES>;
  int q_info_num_votes;
  int q_info_elected_responses<MAX_NUMBER_RESPONSES>;
};

struct question_info_return_t {
  return_code_t question_info_rc;
  question_t question_info<MAX_NUMBER_IDS>;
};
```

17e ⟨*messages.xdr* 14⟩+≡ ◁17d 18a▷

```
struct get_vote_return_t {
  return_code_t get_vote_rc;
  vote_choice_t get_vote;
};
```

3.2.6 Participants

18a ⟨*messages.xdr* 14⟩+≡ ◁17e 18b▷

```
struct max_participant_id_return_t {
  return_code_t max_participant_id_rc;
  int max_participant_id;
};
```

18b ⟨*messages.xdr* 14⟩+≡ ◁18a 18c▷

```
struct add_participant_return_t {
  return_code_t add_participant_rc;
  int add_participant_id;
};
```

18c ⟨*messages.xdr* 14⟩+≡ ◁18b 18d▷

```
struct info_on_participant_t {
  int info_id;
  _int32 int info_timestamp;
  login_t info_login;
  password_t info_password;
  groups_t info_groups;
};

struct participant_info_return_t {
  return_code_t participant_info_rc;
  info_on_participant_t participant_info<MAX_NUMBER_IDS>;
};
```

3.2.7 Tags

18d ⟨*messages.xdr* 14⟩+≡ ◁18c 18e▷

```
struct max_tag_id_return_t {
  return_code_t max_tag_id_rc;
  int max_tag_id;
};
```

18e ⟨*messages.xdr* 14⟩+≡ ◁18d 18f▷

```
struct create_tag_return_t {
  return_code_t create_tag_rc;
  int create_tag_id;
};
```

18f ⟨*messages.xdr* 14⟩+≡ ◁18e 19a▷

```
struct info_on_tag_t {
  int a_tag_id;
  _int32 int a_tag_timestamp;
  tag_label_t a_tag_label;
};

struct tag_info_return_t {
  return_code_t tag_info_rc;
  info_on_tag_t tag_info<MAX_NUMBER_IDS>;
};
```

3.2.8 Tagging

19a ⟨*messages.xdr* 14⟩+≡ ◁18f 19b▷
```
struct question_tag_set_t {
  question_id_t tag_set_qid;
  tag_set_t tag_set;
};

struct tag_set_group_t {
  return_code_t tag_set_group_rc;
  question_tag_set_t tag_set_group<MAX_NUMBER_IDS>;
};
```

3.2.9 Cookie

Each method is using a cookie generated at first login that allows to maintain connection context on the server side.

19b ⟨*messages.xdr* 14⟩+≡ ◁19a 19c▷
```
typedef unsigned int cookie_t;
```

3.3 Method definitions

We include all following methods in the same program and version.

19c ⟨*messages.xdr* 14⟩+≡ ◁19b 19d▷
```
program Demexp {
  version V1 {
```

In the following code, all methods except login use as first argument a cookie that allows to maintain the client context on the server side.

3.3.1 Login management

Method `login` is used by a client to register oneself onto the server. The client gives its `login_t` and `password_t` for authentication. **fixme**: Yes, I know it is not safe at all. It is just dummy security for now. You have be warned! The client also gives the version of the protocol it is handling. The server replies with a `cookie_t` which should be used in all further method call and the version of the protocol used by the server. FIXME

19d ⟨*messages.xdr* 14⟩+≡ ◁19c 19e▷
```
    login_return_t login(int/*client_protocol_version*/,
                         login_t, password_t) = 0;
```

Method `goodbye` is to be called before a client finishes with the server. It allows the latter to cleanup the client context. It should be call ones for all cookies received by a `login()` RPC.

19e ⟨*messages.xdr* 14⟩+≡ ◁19d 19f▷
```
    void goodbye(cookie_t) = 1;
```

3.3.2 Timestamps

Method `get_timestamps` returns timestamps of all the objects on the server. The information returned can be old up to 2 minutes.

19f ⟨*messages.xdr* 14⟩+≡ ◁19e 20a▷
```
     get_timestamps_return_t get_timestamps(cookie_t) = 400;
```

3.3.3 Question management

Method `new_question` creates in the Position base a new question of title `question_desc_t`. It returns `rt_ok` if all went well or `rt_already_exists` if question with same title is in the base.

20a ⟨*messages.xdr* 14⟩+≡ ◁19f 20b▷
```
question_id_return_t new_question(cookie_t, question_desc_t) = 2;
```

Method `get_question_id` returns the question identifier of question identified by its title `question_desc_t`. The return code of `get_question_id_return_t` is equal to `rt_question_not_found` if the question searched for cannot be found.

20b ⟨*messages.xdr* 14⟩+≡ ◁20a 20c▷
```
question_id_return_t get_question_id(cookie_t, question_desc_t) = 3;
```

Method `add_response` add a new response of content `response_desc_t` and with (optional) `external_link_t` for `question_id_t`. It uses the current login identity as response author. It can return: `rt_ok`, `rt_not_found` or `rt_already_exists`.

20c ⟨*messages.xdr* 14⟩+≡ ◁20b 20d▷
```
return_code_t add_response(cookie_t, question_id_t,
                           response_desc_t, external_link_t) = 4;
```

Method `max_question_id` returns the biggest question identifier on the server.

20d ⟨*messages.xdr* 14⟩+≡ ◁20c 20e▷
```
max_question_id_return_t max_question_id(cookie_t) = 5;
```

Method `question_info` returns the information for the set of questions of which identifiers are in [`base_id`, `base_id` + `number`). It returns `rt_request_too_much_ids` if `number` is greater than `MAX_NUMBER_IDS`, `rt_ok` otherwise.

20e ⟨*messages.xdr* 14⟩+≡ ◁20d 20f▷
```
question_info_return_t question_info(cookie_t, int/*base_id*/,
                                     int/*number*/) = 6;
```

Method `set_question_status` update the status (`public` or `tagging_only`) for question of identifier `question_id`. It returns `rt_ok` if no issue, `rt_not_found` if the question identifier is invalid and `rt_bad_status` if the new status is invalid.

20f ⟨*messages.xdr* 14⟩+≡ ◁20e 20g▷
```
return_code_t set_question_status(cookie_t, question_id_t,
                                  question_status_e) = 7;
```

Method `vote` registers on server the vote on question of identifier `question_id_t` for participant identified by `cookie_t`. The `vote_choice` is expressed as an ordered array of integers, preferred choices first.

This method can return `rt_vote_choice_not_found`, `rt_duplicate_vote_choice`, `rt_anonymous_cannot_vo` and `rt_not_found` (in case the `question_id` is invalid).

20g ⟨*messages.xdr* 14⟩+≡ ◁20f 20h▷
```
return_code_t vote(cookie_t, question_id_t, vote_choice_t) = 8;
```

Method `get_vote` returns vote belonging to `login_t` in question of identifier `question_id_t`. To get a vote, either it is a public vote of a delegate, or either it is the requester's own vote. Otherwise `rt_not_enough_rights` is returned. If the question identifier is invalid, `rt_not_found` is returned.

20h ⟨*messages.xdr* 14⟩+≡ ◁20g 21a▷
```
get_vote_return_t get_vote(cookie_t, question_id_t, login_t) = 9;
```

3.3.4 Participants management (for administrators)

In all following methods, the client should have administrator rights to do them (and in that case returns `rt_ok`). If this is not the case, it returns `rt_not_enough_rights`.

Method `max_participant_id` returns a structure containing the biggest participant identifier on the server.

21a ⟨*messages.xdr* 14⟩+≡ ◁20h 21b▷
```
max_participant_id_return_t max_participant_id(cookie_t) = 100;
```

Method `participant_info` returns the details of participants whose identifiers are between `base_id` and `base_id` + `number` (excluded).

21b ⟨*messages.xdr* 14⟩+≡ ◁21a 21c▷
```
participant_info_return_t participant_info(cookie_t,
                                           int /*base_id*/,
                                           int /*number*/) = 101;
```

Method `add_participant` add a new participant in the participant base. It returns `rt_already_exists` error code in case the participant is already in the database. If no error, it returns the identifier of the new participant.

21c ⟨*messages.xdr* 14⟩+≡ ◁21b 21d▷
```
add_participant_return_t add_participant(cookie_t, login_t,
                                         password_t, groups_t) = 102;
```

Method `update_participant` update information of participant (password and groups) with login `login_t` to the Participant base. The handling of `password` field is quite specific. Look at code chunk 197b for details.

21d ⟨*messages.xdr* 14⟩+≡ ◁21c 21e▷
```
return_code_t update_participant(cookie_t, login_t,
                                 password_t, groups_t) = 103;
```

Method `remove_participant` removes a participant on the server.

21e ⟨*messages.xdr* 14⟩+≡ ◁21d 21f▷
```
return_code_t remove_participant(cookie_t, login_t) = 104;
```

3.3.5 Tag management (for administrators)

In all following methods, the client should have administrator rights to do them. If this is not the case, it returns `rt_not_enough_rights`.

Method `max_tag_id` returns a structure containing the biggest tag identifier on the server.

21f ⟨*messages.xdr* 14⟩+≡ ◁21e 21g▷
```
max_tag_id_return_t max_tag_id(cookie_t) = 200;
```

Method `create_tag` adds a new tag on the server. It can returns the identifier of the new tag if successful. In case the tag already exists, it returns `rt_already_exists`.

21g ⟨*messages.xdr* 14⟩+≡ ◁21f 21h▷
```
create_tag_return_t create_tag(cookie_t, tag_label_t) = 201;
```

Method `tag_info` returns the details of tags whose identifiers are between `base_id` (included) and `base_id` + `number` (excluded).

21h ⟨*messages.xdr* 14⟩+≡ ◁21g 21i▷
```
tag_info_return_t tag_info(cookie_t, int /*base_id*/, int /*number*/) = 202;
```

Method `update_tag` change label of tag with identifier `tag_id` with new label `tag_label`. It can return `rt_not_found` if the tag identifier is not found, otherwise `rt_ok`.

21i ⟨*messages.xdr* 14⟩+≡ ◁21h 22a▷
```
return_code_t update_tag(cookie_t, int /*tag_id*/, tag_label_t) = 203;
```

3.3.6 Question tagging (partly for classifiers)

In all following methods, the client should have classifier rights to do them. If this is not the case, it returns `rt_not_enough_rights`.

Method `tag_question` add `tag_id` to the set of tags of `question_id`. It returns `rt_not_found` if the tag id is not found, `rt_ok` otherwise. There is no check of validity of question id.

22a ⟨*messages.xdr* 14⟩+≡ ◁21i 22b▷

```
return_code_t tag_question(cookie_t, question_id_t, int /*tag_id*/) = 300;
```

Method `tag_question` removes `tag_id` from the set of tags of `question_id`. It returns `rt_ok` in all case, even if the tag or question id is not valid.

22b ⟨*messages.xdr* 14⟩+≡ ◁22a 22c▷

```
return_code_t untag_question(cookie_t, question_id_t, int /*tag_id*/) = 301;
```

Method `get_question_tags` returns the set of tag identifiers attached to question of identifier `question_id`. It returns an empty array if the `question_id` is invalid.

22c ⟨*messages.xdr* 14⟩+≡ ◁22b 22d▷

```
tag_set_t get_question_tags(cookie_t, question_id_t) = 302;
```

Method `tag_set_of_question_group` returns the set of tag identifiers attached to each question of which identifiers are between `base_id` (included) and `base_id` + `number` (excluded). The returned `tag_set_group_t` can be empty if the question identifiers are invalid or if no tags are attached to them.

22d ⟨*messages.xdr* 14⟩+≡ ◁22c 22e▷

```
tag_set_group_t tag_set_of_question_group(cookie_t, int /*base_id*/,
                                          int /*number*/) = 303;
```

3.3.7 Server management (for administrators)

In all following methods, the client should have administrator rights to do them. If this is not the case, it returns `rt_not_enough_rights`.

Method `stop_server`, when called by a client, stop the server program.

22e ⟨*messages.xdr* 14⟩+≡ ◁22d 22f▷

```
return_code_t stop_server(cookie_t) = 900;
```

Method `server_timers` returns a string containing the value of all performance timers measured on the server.

22f ⟨*messages.xdr* 14⟩+≡ ◁22e 22g▷

```
server_timers_t server_timers(cookie_t) = 901;
```

3.4 End of RPC definitions

Program `Demexp` is given version 0x20000000 (it should be in the 0x20000000 - 0x3FFFFFFF range, User-defined range[4]) and version `V1` is given version number 1.

Note: Once the protocol is stabilized, we could ask for a static protocol number at SUN (rpc@sun.com).

22g ⟨*messages.xdr* 14⟩+≡ ◁22f

```
    } = 1;
} = 0x20000000;
```

[4] according to "Power Programming with RPC", John Bloomer, O'Reilly & Associates.

Part II

Common code (`lib/`)

Chapter 4

Support of translation (DemexpGettext)

Module DemexpGettext provides configuration of ocaml-getttext for translation of demexp.

24 ⟨*demexpGettext.ml* 24⟩≡

```
(* copyright 2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

module Gettext = Gettext.Program
    (struct
      let textdomain   = "demexp"
      let codeset      = Some "UTF-8"
      let dir          =
        match Sys.os_type with
          "Win32" | "Cygwin" -> Some "locale"
        | _                  -> None
      let dependencies = Gettext.init
    end)
    (GettextStub.Native)
```

Chapter 5

Performance measurement (`Perf`)

Module `Perf` defines routines to measure performance of the different parts of the server and the client.

5.1 Time measurement

Our time measurement routines are built upon `Unix.gettimeofday` function. A `timer` contains a function that, when called with unit parameter, returns the elapsed time in seconds since its creation.

25a ⟨*perf.ml* 25a⟩≡ 25b ▷
```
type timer = unit -> float
```

When we start a timer with `timer_start`, we record the starting time in `start` and return the function that will compute the elapse time.

25b ⟨*perf.ml* 25a⟩+≡ ◁25a 25c ▷
```
let timer_start () =
  let start = Unix.gettimeofday () in
  fun () -> Unix.gettimeofday () -. start
```

Function `time_as_string` returns the `time` in seconds as a string formatted for human reading, with readable units (nanoseconds: `ns`, microseconds: `us`, milliseconds: `ms`, seconds: `s`).

25c ⟨*perf.ml* 25a⟩+≡ ◁25b 26a ▷
```
let time_as_string time =
  match time with
  | v when v < 0.000001 ->
      Printf.sprintf "%3.3f ns" (v *. 1000000000.0)
  | v when v < 0.001 ->
      Printf.sprintf "%3.3f us" (v *. 1000000.0)
  | v when v >= 0.001 && v < 1. ->
      Printf.sprintf "%3.3f ms" (v *. 1000.0)
  | v ->
      Printf.sprintf "%3.3f s" v
```

5.2 Timing storage

We also define a data structure used to store measured times, indexed by timer names. From all the stored measures, we want to extract maximum, minimum and average time.

Firstly, all the timer measures as stored in a structure that records `min`imum and `max`imum values, the `sum` of all recorded values as well as the `number` of them.

26a ⟨*perf.ml* 25a⟩+≡ ◁25c 26b▷

```
type timer_measure = {
    mutable min : float;
    mutable max : float;
    mutable sum : float;
    mutable number : int;
  }
```

The global data structure that store measures, `measure_storage`, is a hash table from timer names to recorded measures.

26b ⟨*perf.ml* 25a⟩+≡ ◁26a 26c▷

```
let measure_storage : (string, timer_measure) Hashtbl.t = Hashtbl.create 3
              (* timer_name -> measures *)
```

Function `record_measure` stores a new measure `delta` for `timer_name`. It simply updates the `min`, `max`, `sum` and `number` fields of it.

26c ⟨*perf.ml* 25a⟩+≡ ◁26b 26d▷

```
let record_measure timer_name delta =
  let update_measure measure delta =
    if delta < measure.min then measure.min <- delta;
    if delta > measure.max then measure.max <- delta;
    measure.sum <- measure.sum +. delta;
    measure.number <- measure.number + 1 in
  let empty_measure =
    { min = infinity; max = neg_infinity; sum = 0.; number = 0 } in
  try
    let m = Hashtbl.find measure_storage timer_name in
    update_measure m delta
  with Not_found ->
    let m = empty_measure in
    update_measure m delta;
    Hashtbl.add measure_storage timer_name m
```

Function `timer_stop_and_record` stops the `timer` and records the elapsed time for `timer_name`.

26d ⟨*perf.ml* 25a⟩+≡ ◁26c 26e▷

```
let timer_stop_and_record timer_name timer =
  record_measure timer_name (timer ())
```

Function `get_min_avg_max` returns the minimum, average and maximum of a recored values stored for `timer_name`. It raises `Not_found` exception if no measure exists for `timer_name`.

26e ⟨*perf.ml* 25a⟩+≡ ◁26d 27a▷

```
let get_min_avg_max timer_name =
  let measure = Hashtbl.find measure_storage timer_name in
  (measure.min, measure.sum /. (float_of_int measure.number), measure.max)
```

Function `timers_as_string` returns a string that contains all stored measures (minimum, average and maximum), sorted by timer name.

27a ⟨*perf.ml* 25a⟩+≡ ◁26e 27b▷

```
let timers_as_string () =
  let timer_as_string name measure =
    let min, avg, max = (measure.min,
                         measure.sum /. (float_of_int measure.number),
                         measure.max) in
    Printf.sprintf "%s\t %s\t %s\t (%d) %s\n"
      (time_as_string min) (time_as_string avg) (time_as_string max)
      measure.number name in
  let timers =
    Hashtbl.fold
      (fun name measure l -> (name, (timer_as_string name measure)) :: l)
      measure_storage [] in
  let sorted_timers = List.sort compare timers in
  let buf = Buffer.create 3 in
  Buffer.add_string buf (Printf.sprintf "min\t\tavg\t\tmax\t\tcount & name\n");
  Buffer.add_string buf (Printf.sprintf "---\t\t---\t\t---\t\t-----   ----\n");
  List.iter (fun (_, str) -> Buffer.add_string buf str) sorted_timers;
  Buffer.contents buf
```

Function `print_timers` print all stored measures (minimum, average and maximum) on standard output.

27b ⟨*perf.ml* 25a⟩+≡ ◁27a 27c▷

```
let print_timers () =
  Printf.printf "%s" (timers_as_string ())
```

5.3 Automatic tests

27c ⟨*perf.ml* 25a⟩+≡ ◁27b

```
let _ =
  if Config.do_autotests then begin
    Printf.eprintf "  perf autotests...";
    assert(time_as_string 0.000000001 = Printf.sprintf "%.3f ns" 1.0);
    assert(time_as_string 0.000001 = Printf.sprintf "%.3f us" 1.0);
    assert(time_as_string 0.001 = Printf.sprintf "%.3f ms" 1.0);
    assert(time_as_string 1. = Printf.sprintf "%.3f s" 1.0);
    record_measure "a" 1.;
    record_measure "a" 2.;
    assert(get_min_avg_max "a" = (1., 1.5, 2.));
    (try
      ignore(get_min_avg_max "b");
      assert(false)
    with Not_found -> ());
    (* example of use of timer *)
    let t = timer_start () in
      (* do something *)
    timer_stop_and_record "c" t;
    (* cleanup data structure *)
    Hashtbl.clear measure_storage;
    Printf.eprintf "done\n"
  end
```

Chapter 6

Normalization of user input

Module `Norm` provides routines to normalize and to check the correctness of a descriptor (of a question or a response), a login, etc.

For each type of information to check, we provide two functions:

`normalize_*:` to normalize an entry, removing ambiguities;

`check_*:` to check that an entry is valid.

An exception `Invalid_format` is raised when a check fails.

28a ⟨*norm.ml* 28a⟩≡ 28b ▷
```
exception Invalid_format
```

28b ⟨*norm.ml* 28a⟩+≡ ◁28a 28c ▷
```
(* copyright 2005 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Str
```

6.1 Question

In order to normalize a question, we define regular expressions that match respectively multiple spaces in the middle, at the beginning and at the end of a question description.

28c ⟨*norm.ml* 28a⟩+≡ ◁28b 28d ▷
```
let multiple_spaces_regex = regexp "[\t\n ]+"

let leading_spaces_regex = regexp "^[\t\n ]+"

let trailing_spaces_regex = regexp "[\t\n ]+$"
```

The normalization operation of a question descriptor consists in removing all multiple spaces and leading and trailing spaces.

28d ⟨*norm.ml* 28a⟩+≡ ◁28c 28e ▷
```
let normalize_question ~q_desc =
  let a = global_replace multiple_spaces_regex " " q_desc in
  let b = global_replace leading_spaces_regex "" a in
  global_replace trailing_spaces_regex "" b
```

A valid question is a non empty one.

28e ⟨*norm.ml* 28a⟩+≡ ◁28d 29a ▷
```
let check_question ~q_desc =
  if q_desc = "" then raise Invalid_format
```

6.2 Response

For a response, we apply the same formatting as for a question.

29a ⟨*norm.ml* 28a⟩+≡ ◁28e 29b▷
```
let normalize_response ~r_desc = normalize_question ~q_desc:r_desc

let check_response ~r_desc = check_question ~q_desc:r_desc

let is_valid_response ~r_desc =
  try check_response ~r_desc; true with Invalid_format -> false
```

6.3 Login

To normalize a login, we remove all spaces.

29b ⟨*norm.ml* 28a⟩+≡ ◁29a 29c▷
```
let normalize_login login =
  global_replace multiple_spaces_regex "" login
```

A valid login as only characters in `A` to `Z` range, `a` to `z` range, `.`, `-` or `_`, starting with at least one ASCII character.

29c ⟨*norm.ml* 28a⟩+≡ ◁29b 29d▷
```
let valid_login_regex = regexp "^[A-Za-z][-A-Za-z_.]*$"

let check_login login =
  if not(string_match valid_login_regex login 0) then raise Invalid_format
```

6.4 Tag

We assume that the specific tag associated to each question is of the form "question *nn*" where *nn* in a number.

29d ⟨*norm.ml* 28a⟩+≡ ◁29c 29e▷
```
let question_specific_tag_regexp = regexp "^question [0-9]+$"
```

Function `is_question_specific_tag` returns `true` is the given `label` is of the same form as the specific tag associated to each question.

29e ⟨*norm.ml* 28a⟩+≡ ◁29d 29f▷
```
let is_question_specific_tag label =
  string_match question_specific_tag_regexp label 0
```

6.5 Link

When normalizing a link, we simply remove leading, trailing and multiple spaces.

29f ⟨*norm.ml* 28a⟩+≡ ◁29e 30a▷
```
let normalize_link link = normalize_question ~q_desc:link
```

We make a simple check on web links, assuming they are of the form: `http://server.com:5000/path/to/docu`
We only allow ASCII character set.

30a ⟨*norm.ml* 28a⟩+≡ ◁29f 30b▷

```
let valid_link_regex =
  regexp "^http://[-A-Za-z0-9_.]+\\(:[0-9]+\\)?[-A-Za-z0-9+&:;@_.%=?/]*$"

let check_link link =
  if link <> ""
     && not(string_match valid_link_regex link 0) then raise Invalid_format

let is_valid_link link =
  try check_link link; true with Invalid_format -> false
```

6.6 Automatic tests

30b ⟨*norm.ml* 28a⟩+≡ ◁30a

```
let _ =
  if Config.do_autotests then begin
    Printf.eprintf "  norm autotests...";
    assert(normalize_question ~q_desc:"  \n\n \t Hello \n\t  world\n  !  \n"
             = "Hello world !");
    (try check_question ~q_desc:""; assert(false) with Invalid_format ->());
    assert(normalize_response ~r_desc:"  \n\n \t Hello \n\t  world\n  !  \n"
             = "Hello world !");
    assert(normalize_response ~r_desc:"  " = "");
    (try check_response ""; assert(false) with Invalid_format ->());
    assert(normalize_login "\t  \n A - wonderful . login _ \n \t "
             = "A-wonderful.login_");
    check_login "A.valid-login_";
    (try check_login "-invalid_login"; assert(false) with Invalid_format ->());
    (try check_login "\ninvalid\n"; assert(false) with Invalid_format ->());
    check_login "a";
    assert(is_question_specific_tag "question 3" = true);
    assert(is_question_specific_tag "question a" = false);
    assert(is_question_specific_tag "question 33 " = false);
    assert(normalize_link "  http://www.demexp.org  "
             = "http://www.demexp.org");
    check_link "http://www.demexp.org";
    check_link "http://www.demexp.org:80";
    check_link "http://www.demexp.org/rubrique.php3?id_rubrique=2";
    check_link "http://www.demexp.org:80/rubrique.php3?id_rubrique=2";
    check_link "http://127.0.0.1:80/rubrique.php3?id_rubrique=2";
    Printf.eprintf "done\n"
  end
```

Chapter 7

Manipulation of time

Module `Time` provides routines to manipulate time.

31a ⟨*time.ml* 31a⟩≡ 31b ▷
```
(* copyright 2005 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)
```

Variable `localtime_timezone` stores the local time offset.

31b ⟨*time.ml* 31a⟩+≡ ◁31a 31c ▷
```
let localtime_timezone =
  let offset = Unix.time () in
  let time = Unix.localtime offset
  and utc_time = Unix.gmtime offset in
  Printf.sprintf "%+03d%02d"
    (time.Unix.tm_hour - utc_time.Unix.tm_hour)
    (time.Unix.tm_min - utc_time.Unix.tm_min)
```

Function `time_as_localtime_iso_string` converts `offset` in international ISO-8601 string format, with indication of local time offset.

31c ⟨*time.ml* 31a⟩+≡ ◁31b 31d ▷
```
let time_as_localtime_iso_string offset =
  let time = Unix.localtime offset in
  Printf.sprintf "%04d-%02d-%02dT%02d:%02d:%02d%s"
    (1900 + time.Unix.tm_year) (time.Unix.tm_mon + 1) (time.Unix.tm_mday)
    (time.Unix.tm_hour) (time.Unix.tm_min) (time.Unix.tm_sec)
    localtime_timezone
```

7.1 Automatic tests

31d ⟨*time.ml* 31a⟩+≡ ◁31c
```
let _ =
  if Config.do_autotests then begin
    Printf.eprintf "  time autotests...";
    Printf.eprintf "done\n"
  end
```

Chapter 8

Timestamp

Module `Timestamp` provides routines to manipulate timestamps and timestamp zones, used to know the freshness of objects in client cache.

32 ⟨*timestamp.ml* 32⟩≡ 33a ▷

```
(* copyright 2005-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Int32
```

As suggested by Félix Henry[1], we use a brute force approach to know if client cache objects are up-to-date with the server.

Each time an object is modified, we update its associated timestamp stored as a 32 bits integer containing time since 00:00:00 GMT, Jan. 1, 2005, in 2 seconds resolution. With a signed 32 bits integer, we can last for $2^{31} = 2147483648$, which is $2147483648/(3600/2 * 24 * 365) = 136.9$ years.

If we consider a base of moderate size – 100,000 users, 1,000,000 questions and 1,000,000 tags – we have 2,100,000 timestamp. They use about $2100000 * 4/1024/1024 = 8$ MB. However, there is a lot of redundancy in those timestamps, so by using a gzip-like compression algorithm we can have a high compression ration (10:1 is expected) so compressed timestamps would occupy about 800 kB.

When the client starts, it connects to the server and get all timestamps at once. It then check if its cache is up-to-date locally.

On the server side, the whole compressed timestamps are recreated regularly, e.g. each 2 minutes.

8.1 32 bits integer to/from string conversion

Function `be_of_int32` converts an `Int32` into its big endian binary representation.

33a ⟨*timestamp.ml* 32⟩+≡ ◁32 33b▷

```
let be_of_int32 n =
  let byte_mask = of_int 0xff in
  let char_of_int32 x = Char.chr (to_int x) in
  let d0 = char_of_int32 (logand n byte_mask) in
  let d1 = char_of_int32 (logand (shift_right_logical n 8) byte_mask) in
  let d2 = char_of_int32 (logand (shift_right_logical n 16) byte_mask) in
  let d3 = char_of_int32 (logand (shift_right_logical n 24) byte_mask) in
  let big_endian = String.make 4 d3 in
  big_endian.[1] <- d2;
  big_endian.[2] <- d1;
  big_endian.[3] <- d0;
  big_endian
```

Function `int32_of_be` converts an big endian binary representation of a 32 bits integer into an `Int32`.

33b ⟨*timestamp.ml* 32⟩+≡ ◁33a 33c▷

```
let int32_of_be be =
  if String.length be <> 4 then
   raise (Invalid_argument "int32_from_big_endian");
  let d3 = of_int (Char.code be.[3])
  and d2 = of_int (Char.code be.[2])
  and d1 = of_int (Char.code be.[1])
  and d0 = of_int (Char.code be.[0]) in
  (logor (shift_left d0 24)
     (logor (shift_left d1 16)
        (logor (shift_left d2 8) d3)))
```

8.2 Timestamps

Internally, a timestamp is a 32 bits integer. We also give function to convert from various data types.

33c ⟨*timestamp.ml* 32⟩+≡ ◁33b 34a▷

```
type t = Int32.t
```

[1] http://lists.nongnu.org/archive/html/demexp-dev/2005-06/msg00021.html

Function `current` returns the current timestamp, as an `Int32` storing number of 2s time intervals since 00:00:00 GMT, Jan. 1, 2005.

34a ⟨*timestamp.ml* 32⟩+≡ ◁33c 34b▷
```
let current () =
  let offset = (Unix.time () -. 35.0 *. (3600.0 *. 365.0 *. 24.0)) /. 2.0 in
  of_float offset
```

Function `to_string` converts a timestamp in its string counterpart, and `of_string` does the reverse conversion. Ditto for `of_int` and `of_float`.

34b ⟨*timestamp.ml* 32⟩+≡ ◁34a 34c▷
```
let to_string timestamp = Int32.to_string timestamp
let of_string str : t = Int32.of_string str

let of_int i : t = Int32.of_int i
let of_float f : t = Int32.of_float f
```

8.3 Timestamp blocks

We store a group of timestamps either as a `block` of timestamps or a `compressed_block`. In fact, a `block` is a `Bigarray.Array1` of kind `int32`.

34c ⟨*timestamp.ml* 32⟩+≡ ◁34b 34d▷
```
type block = (int32, Bigarray.int32_elt, Bigarray.c_layout) Bigarray.Array1.t
type compressed_block = string
```

Function `create` returns a new timestamp block of `size` timestamps.

34d ⟨*timestamp.ml* 32⟩+≡ ◁34c 34e▷
```
let create size =
  Bigarray.Array1.create Bigarray.int32 Bigarray.c_layout size
```

Function `length` returns the number of timestamps in a `block`.

34e ⟨*timestamp.ml* 32⟩+≡ ◁34d 34f▷
```
let length block = Bigarray.Array1.dim block
```

Function `compress` transforms a `block` into its gzipped form.

34f ⟨*timestamp.ml* 32⟩+≡ ◁34e 35a▷
```
let compress block =
  let num_timestamps = Bigarray.Array1.dim block in
  let buf = String.create (num_timestamps * 4) in
  (* convert all timestamps in their big endian binary string counterpart *)
  for i = 0 to num_timestamps - 1 do
    let be = be_of_int32 block.{i} in
    StringLabels.blit ~src:be ~src_pos:0 ~dst:buf ~dst_pos:(i * 4) ~len:4;
  done;
  (* compress them *)
  Gz.compress buf ~pos:0 ~len:(String.length buf)
```

Function `uncompress` transforms a `compressed_block` into a block of addressable timestamps.

35a ⟨*timestamp.ml* 32⟩+≡ ◁34f 35b▷

```
let uncompress compressed_block =
  (* uncompress compressed_block *)
  let buf =
    Gz.uncompress compressed_block
      ~pos:0 ~len:(String.length compressed_block) in
  (* transform it in an array of Int32 *)
  let buf_size = String.length buf in
  assert(buf_size mod 4 = 0);
  let num_timestamps = buf_size / 4 in
  let block = create num_timestamps in
  for i = 0 to num_timestamps - 1 do
    block.{i} <- int32_of_be (StringLabels.sub buf ~pos:(i * 4) ~len:4)
  done;
  block
```

8.4 Automatic tests

35b ⟨*timestamp.ml* 32⟩+≡ ◁35a

```
let _ =
  if Config.do_autotests then begin
    Printf.eprintf "  timestamp autotests...";
    assert(int32_of_be "\001\002\003\004" = of_string "0x01020304");
    assert(be_of_int32 (of_string "0x01020304") = "\001\002\003\004");
    assert(int32_of_be "\255\254\253\252" = of_string "0xfffefdfc");
    assert(be_of_int32 (of_string "0xfffefdfc") = "\255\254\253\252");

    let block = create 3 in
    block.{0} <- of_string "0x01020304";
    block.{1} <- of_string "0xfffefdfc";
    block.{2} <- of_string "0xa5a5a5a5";
    let compressed = compress block in
    let uncompressed = uncompress compressed in
    assert(uncompressed.{0} = of_string "0x01020304");
    assert(uncompressed.{1} = of_string "0xfffefdfc");
    assert(uncompressed.{2} = of_string "0xa5a5a5a5");
    assert(length uncompressed = 3);
    Printf.eprintf "done\n"
  end
```

Chapter 9

Miscellaneous (`Misc`)

Module `Misc` implements miscellaneous helper functions used by other modules.

36a ⟨*misc.ml* 36a⟩≡ 36b ▷
```
(* copyright 2004-2005 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Messages_aux
open Messages_clnt
open Rpc
```

9.1 Error handling

Exception `Display_error` is used to display an error message to the user.

36b ⟨*misc.ml* 36a⟩+≡ ◁ 36a 36c ▷
```
exception Display_error of string
```

9.2 Network routines

Helper routine `string_of_server_error` transforms a `Rpc.server_error error` into a printable string.

36c ⟨*misc.ml* 36a⟩+≡ ◁ 36b 37a ▷
```
let string_of_server_error error =
  match error with
    Unavailable_program -> "Unavailable program"
  | Unavailable_version _ -> "Unavailable version"
  | Unavailable_procedure -> "Unavailable procedure"
  | Garbage -> "Garbage"
  | System_err -> "System error"
  | Rpc_mismatch _ -> "RPC mismatch"
  | Auth_bad_cred -> "Auth_bad_cred"
  | Auth_rejected_cred -> "Auth_rejected_cred"
  | Auth_bad_verf -> "Auth_bad_verf"
  | Auth_rejected_verf -> "Auth_rejected_verf"
  | Auth_too_weak -> "Auth_too_weak"
  | Auth_invalid_resp -> "Auth_invalid_resp"
  | Auth_failed -> "Auth_failed"
```

Helper function `string_of_return_code` returns the RPC return `code` as a string.

37a ⟨*misc.ml* 36a⟩+≡ ◁36c 37b▷

```
let string_of_return_code code =
  match code with
  | v when v = rt_ok ->
      "Ok"
  | v when v = rt_generic_client_error ->
      "Generic client error"
  | v when v = rt_not_enough_rights ->
      "Not enough rights"
  | v when v = rt_already_exists ->
      "Already exists"
  | v when v = rt_not_found ->
      "Not found"
  | v when v = rt_request_too_much_ids ->
      "Request too much ids"
  | v when v = rt_generic_server_error ->
      "Generic server error"
  | v when v = rt_vote_choice_not_found ->
      "Vote choice not found"
  | v when v = rt_duplicate_vote_choice ->
      "Duplicate vote choice"
  | v when v = rt_anonymous_cannot_vote ->
      "Anonymous cannot vote"
  | v when v = rt_bad_format ->
      "An entry (question, response or login) is badly formatted"
  | v when v = rt_bad_login ->
      "Invalid login"
  | v -> let c = Rtypes.int_of_int4 v in
    Printf.sprintf "Unknown return code %d" c
```

9.3 String routines

We define a regular expression that matches spaces (space or tabulation).

37b ⟨*misc.ml* 36a⟩+≡ ◁37a 37c▷

```
let space_regexp = Str.regexp "[ \t]+"
```

Function `add_line_splits` transforms a single line string into a multi-lines string of maximum width `max_width`.

37c ⟨*misc.ml* 36a⟩+≡ ◁37b 38a▷

```
let add_line_splits max_width str =
  Format.pp_set_margin Format.str_formatter max_width;
  let words = Str.split space_regexp str in
  List.iter (fun word -> Format.fprintf Format.str_formatter "%s@ " word)
    words;
  let multiline = Format.flush_str_formatter () in
  if String.length multiline > 0 then
    String.sub multiline 0 (String.length multiline - 1) (* remove final
                                                            carriage return *)
  else
    multiline
```

9.4 Vote routines

Function `split_responses` takes as input a list of couples (question id, question description) as `q_responses` and a list of question ids in `my_vote`. It returns two lists of couples, the one containing the responses in `my_vote`, and the other containing the remaining responses.

As an example: `split_responses [1] [(1,"a"); (2,"b")] = ([(1, "a")], [(2, "b")])`

38a ⟨*misc.ml* 36a⟩+≡ ◁37c 38b▷

```
let split_responses my_vote q_responses =
  let rec split_aux my_vote my_vote_with_desc other_with_desc =
    match my_vote with
    | [] -> (my_vote_with_desc, other_with_desc)
    | vote :: tail ->
        let in_my_vote, new_other_with_desc =
          List.partition (fun (q_id, _) -> q_id = vote) other_with_desc in
        split_aux tail (my_vote_with_desc @ in_my_vote)
          new_other_with_desc in
  split_aux my_vote [] q_responses
```

9.5 Autotests

38b ⟨*misc.ml* 36a⟩+≡ ◁38a

```
let _ =
  if Config.do_autotests then begin
    Printf.eprintf "  misc autotests...";
    assert(add_line_splits 5 "a" = "a");
    assert(add_line_splits 5 "" = "");
    assert(add_line_splits 5 "1234 abcd" = "1234\nabcd");
    Printf.eprintf "done\n"
  end
```

Chapter 10

Cache of network values (Cache)

Module Cache temporarily stores server data within the client so as to reduce network load and improve response time. On a regular basis, the content of the cache is synchronized with the server.

The cache stores data structures as sent on the network. The cache content can be stored on disk and restored at client restart.

39a ⟨*cache.ml* 39a⟩≡ 39b ▷

```
(* copyright 2005-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Messages_aux
open Messages_clnt
```

10.1 Cache data structure

A cache contains information to contact the client (client and cookie) as well as the cache content itself, stored as three hash tables indexed by identifiers. Each table entry contains:

- a boolean indicating if the entry as been updated since the cache was last saved;
- the timestamp of the entry;
- the cached data.

39b ⟨*cache.ml* 39a⟩+≡ ◁39a 39c ▷

```
type t = {
    filename : string;
    question : (int, bool * Timestamp.t * question_t) Hashtbl.t;
    participant : (int, bool * Timestamp.t * info_on_participant_t) Hashtbl.t;
    tag : (int, bool * Timestamp.t * info_on_tag_t) Hashtbl.t;
  }
```

Function create_empty_cache returns a fresh empty cache.

39c ⟨*cache.ml* 39a⟩+≡ ◁39b 40a ▷

```
let create_empty_cache filename =
  { filename = filename;
    question = Hashtbl.create 3;
    participant = Hashtbl.create 3;
    tag = Hashtbl.create 3; }
```

In all code below, c is a cache data structure.

10.2 Saving and loading from disk

When reading the cache, we check that the version of the client and the OCaml compiler used to generate the serialized cache content on file is the same as the `client_cache_version` of this client, otherwise unserialization won't work. If the version is different, we silently drop the cache content on disk.

40a ⟨*cache.ml* 39a⟩+≡ ◁39c 40b▷

```
let client_cache_version = Sys.ocaml_version ^ "/" ^ Config.client_version
```

Function `load` reads the cache content from disk in file named `cache_filename`. When used, the returned cache object will query the server using `client` and `cookie` parameters.

40b ⟨*cache.ml* 39a⟩+≡ ◁40a 41a▷

```
let load cache_filename =
  let mark_all_entries_not_updated hash =
    Hashtbl.iter
      (fun id (_, ts, data) -> Hashtbl.replace hash id (false, ts, data))
      hash in
  let empty_cache = create_empty_cache cache_filename in
  if not (Sys.file_exists cache_filename) then
    empty_cache
  else
    try
      let ic = open_in cache_filename in
      let on_disk_version = input_line ic in
      let cache =
        if on_disk_version <> client_cache_version then
          empty_cache
        else (
          let question = input_value ic in
          mark_all_entries_not_updated question;
          let participant = input_value ic in
          mark_all_entries_not_updated participant;
          let tag = input_value ic in
          mark_all_entries_not_updated tag;
          { filename = cache_filename;
            question = question;
            participant = participant;
            tag = tag; }
         ) in
      close_in ic;
      cache
    with
    | Sys_error str ->
        raise (Misc.Display_error
                 (Printf.sprintf "ERROR: cannot read from file '%s': %s"
                    cache_filename str));
        empty_cache
    | End_of_file ->
        raise (Misc.Display_error
                 (Printf.sprintf
                    "ERROR: unexpected end of file '%s'" cache_filename));
        empty_cache
```

Function `save` saves the cache content of `cache` into its on disk file.

41a ⟨*cache.ml* 39a⟩+≡ ◁40b 41b▷

```
let save c =
  try
    let oc = open_out c.filename in
    output_string oc (client_cache_version ^ "\n");
    output_value oc c.question;
    output_value oc c.participant;
    output_value oc c.tag;
    close_out oc
  with
  | Sys_error str ->
      raise (Misc.Display_error
               (Printf.sprintf
                  "ERROR: cannot write to file '%s': %s" c.filename str))
```

10.3 Cache validation

Helper function `validate_entries` compares timestamps in `timestamp_table` to those in `hash` and remove corresponding entry in `hash` if it is older.

41b ⟨*cache.ml* 39a⟩+≡ ◁41a 41c▷

```
let validate_entries timestamp_table hash =
  for i = 0 to Timestamp.length timestamp_table - 1 do
    if Hashtbl.mem hash i then (
      let _, hash_ts, _ = Hashtbl.find hash i in
      if hash_ts < timestamp_table.{i} then
        Hashtbl.remove hash i
     )
  done
```

Function `validate` gets the latest timestamps from the server and check that each entry is the latest server version. If not the entry is removed from the cache.

41c ⟨*cache.ml* 39a⟩+≡ ◁41b 41d▷

```
let validate c client cookie =
  let timer = Perf.timer_start () in
  let ts = Demexp.V1.get_timestamps client cookie in
  validate_entries (Timestamp.uncompress ts.gt_question) c.question;
  Perf.timer_stop_and_record "Cache.validate" timer
```

10.4 Cache use

To use the cache, we define a function of the same name and type as the corresponding RPC it is caching. This function constructs the sets of missing information in the cache, get them from server and construct the expected response.

Function `create` returns the cache corresponding to file `cache_filename` stored on disk and validated with respect to server latest content.

41d ⟨*cache.ml* 39a⟩+≡ ◁41c 42a▷

```
let create cache_filename client cookie =
  let cache = load cache_filename in
  validate cache client cookie;
  cache
```

Function `build_toget_list` constructs the set of ranges to get from server because they are not in `hash` in order to fill requested range from `base` to `base + size - 1` (included). For example, if we request `base`=0 and `size`=10, and `hash` contains 3, 4 and 6, the function is going to return ranges [0-2], [5] and [7-9].

42a ⟨*cache.ml* 39a⟩+≡ ◁41d 42b▷

```
let build_toget_list hash base size =
  let rec build_aux base size remaining toget =
    if remaining <= 0 then
      if size > 0 then
        (base, size) :: toget
      else
        toget
    else
      if Hashtbl.mem hash (base + size) then
        if size > 0 then
          build_aux (base + size + 1) 0 (remaining - 1)
            ((base, size) :: toget)
        else
          build_aux (base + 1) 0 (remaining - 1) toget
      else
        build_aux base (size + 1) (remaining - 1) toget in
  build_aux base 0 size []
```

Function `generic_fill_hash` fills missing entries in `hash` between `base` and `base + size - 1` with data from server. For each missing range in `hash`, function `get_range` is called. The `hash` is then updated, using `get_timestamp` to extract timestamp from server's returned data.

42b ⟨*cache.ml* 39a⟩+≡ ◁42a 42c▷

```
let generic_fill_hash hash get_range get_id get_timestamp base size =
  (* construct what is needed to get from server *)
  let toget_list = build_toget_list hash base size in
  (* get it *)
  let missing = List.map get_range toget_list in
  (* update cache *)
  let put_range_in_hash range =
    for i = 0 to Array.length range - 1 do
      Hashtbl.replace
        hash (get_id range.(i)) (true, get_timestamp range.(i), range.(i))
    done in
  List.iter put_range_in_hash missing
```

Function `generic_build_array` builds an array of `size` elements, extracting needed information from `hash`, for `hash`'s elements with index between `base` and `base + size - 1`.

42c ⟨*cache.ml* 39a⟩+≡ ◁42b 43a▷

```
let generic_build_array hash base size =
  let l = ref [] in
  for i = base to base + size - 1 do
    if Hashtbl.mem hash i then
      let _, _, e = Hashtbl.find hash i in
      l := e :: !l
  done;
  Array.of_list !l
```

Function `question_info` implements RPC `question_info` with caching in `c`.

43a ⟨*cache.ml* 39a⟩+≡ ◁42c 43b▷

```
let question_info c
    (client : Rpc_client.t)
    (cookie, base, number_to_get : t_Demexp'V1'question_info'arg )
      : t_Demexp'V1'question_info'res =
  let get_question_range (base, size) =
    let timer = Perf.timer_start () in
    let ret = Demexp.V1.question_info client (cookie, base, size) in
    Perf.timer_stop_and_record "Demexp.V1.question_info" timer;
    if ret.question_info_rc <> rt_ok then
      raise (Misc.Display_error
               (Printf.sprintf
                  "unable to get info for questions %d to %d (%s)"
                  base (base + size - 1)
                  (Misc.string_of_return_code ret.question_info_rc)));
    ret.question_info in
  let get_question_timestamp q = q.q_timestamp
  and get_question_id q = q.q_id in
  generic_fill_hash c.question get_question_range
    get_question_id get_question_timestamp base number_to_get;
  { question_info_rc = rt_ok;
    question_info = generic_build_array c.question base number_to_get; }
```

Function `participant_info` implements RPC `participant_info` with caching in `c`.

43b ⟨*cache.ml* 39a⟩+≡ ◁43a 44a▷

```
let participant_info c
    (client : Rpc_client.t)
    (cookie, base, number_to_get : t_Demexp'V1'participant_info'arg )
      : t_Demexp'V1'participant_info'res =
  let get_participant_range (base, size) =
    let timer = Perf.timer_start () in
    let ret = Demexp.V1.participant_info client (cookie, base, size) in
    Perf.timer_stop_and_record "Demexp.V1.participant_info" timer;
    if ret.participant_info_rc <> rt_ok then
      raise (Misc.Display_error
               (Printf.sprintf
                  "unable to get info for participants %d to %d (%s)"
                  base (base + size - 1)
                  (Misc.string_of_return_code ret.participant_info_rc)));
    ret.participant_info in
  let get_participant_timestamp p = p.info_timestamp
  and get_participant_id p = p.info_id in
  generic_fill_hash c.participant get_participant_range get_participant_id
    get_participant_timestamp base number_to_get;
  { participant_info_rc = rt_ok;
    participant_info = generic_build_array c.participant base number_to_get; }
```

Function `tag_info` implements RPC `tag_info` with caching in `c`.

44a ⟨*cache.ml* 39a⟩+≡ ◁43b 44b▷

```
let tag_info c
    (client : Rpc_client.t)
    (cookie, base, number_to_get : t_Demexp'V1'tag_info'arg )
      : t_Demexp'V1'tag_info'res =
  let get_tag_range (base, size) =
    let timer = Perf.timer_start () in
    let ret = Demexp.V1.tag_info client (cookie, base, size) in
    Perf.timer_stop_and_record "Demexp.V1.tag_info" timer;
    if ret.tag_info_rc <> rt_ok then
      raise (Misc.Display_error
               (Printf.sprintf
                  "unable to get info for tags %d to %d (%s)"
                  base (base + size - 1)
                  (Misc.string_of_return_code ret.tag_info_rc)));
    ret.tag_info in
  let get_tag_timestamp t = t.a_tag_timestamp
  and get_tag_id t = t.a_tag_id in
  generic_fill_hash c.tag get_tag_range get_tag_id
    get_tag_timestamp base number_to_get;
  { tag_info_rc = rt_ok;
    tag_info = generic_build_array c.tag base number_to_get; }
```

In following functions, we tell on `what` we want to do a specific operation.

44b ⟨*cache.ml* 39a⟩+≡ ◁44a 44c▷

```
type what =
  | Question of int (* q_id *)
  | Tag of int (* tag_id *)
  | Participant of int (* participant_id *)
```

Function `invalidate` removes from the cache the question, tag or participant of a given identifier as given in the `what` argument.

Internal function `check_is_in_hash` is used to print a warning on stderr when we try to remove an object which is not in the cache, as this case should never happen.

44c ⟨*cache.ml* 39a⟩+≡ ◁44b 45a▷

```
let invalidate c what =
  let check_is_in_hash h kind id =
    if not (Hashtbl.mem h id) then
      Printf.eprintf "WARNING: invalidate %s_id:%d which is not in cache\n"
        kind id in
  match what with
  | Question q_id ->
      check_is_in_hash c.question "question" q_id;
      Hashtbl.remove c.question q_id
  | Tag tag_id ->
      check_is_in_hash c.tag "tag" tag_id;
      Hashtbl.remove c.tag tag_id
  | Participant p_id ->
      check_is_in_hash c.participant "participant" p_id;
      Hashtbl.remove c.participant p_id
```

Function `timestamp` returns the timestamp if the requested element. It raises `Not_found` if we don't have its timestamp in the cache.

45a ⟨*cache.ml* 39a⟩+≡ ◁44c 45b▷

```
let timestamp c what =
  match what with
  | Question q_id ->
      let _, timestamp, _ = Hashtbl.find c.question q_id in
      timestamp
  | Tag tag_id ->
      let _, timestamp, _ = Hashtbl.find c.tag tag_id in
      timestamp
  | Participant p_id ->
      let _, timestamp, _ = Hashtbl.find c.participant p_id in
      timestamp
```

10.5 Helper function to get tags and questions

Helper function `update_tags_hash` gets from the server the set of available tags. It updates the `h` hash table (id → tag label). In case of error, it display the error message in the status bar.

45b ⟨*cache.ml* 39a⟩+≡ ◁45a 46▷

```
let update_tags_hash h client cookie cache =
  Hashtbl.clear h;
  let ret = Demexp.V1.max_tag_id client cookie in
  if ret.max_tag_id_rc <> rt_ok then
    raise (Misc.Display_error
             (Printf.sprintf "unable to get max_tag_id (%s)"
                (Misc.string_of_return_code ret.max_tag_id_rc)));
  let max_number = Rtypes.int_of_uint4 max_number_ids in
  let rec get_some_tags base max_id =
    let number_to_get =
      if max_id - base + 1 <= max_number then max_id - base + 1
      else max_number in
    if base <= max_id then (
      let ret = tag_info cache client (cookie, base, number_to_get) in
      if ret.tag_info_rc <> rt_ok then
        raise (Misc.Display_error
                 (Printf.sprintf "unable to get info for tag %d to %d (%s)"
                    base (base + number_to_get - 1)
                    (Misc.string_of_return_code ret.tag_info_rc)));
      Array.iter (fun elt -> Hashtbl.add h elt.a_tag_id elt.a_tag_label)
        ret.tag_info;
      get_some_tags (base + number_to_get) max_id
     ) in
  get_some_tags 0 ret.max_tag_id
```

Helper function `tags_of_questions` returns all tag identifiers attached to each question between `base` and `base` + `number` - 1. The returned data type is an hash table indexed by question identifier and pointing to the list of tag identifiers attached to each question id.

46 ⟨*cache.ml* 39a⟩+≡ ◁45b 47▷

```
let tags_of_questions client cookie base number =
  let timer3 = Perf.timer_start () in
  let tags = Hashtbl.create number in
  let r =
    Demexp.V1.tag_set_of_question_group client
      (cookie, base, number) in
  if r.tag_set_group_rc <> rt_ok then
    raise (Misc.Display_error
             (Printf.sprintf
                "unable to get tags for questions %d to %d (%s)"
                base (base + number - 1)
                (Misc.string_of_return_code r.tag_set_group_rc)));
  Array.iter
    (fun s -> Hashtbl.add tags s.tag_set_qid (Array.to_list s.tag_set))
    r.tag_set_group;
  Perf.timer_stop_and_record "Cache.tags_of_question" timer3;
  tags
```

Function `update_questions_hash` updates the `h` hash table with the set of all public questions of the server, using `cache`. The hash table is indexed by question identifier and it contains question descriptor and its list of tag identifiers.

Using `get_some_questions`, information on questions is got from server in a group of up to `max_number_ids` questions.

In order to let the user know about this sometimes lengthy update, the `update_cb` callback function is regularly called, with the current `base` as argument.

47 ⟨*cache.ml* 39a⟩+≡ ◁46 48▷

```
let update_questions_hash ?(update_cb=fun _base -> ())
                                  h max_question_id client cookie cache =
    let timer = Perf.timer_start () in
    Hashtbl.clear h;
    let add_question tags q =
      if q.q_info_status = public then (
        let tags_of_question = Hashtbl.find tags q.q_id in
        Hashtbl.add h q.q_id (q.q_desc, tags_of_question)
       ) in
    let max_number = Rtypes.int_of_uint4 max_number_ids in
    let rec get_some_questions base max_id =
      if base <= max_id then (
        update_cb base;
        let number_to_get =
          if max_id - base + 1 <= max_number then max_id - base + 1
          else max_number in
        let ret =
          question_info cache client
            (cookie, base, number_to_get) in
        if ret.question_info_rc <> rt_ok then
          raise (Misc.Display_error
                   (Printf.sprintf
                      "unable to get info for questions %d to %d (%s)"
                      base (base + number_to_get - 1)
                      (Misc.string_of_return_code ret.question_info_rc)));
        let tags =
          tags_of_questions client cookie base number_to_get in
        Array.iter (add_question tags) ret.question_info;
        get_some_questions (base + number_to_get) max_id
       ) in
    get_some_questions 0 max_question_id;
    Perf.timer_stop_and_record "Cache.update_questions_hash" timer
```

10.6 Autotests

48 ⟨*cache.ml* 39a⟩+≡ ◁47

```
type e = { id : int; t : int }

let _ =
  if Config.do_autotests then begin
    Printf.eprintf "  cache autotests...";
    let h = Hashtbl.create 3 in
    assert(List.rev(build_toget_list h 0 10) = [(0, 10)]);
    Hashtbl.add h 3 3;
    assert(List.rev(build_toget_list h 0 10) = [(0, 3); (4, 6)]);
    Hashtbl.add h 4 4;
    Hashtbl.add h 6 6;
    assert(List.rev(build_toget_list h 0 10) = [(0, 3); (5, 1); (7, 3)]);
    Hashtbl.add h 0 0;
    assert(List.rev(build_toget_list h 0 10) = [(1, 2); (5, 1); (7, 3)]);

    (* test server access functions *)
    let get_range (base, size) =
      Array.init size (fun i -> {id = base + i; t = base + i + 10;} ) in
    let get_id e = e.id in
    let get_timestamp e = Int32.of_int (e.t) in
    let c = Hashtbl.create 10 in
    Hashtbl.add c 3 (false, Int32.of_int 3, {id=3; t=3});
    Hashtbl.add c 4 (false, Int32.of_int 4, {id=4; t=4});
    Hashtbl.add c 6 (false, Int32.of_int 6, {id=6; t=6});
    generic_fill_hash c get_range get_id get_timestamp 1 9;
    assert(Hashtbl.mem c 0 = false);
    assert(Hashtbl.find c 1 = (true, Int32.of_int 11, {id=1; t=11}));
    assert(Hashtbl.find c 2 = (true, Int32.of_int 12, {id=2; t=12}));
    assert(Hashtbl.find c 3 = (false, Int32.of_int 3, {id=3; t=3}));
    assert(Hashtbl.find c 4 = (false, Int32.of_int 4, {id=4; t=4}));
    assert(Hashtbl.find c 5 = (true, Int32.of_int 15, {id=5; t=15}));
    assert(Hashtbl.find c 6 = (false, Int32.of_int 6, {id=6; t=6}));
    assert(Hashtbl.find c 7 = (true, Int32.of_int 17, {id=7; t=17}));
    assert(Hashtbl.find c 8 = (true, Int32.of_int 18, {id=8; t=18}));
    assert(Hashtbl.find c 9 = (true, Int32.of_int 19, {id=9; t=19}));
    assert(Hashtbl.mem c 10 = false);
    assert(generic_build_array c 9 2 = Array.of_list [{id=9; t=19}]);
    Printf.eprintf "done\n"
  end
```

Part III

Web interface (`web/`)

Chapter 11

Translation support DemexpWebGettext

For translations to work, one needs to have the UTF-8 locales, e.g. `fr_FR.UTF-8`, `en_US.UTF-8` or `sv_SE.UTF-8`[1]. Use `locale -a` to check that you have them.

50 ⟨*demexpWebGettext.ml* 50⟩≡

```
module Gettext = Gettext.Program (struct
  let textdomain   = "demexp-web"
  let codeset      = Some "UTF-8"
  let dir          = match Sys.os_type with
                     | "Win32" | "Cygwin" -> Some "locale"
                     | _                  -> None
  let dependencies = Gettext.init
end)
(GettextStub.Native)
```

[1] http://article.gmane.org/gmane.politics.organizations.demexp.devel/1238

Chapter 12

Common code (Common)

51 ⟨*common.ml* 51⟩≡

```
module Option = struct
  (*include Option*)
  let map f = function
    | None -> None
    | Some x -> Some (f x)
end

(* put ugly things here to not polute the rest of the code too much *)
module Kludge = struct
  let display_tag tag_name =
    String.length tag_name < 9 || String.sub tag_name 0 9 <> "question "
end
```

Chapter 13

GlobalState

The module `GlobalState` contains stuff having with the global state of the application to do. Using global state might be a bad longtime solution, so lets make it short term :)

52 ⟨*globalState.ml* 52⟩≡

```
module Session = struct
  exception Invalid_id
  let max_id = ref 0
  let new_id () = incr max_id; string_of_int !max_id
  let version = 0
  type t =
    { login: string;
      pass: string;
      lang: string;
      vote_timestamps: (int, Timestamp.t) Hashtbl.t}
  let default () =
    { login = ""; pass = ""; lang = "en"; vote_timestamps = Hashtbl.create 10 }
  let hash = (Hashtbl.create 10 : (string, t) Hashtbl.t)
  let exists id = Hashtbl.mem hash id
  let get id = try Hashtbl.find hash id with Not_found -> raise Invalid_id
  let set id session = Hashtbl.replace hash id session
  let update id f = set id (f (get id))
  let invalidate id = Hashtbl.remove hash id
  let store file =
    (* This fails in case of no disk space or similar error *)
    (* But do we care? *)
    (* A currupt file might lead to segfault when loading it with stupid Marshal *)
    (* I was getting Invalid_id exceptions though, when i run out of disc space *)
    let f id session l = (id,session)::l in
    let ch = open_out_bin file in
    output_byte ch version;
    Marshal.to_channel ch (!max_id, Hashtbl.fold f hash []) [];
    close_out ch
  let load file =
    Hashtbl.clear hash;
    try
      let ch = open_in_bin file in
      assert (input_byte ch = version);
      let (id, l) = Marshal.from_channel ch in
      let f (id,session) = set id session in
      max_id := id;
      List.iter f l
    with _ -> () (* it is not our fault we do not care :p *)
end
```

```
module Conf = struct
  open Config_file
  let group = new group
  let str name default help = new string_cp ~group [name] default help
  let bool name default help = new bool_cp ~group [name] default help
  let public_login = str "public login" "demo" ""
  let public_password = str "public password" "demo" ""
  let has_admin_account = bool "has admin account" true ""
  let admin_login = str "admin login" "root" ""
  let admin_password = str "admin password" "demexp" ""
  let default_new_user_groups =
    new list_cp string_wrappers ~group ["default_new_user_groups"]
                ["classifier"] ""
end

module Var = struct
  open Wd_types
  let dlg = ref (None : Wd_dialog.dialog option)
  let d () = match !dlg with Some d -> d | None -> assert false
  type 'a var = { get: unit -> 'a ; set: 'a -> unit; name: string }
  let of_ref r = { get = (fun () -> !r); set = (fun x -> r := x); name = "" }
  let objify var =
    object method get = var.get () method set = var.set method name = var.name end
  let bug n f x = try f x with e -> prerr_endline ("Hint: "^n); raise e
  let mkvar f1 f2 name = objify
    { get = (fun () -> bug name (f1 (d())) name);
      set = (fun x -> bug name ((d())#set_variable name) (f2 x));
      name = name }
  let string_var = mkvar (fun d -> d#string_variable)
                         (fun x -> String_value x)
  let bool_var = mkvar (fun d n -> int_of_string (d#string_variable n) <> 0)
                       (function true ->  String_value "1" | false -> String_value "0")
  let int_var = mkvar (fun d n -> int_of_string (d#string_variable n))
                      (fun x -> String_value (string_of_int x))
  let dyn_enum_var = mkvar (fun d -> d#dyn_enum_variable)
                           (fun x -> Dyn_enum_value x)
  let assert_string = function String_value x -> x | _ -> assert false
  let string_alist_var =
    mkvar (fun d n -> List.map (fun (a,b) -> a, assert_string b) (d#alist_variable n))
          (fun x -> Alist_value (List.map (fun (a,b) -> a, String_value b) x))
  (* variables: *)
  let session_id = let r = ref "" in objify (of_ref r)
  let logged_in = let r = ref false in objify (of_ref r)
  let login = string_var "session.login"
  let password = string_var "session.password"
  let password_confirm = string_var "session.password-confirm"
  let cur_page = string_var "cur-page"
  let error_message = string_var "error-message"
  let lang = string_var "lang"
  let head = string_var "head"
  let body = string_var "body"
  module Bool = struct
    let nav = bool_var "bool.nav"
    let add_tag = bool_var "bool.new-tag"
    let add_ans = bool_var "bool.new-ans"
    let add_question = bool_var "bool.new-question"
    let tag_question = bool_var "bool.tag-question"
```

```
    let reg_success = bool_var "bool.reg-success"
  end
  module T = struct (* tag *)
    let list = dyn_enum_var "tag.list"
    let selected = string_var "tag.selected"
    let previous = string_var "tag.previous"
    let for_addition = string_var "tag.new"
  end
  module Q = struct (* question *)
    let list = dyn_enum_var "question.list"
    let selected = string_var "question.selected"
    let previous = string_var "question.previous"
    let title = string_var "question.title"
    let responses = dyn_enum_var "question.responses"
    let limit_date = string_var "question.limit-date"
    let tags = dyn_enum_var "question.tags"
    let winning_responses = string_var "question.winning-responses"
    let number_of_votes = int_var "question.number-of-votes"
  end
  module V = struct (* vote *)
    let list = dyn_enum_var "vote.list"
    let selected = string_var "vote.selected"
  end
  module Add = struct (* values for addition *)
    let tag = T.for_addition
    let q_desc = string_var "question.new"
    let ans_desc = string_var "question.new-ans"
    let ans_url = string_var "question.new-ans-url"
    let ans_desc_list = string_alist_var "question.new-ans-list"
    let ans_url_list = string_alist_var "question.new-ans-url-list"
  end
  module Err = struct (* error variables *)
    let general = error_message
    let q_desc = string_var "question.error-new"
    let ans_desc_list = string_alist_var "question.new-ans-error"
    let ans_url_list = string_alist_var "question.new-ans-url-error"
  end
end
```

Chapter 14

Connection to server (`ServerConnection`)

Module `ServerConnection` provides code to connect to the server.

55a ⟨*serverConnection.ml* 55a⟩≡ 55b ▷
```
(* copyright 2005 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Messages_aux
open Messages_clnt
open GlobalState
```

The server to connect to is decided by one of

1. the file `/etc/demexp_server`;
2. the file `./demexp_server`;
3. the hardcoded value of `Config.default_server_name`.

TODO: use `Config_file` library instead.

55b ⟨*serverConnection.ml* 55a⟩+≡ ◁55a 55c ▷
```
let web_default_server =
  let great = ref "" in
  let use file =
    if Sys.file_exists file
    then (let ch = open_in file in great := input_line ch; close_in ch; true)
    else false
  in
  if use "/etc/demexp_server" then !great else
  if use "demexp_server" then !great
  else Config.default_server_name

let web_default_port = Config.default_server_port
```

We define some exceptions in case login goes wrong.

55c ⟨*serverConnection.ml* 55a⟩+≡ ◁55b 56 ▷
```
exception Login_error of string * Rpc_client.t * int
                         (* msg *    client    * cookie *)

exception Protocol_warning of string * Rpc_client.t * int
                              (* msg *    client    * cookie *)
```

We use an on-disk cache to avoid spurious network accesses. The cache file is named `cache_filename`.

56 ⟨*serverConnection.ml* 55a⟩+≡ ◁55c 57▷

```
let cache_filename =
  "/tmp/demexp-web-cache-" ^ web_default_server ^ ":"
  ^ (string_of_int web_default_port)
```

Bla bla, bla?

57 ⟨*serverConnection.ml* 55a⟩+≡ ◁56 59▷

```
(* TODO *)
type server =
  < question_info : int -> int -> question_t array
  >

let chk_rc rc errfunc =
  if rc <> Messages_aux.rt_ok then (
    let msg = Misc.string_of_return_code rc in
    errfunc msg
  )

(* maybe we can do something partly automated with the help of rpc definition instead? *)
let server_of_ccc client cookie cache errfunc =
  let chk rc = chk_rc rc errfunc in
  let tags = Hashtbl.create 3(*??*) in
  let questions = Hashtbl.create 50 in

  let max_question_id () =
    let r = Demexp.V1.max_question_id client cookie in
    chk r.max_question_id_rc;
    r.max_question_id
  in

  let update_tags () = Cache.update_tags_hash tags client cookie cache in
  let update_questions () =
    Cache.update_questions_hash questions (max_question_id ()) client cookie cache
  in
  object (self)
    (* Is this safe? *) (* well... it is ugly *)
    method client = client method cookie = cookie method cache = cache

    (*DEPR*)method question_info id quantity : question_t array =
      let r = Cache.question_info cache client (cookie, id, quantity) in
      chk r.question_info_rc;
      r.question_info

    method tags_of_question id : int array =
      let () = update_questions () in
      snd (Hashtbl.find id questions)

    (*DEPR*)method get_question_tags id : int array =
      let r = Demexp.V1.get_question_tags client (cookie, id) in
      r

    method all_tags =
      let () = update_tags () in
      let f id label tag_list =
        if not (Norm.is_question_specific_tag label)
        then (id,label)::tag_list
        else tag_list
      in Hashtbl.fold f tags []

    method name_of_tag id : string =
      let () = update_tags () in
      Hashtbl.find tags id
```

```
  method all_questions =
    let () =
      Cache.update_questions_hash questions self#max_question_id client cookie cache
    in
    let f id x l = (id,x)::l in
    Hashtbl.fold f questions []

  (*DEPR*)method tag_info id quantity : info_on_tag_t array =
    let r = Demexp.V1.tag_info client (cookie, id, quantity) in
    chk r.tag_info_rc;
    r.tag_info

  method get_vote id login : int array =
    let r = Demexp.V1.get_vote client (cookie, id, login) in
    chk r.get_vote_rc;
    r.get_vote
  method vote q_id ans_ids =
    let r = Demexp.V1.vote client (cookie, q_id, ans_ids) in
    chk r;
    Cache.invalidate cache (Cache.Question q_id)

  method tag_question q_id tag_id =
    let r = Demexp.V1.tag_question client (cookie, q_id, tag_id) in
    chk r
  method untag_question q_id tag_id =
    let r = Demexp.V1.untag_question client (cookie, q_id, tag_id) in
    chk r
  method create_tag label : int =
    let r = Demexp.V1.create_tag client (cookie, label) in
    chk r.create_tag_rc;
    r.create_tag_id
  method add_response q_id desc link =
    let r = Demexp.V1.add_response client (cookie, q_id, desc, link) in
    chk r;
    Cache.invalidate cache (Cache.Question q_id)
  method new_question question : int =
    let r = Demexp.V1.new_question client (cookie, question) in
    chk r.question_id_return_code;
    r.question_id_id
  method set_question_status q_id status =
    let status = Rtypes.int4_of_int status in
    let r = Demexp.V1.set_question_status client (cookie, q_id, status) in
    chk r
  method max_question_id = max_question_id ()
  method add_participant login pass groups =
    let r = Demexp.V1.add_participant client (cookie, login, pass, groups) in
    chk r.add_participant_rc;
    r
end
```

Function `do_in_server` runs a function inside the scope of a server-connection. The function is expected to take a server object as argument. The argument `on_error` is passed on to the creation of the server object. If login fails `Login_error` is raised. If the current protocol version does not match the server one, `Protocol_warning` is raised.

59 ⟨*serverConnection.ml* 55a⟩+≡ ◁57

```
let default_error_func err = raise (Misc.Display_error err)

(* Should we call goodbye and shut_down in case of warning/error? *)
let do_in_server
    ?(url=web_default_server) ?(port=web_default_port) ?(on_error=default_error_func)
    ?(login=Var.login#get) ?(pass=Var.password#get) f =
  (*let (login,pass) =
    if login = "" then (Conf.public_login#get, Conf.public_password#get)
                  else (login,pass)
  in*)
  let protocol = Rtypes.int_of_uint4 protocol_version in
  let client = Demexp.V1.create_client (Rpc_client.Inet (url,port)) Rpc.Tcp in
  let r = Demexp.V1.login client (protocol, login, pass) in
  if login <> "" then (
    chk_rc r.login_return_code (fun x ->
      raise (Login_error ("Unable to login: "^x, client, r.login_cookie)))
  );
  let () = if r.server_protocol_version <> protocol then
    let msg = Printf.sprintf
      "Unmatched protocol version (server:%d <> client:%d), please inform the administrator of
      r.server_protocol_version protocol in
    raise (Protocol_warning (msg, client, r.login_cookie)) in
  let cookie = r.login_cookie in
  let cache = Cache.create cache_filename client cookie in
  try
    let x = f (server_of_ccc client cookie cache on_error) in
    Cache.save cache;
    Demexp.V1.goodbye client cookie;
    Rpc_client.shut_down client;
    x
  with e ->
    Cache.save cache;
    Demexp.V1.goodbye client cookie;
    Rpc_client.shut_down client;
    raise e
```

Chapter 15

Factory

60 ⟨*factory.ml* 60⟩≡

```
open GlobalState
open Common
open DemexpWebGettext.Gettext

let escape s = s (* TODO *)

module Xml = struct
  let mk_elem ?(attr=[]) ~name ?(not_empty=false) body =
    let f (a,b) = " " ^ a ^ "=\"" ^ b ^ "\"" in
    let attr = String.concat "" (List.map f attr) in
    "<" ^ name ^ attr ^
    if body = "" && not not_empty then "/>" else (">" ^ body ^ "</" ^ name ^ ">")
end
open Xml

module Html = struct
  let maybe name = function Some x -> [name,x] | None -> []

  let br = mk_elem ~name:"br" ""

  let div ?style body =
    let attr = maybe "style" style in
    mk_elem ~name:"div" ~attr body

  let p ?style body =
    let attr = maybe "style" style in
    mk_elem ~name:"p" ~attr body

  let span ?style body =
    let attr = maybe "style" style in
    mk_elem ~name:"span" ~attr body

  let table ?cellpadding ?style body =
    let attr = maybe "cellpadding" cellpadding @ maybe "style" style in
    mk_elem ~name:"table" ~attr body

  let tr = mk_elem ~name:"tr"
  let td = mk_elem ~name:"td"

  let a ?(href="") ?onclick body =
    let may a = function Some x -> [a,x] | None -> [] in
```

```
    let attr = ("href",href) :: may "onclick" onclick in
    mk_elem ~name:"a" ~attr body

  let form ?(meth=‘POST) ~name ~action body =
    let meth = match meth with ‘POST -> "post" | ‘GET -> "get" in
    let attr = ["method",meth; "name",name; "action",action] in
    mk_elem ~name:"form" ~attr body

  let input ~t ?name ~value ?onclick ?style ?size () =
    let attr =
      ("type",t)::("value",value):: maybe "name" name @ maybe "style" style @
        maybe "onclick" onclick @ maybe "size" (Option.map string_of_int size)
    in
    mk_elem ~name:"input" ~attr ""

  let select ~name ?size ?onclick ?onclickf ?selected options =
    let mk_opt (a,b) =
      let sel = if Some a = selected then ["selected","selected"] else [] in
      let onc = match onclickf with None -> [] | Some f -> ["onclick",f (a,b)] in
      mk_elem ~name:"option" ~attr:(("value",a)::onc@sel) "" ^ b
    in
    let attr =
      let f name = function Some x -> [name,x] | None -> [] in
      ("name",name) :: f "size" size @ f "onclick" onclick
    in
    mk_elem ~name:"select" ~attr (String.concat "" (List.map mk_opt options))

  let button ?onclick body =
    let attr =
      let f name = function Some x -> [name,x] | None -> [] in
      f "onclick" onclick
    in
    mk_elem ~name:"button" ~attr body
end

module Ui = struct
  open Wd_types

  let form ~dlg body =
    let () = dlg#session#commit_changes () in
    let session = dlg#session#serialize in
    let uiobject_visible_pages = "main" in (* TODO *)
    let attr = ["method","post"; "name","uiform"; "action","./demexpweb.cgi";
                "enctype","multipart/form-data"; "accept-charset","UTF-8";
                "action-suffix",""] in
    mk_elem ~name:"form" ~attr (
      mk_elem ~name:"script" ~attr:["language","javascript"]
      "function uiform_click(id) {
         if (document.uiform.onsubmit == null || document.uiform.onsubmit() != false) {

            document.uiform.elements[id].value=’1’;
            uiform_submit();
         }
       }
       function uiform_submit() {
         document.uiform.submit();
       }" ^
      body ^
```

```
      Html.input ~t:"hidden" ~name:"uiobject_session" ~value:session () ^
      Html.input ~t:"hidden" ~name:"uiobject_extra_args" ~value:"" () ^
      Html.input ~t:"hidden" ~name:"uiobject_visible_pages"
        ~value:uiobject_visible_pages () ^
      Html.input ~t:"hidden" ~name:"uiobject_popup_index" ~value:"" ()
    )

  (* TODO: split in indexed version and non-indexed version *)
  let text ~dlg ~var ?index ?size () =
    let indX () = match index with Some x -> x | None -> assert false in
    let rec f indexf = function
      | String_value s -> s
      | Dyn_enum_value l -> List.assoc (indexf ()) l
      | Alist_value l -> f (fun _ -> assert false) (List.assoc (indexf ()) l)
      | Dialog_value _ | Enum_value _ -> failwith "not implemented, sorry"
    in
    let index = match index with Some x -> x | None -> "" in
    let value = f indX (dlg#variable var) in
    let ia = dlg#interactors.Wd_types.ui_vars in
    let id = Wd_interactor.add ia var index None () in
    Html.input ~t:"text" ~name:("var_" ^ id) ~value ?size ()

  let password ~dlg ~var ?size () =
    let value = dlg#string_variable var in
    let ia = dlg#interactors.Wd_types.ui_vars in
    let id = Wd_interactor.add ia var "" None () in
    Html.input ~t:"password" ~name:("var_" ^ id) ~value ?size ()

  let button ~dlg ~name ?style ~label () =
    let ia = dlg#interactors.Wd_types.ui_buttons in
    let id = Wd_interactor.add ia name "" None None in
    Html.input ~t:"submit" ~name:("button_" ^ id) ?style ~value:label ()

  let x_button ~dlg ~name ~index ~label =
    let ia = dlg#interactors.Wd_types.ui_indexed_buttons in
    let id = Wd_interactor.add ia name index None None in
    Html.input ~t:"submit" ~name:("xbutton_" ^ id) ~value:label ()

  let a ~dlg ~name ?onclick body =
    let ia = dlg#interactors.Wd_types.ui_anchors in
    let id = Wd_interactor.add ia name "" None None in
    Html.input ~t:"hidden" ~name:("anchor_" ^ id) ~value:"0" () ^
    Html.a ~href:("javascript:uiform_click('anchor_"^id^"')") ?onclick body

  let x_a ~dlg ~name ~index ?onclick body =
    let ia = dlg#interactors.Wd_types.ui_indexed_anchors in
    let id = Wd_interactor.add ia name index None None in
    Html.input ~t:"hidden" ~name:("xanchor_" ^ id) ~value:"0" () ^
    Html.a ~href:("javascript:uiform_click('xanchor_"^id^"')") ?onclick body

  let select ~dlg ~var ~base ?size ?onclick ?onclickf () =
    let selected = dlg#string_variable var in
    let options = dlg#dyn_enum_variable base in
    let ia = dlg#interactors.Wd_types.ui_vars in
    let id = Wd_interactor.add ia var "" None () in
    Html.select ~name:("var_" ^ id) ?size ?onclick ?onclickf ~selected options
end
```

```
let unique =
  let id = ref 0 in
  fun () -> incr id; string_of_int !id

module Page = struct
  type t = { e_handlers: (Wd_types.event -> bool) list }
  let empty = { e_handlers = [] }

  let string_of_event = function
    | Wd_types.Button str -> "button: " ^ str
    | Wd_types.No_event -> "No event"
    | _ -> failwith "not implemented"

  let add page eventf =
    { e_handlers = eventf :: page.e_handlers }

  let handle ~page event =
    let rec f = function [] -> raise Not_found
                   | x::tl -> if not (x event) then f tl in
    try f page.e_handlers
    with Not_found ->
      Var.Err.general#set ("unrecognized event: " ^ string_of_event event)

  type 'a button =
    < label: string; xml: 'a -> string >
  type button_style = Button | Link | Button_style of string

  let button ~label ~pageref ?(style=Button) f =
    let id = unique () in
    pageref := add !pageref (fun e -> e = Wd_types.Button id && (f (); true));
    let xml () =
      match style with
      | Button -> Ui.button ~dlg:(Var.d()) ~name:id ~label ()
      | Button_style s -> Ui.button ~dlg:(Var.d()) ~name:id ~style:s ~label ()
      | Link -> Ui.a ~dlg:(Var.d()) ~name:id label
    in
    object method label = label method xml = xml end

  let x_button ~label ~pageref ?(style=Button) f =
    let id = unique () in
    let f = function
      | Wd_types.Indexed_button (name, index) when name = id -> f index; true
      | _ -> false
    in
    pageref := add !pageref f;
    let xml index =
      match style with
      | Button -> Ui.x_button ~dlg:(Var.d()) ~name:id ~label ~index
      | Button_style _ -> Ui.x_button ~dlg:(Var.d()) ~name:id ~label ~index
      | Link -> Ui.x_a ~dlg:(Var.d()) ~name:id ~index label
    in
    object method label = label method xml = xml end

  let select ~list ~var ?size ?(click=`None) ?clickf () =
    let size = match size with Some s -> Some (string_of_int s) | None -> None in
    let onclick = match click with
      | `Submit -> Some "document.uiform.submit()"
      | `Script s -> Some s
```

```
      | `None -> None
    in
    Ui.select ~dlg:(Var.d()) ~var:var#name ~base:list#name ?size ?onclick
              ?onclickf:clickf ()

  let text ~var ?size ?index () =
    Ui.text ~dlg:(Var.d()) ~var:var#name ?size ?index ()

  let password ~var ?size () =
    Ui.password ~dlg:(Var.d()) ~var:var#name ?size ()

  let std_script = "
    function setlocation(uri) {
      window.location=uri
    }
  "

  let top_bar = ref (fun () -> "")

  (* TODO: we should set some no cache directives in header *)
  let std ~title ?script body =
    let pagename = Var.cur_page#get in
    mk_elem "html" (
      mk_elem "head" (
        mk_elem ~name:"title" title ^
        mk_elem ~name:"style" ~attr:["type","text/css"]
        "a:visited { color: blue }" ^
        mk_elem ~name:"script" ~attr:["language","javascript"] ~not_empty:true
          std_script ^ (match script with None -> "" | Some s -> s)
      ) ^
      mk_elem "body" (
        let errmsg = Var.error_message#get in
        !top_bar () ^
        mk_elem ~name:"hr" "" ^
        mk_elem ~name:"h1" title ^
        (if errmsg <> ""
        then mk_elem ~name:"font" ~attr:["color","red"] ("Error: " ^ errmsg)
        else "") ^
        Ui.form ~dlg:(Var.d()) (
          body ^
          Html.br ^
          Html.br ^
          mk_elem ~name:"hr" "" ^
          mk_elem ~name:"p" (
            let lang xx =
              Html.a ~href:("?page=" ^ pagename ^ "&lang=" ^ xx) xx in
            (s_"Language: ") ^
            lang "fr" ^ " " ^ lang "en" ^ " " ^ lang "sv"
          )
        )
      )
    )
end
```

Chapter 16

`variables` dialog

The `variables` dialog contains the variables used by all other dialogs. It allows to have persistency of the state between the dialogs.

65 ⟨*variables.ui* 65⟩≡ 66 ▷

```
<!-- copyright 2005-2006 David MENTRE -->
<!-- this software is under GNU GPL. See COPYING.GPL file for details -->
```

66 ⟨*variables.ui* 65⟩+≡ ◁65

```
<ui:dialog name="session" start-page="empty">
  <ui:variable name="login"/>
  <ui:variable name="password"/>
  <ui:variable name="password-confirm"/>
  <ui:page name="empty"/>
</ui:dialog>

<ui:dialog name="tag" start-page="empty">
  <ui:variable name="list" type="dynamic-enumerator"/>
  <ui:variable name="selected"/>
  <ui:variable name="previous"/>
  <ui:variable name="new"/>
  <ui:page name="empty"/>
</ui:dialog>

<ui:dialog name="question" start-page="empty">
  <ui:variable name="list" type="dynamic-enumerator"/>
  <ui:variable name="selected"/>
  <ui:variable name="previous"/>
  <ui:variable name="title"/>
  <ui:variable name="responses" type="dynamic-enumerator"/>
  <ui:variable name="limit-date"/>
  <ui:variable name="tags" type="dynamic-enumerator"/>
  <ui:variable name="winning-responses"/>
  <ui:variable name="number-of-votes"/>
  <ui:variable name="new-ans"/>
  <ui:variable name="new-ans-url"/>
  <ui:variable name="new"/>
  <ui:variable name="error-new"/>
  <ui:variable name="new-ans-list" associative="yes"/>
  <ui:variable name="new-ans-url-list" associative="yes"/>
  <ui:variable name="new-ans-error" associative="yes"/>
  <ui:variable name="new-ans-url-error" associative="yes"/>
  <ui:variable name="new-tag"/>
  <ui:page name="empty"/>
</ui:dialog>

<ui:dialog name="vote" start-page="empty">
  <ui:variable name="list" type="dynamic-enumerator"/>
  <ui:variable name="selected"/>
  <ui:page name="empty"/>
</ui:dialog>

<ui:dialog name="bool" start-page="empty">
  <ui:variable name="nav">&true;</ui:variable>
  <ui:variable name="new-tag">&false;</ui:variable>
  <ui:variable name="new-ans">&false;</ui:variable>
  <ui:variable name="new-question">&false;</ui:variable>
  <ui:variable name="tag-question">&false;</ui:variable>
  <ui:variable name="reg-success">&false;</ui:variable>
  <ui:variable name="debug">&false;</ui:variable>
  <ui:page name="empty"/>
</ui:dialog>
```

Chapter 17

Pages

67a ⟨*pages.ml* 67a⟩≡ 67b ▷
```
open Messages_aux
open Messages_clnt
open GlobalState
open DemexpWebGettext.Gettext
open Factory
open Common

exception Display_error = Misc.Display_error

let raise_if b exc = if b then raise exc
```

`ppe` wraps an expression in a `try ... with` statement and prepends every `Display_error` with a message. It uses the `Obj` module, so it requiers some explanation: If `x` is not a closure we return `x`. If `x` is a closure we assert that `x` has type `('a -> 'b)`, and that `(ppe' (wrap x y))` has type `'b` for all `y`. The first follows from `x` being a closure, and the second follows from the type of `wrap`.

67b ⟨*pages.ml* 67a⟩+≡ ◁ 67a 68 ▷
```
let ppe msg = (* prepend error message *)
  let wrap a b : ('a -> 'b) -> 'b =
    try a b with Display_error s -> raise (Display_error (msg^s))
  in
  let rec ppe' x =
    if Obj.tag (Obj.repr x) <> Obj.closure_tag
    then x
    (* TODO: check why (wrap (Obj.magic x) y) does not pass the type system *)
    else Obj.magic (fun y -> ppe' (Obj.magic wrap x y))
  in fun x -> (ppe' : 'a -> 'a) x
```

```
let redirect_to_question id =
  let module Local = struct
    open Wd_types
    let rh = (Var.d())#environment.response_header
    let () = rh.rh_status <- `Temporary_redirect;
             rh.rh_fields <- ["Location",["?question_id="^string_of_int id]]
  end in
  ()

let do_in_server = ServerConnection.do_in_server
let error x = Var.error_message#set x

type page = Login | Register | Browse | Personal | Logout_confirm
let page_mapping =
  [ Login, "login";
    Register, "register";
    Browse, "browse";
    Personal, "personal";
    Logout_confirm, "logout confirm"]

exception InvalidPageName of string

let page_of_string name =
  try fst (List.find (fun (_,str) -> str = name) page_mapping)
  with Not_found -> raise (InvalidPageName name)

let string_of_page page =
  snd (List.find (fun (p,_) -> p = page) page_mapping)

let rec cur_page ?default () =
  try page_of_string Var.cur_page#get
  with InvalidPageName s ->
    match default with
    | Some p -> change_page p; p
    | None -> raise (InvalidPageName s)

and change_page new_page =
  if try new_page <> cur_page () with InvalidPageName _ -> true
  then Var.Err.general#set "";
  Var.cur_page#set (string_of_page new_page)

let goto_page ?(output=`Script) page =
  match output with
  | `Script -> "setlocation(&quot;?page=" ^ string_of_page page ^ "&quot;)"
  | `URI -> "?page=" ^ string_of_page page

let goto ?(output=`Script) ?(tag=Var.T.selected#get) ?question
         ?(nav=Var.Bool.nav#get) () =
  let question = match question with
    | None -> Var.Q.selected#get
    | Some id -> string_of_int id
  in
  let nav = if nav then "1" else "0" in
  let compose l =
    let f (a, b) = a ^ "=" ^ b in
    "?" ^ String.concat "&" (List.map f l)
  in
```

```
  let may name value = if value <> "" then [name,value] else [] in
  let uri = compose (may "tag_id" tag @
                     may "question_id" question @
                     may "navigation" nav) in
  match output with
  | ‘Script -> "setlocation(&amp;quot;" ^ uri ^ "&amp;quot;)"
  | ‘URI -> uri

let () =
  Factory.Page.top_bar := (fun () ->
    let pagelink p linkname =
      let onclick = goto_page ~output:‘Script p in
      Html.button ~onclick linkname
      (*Html.form ~meth:‘POST ~name:"stupidform" ~action:link (
        Html.input ~t:"submit" ~value:linkname ()
      )*)
    in
    if Var.logged_in#get && cur_page () <> Logout_confirm then
      s_"Logged in as: " ^ Xml.mk_elem ~name:"b" Var.login#get ^ " " ^
      pagelink Personal (
        "Personal Page"
      ) ^
      pagelink Browse (
        "Browse Page"
      ) ^
      pagelink Logout_confirm (
        "Logout"
      )
    else
      pagelink Browse (
        "Browse Page"
      ) ^
      pagelink Login (
        "Login Page"
      ) ^
      pagelink Register (
        "Registration Page"
      )
  )
```

Helper function `fill_tag_list` get tags from the server and puts them in Var.T.list
todo: UTF-8 encoded strings shold be sorted according to current locale.(maybe?) TO DO

70 ⟨*pages.ml* 67a⟩+≡ ◁68 71▷

```
let fill_tag_list srv =
  (*let tags = Hashtbl.create 3 in
  Cache.update_tags_hash tags srv#client srv#cookie srv#cache;
  let aggregate_non_question_tags id label tag_list =
    if not (Norm.is_question_specific_tag label) then
      (id, label) :: tag_list
    else tag_list in
  let tags_dyn_val =
    let raw_tags = Hashtbl.fold aggregate_non_question_tags tags [] in
    let sorted = List.sort (fun (_, l1) (_, l2) -> compare l1 l2) raw_tags in
    List.map (fun (id, label) ->
      (string_of_int id, Printf.sprintf "%s" label)) sorted in*)
  let sorted = List.sort (fun a b -> compare (snd a) (snd b)) srv#all_tags in
  let stupid =
    List.map (fun (id,label) -> string_of_int id, label) sorted
  in
  Var.T.list#set stupid
```

```
let fill_question_list srv =
  let max_question_id =
    ppe (s_"Unable to get max_question_id :") (fun () -> srv#max_question_id) () in
  let questions = Hashtbl.create 3 in
  Cache.update_questions_hash
    questions max_question_id srv#client srv#cookie srv#cache;
  try
    let chosen_tag = int_of_string (Var.T.selected#get) in
    let select_question id (desc, tags) selection =
      if List.exists (fun tag_id -> tag_id = chosen_tag) tags then
        (string_of_int id, desc) :: selection
      else
        selection in
    Var.Q.list#set (Hashtbl.fold select_question questions [])
  with Failure "int_of_string" -> ()

let fill_question_details q_id srv =
  let q_info =
    let x = ppe (s_"Unable to load question information: ") srv#question_info q_id 1 in
    assert (Array.length x = 1); x.(0)
  in
  try
    Var.Q.title#set q_info.q_desc;
    (* limit date *)
    Var.Q.limit_date#set
      (match q_info.q_info_limit_date with
       | x when x = Int64.zero -> s_"no limit date"
       | x -> (* transform limit date in local time *)
           let offset = Int64.to_float x in
           Time.time_as_localtime_iso_string offset);
    (* tags *)
    let tag_set = srv#get_question_tags q_id in
    let tag_pair id =
      let arr = srv#tag_info id 1 in
      (string_of_int id, arr.(0).a_tag_label)
    in
    let tag_pairs = List.map tag_pair (Array.to_list tag_set) in
    let f (_, s) = Kludge.display_tag s in
    Var.Q.tags#set (List.filter f tag_pairs);
    (* WARNING: the url is never displayed (TODO) *)
    let string_of_response i r = (string_of_int i, r.r_info_desc) in
    let str_responses =
      Array.to_list (Array.mapi string_of_response
                       q_info.q_info_responses) in
    Var.Q.responses#set str_responses;
    Var.Q.number_of_votes#set q_info.q_info_num_votes;
    (* winning response(s) *)
    let response_desc r_id = q_info.q_info_responses.(r_id).r_info_desc in
    let str =
      Array.fold_left
        (fun str r_id ->
          str ^ "≫" ^ response_desc r_id ^ "≪" ^ "   ") ""
        q_info.q_info_elected_responses in
    Var.Q.winning_responses#set str;
    if Var.logged_in#get then ( (* get own vote *)
      let vote = ppe (s_"Cannot get own vote: ") srv#get_vote q_id Var.login#get in
```

```
      let vote = Array.to_list vote in
      let responses =
        Array.to_list (Array.mapi (fun i r -> (i, r.r_info_desc))
                         q_info.q_info_responses) in
      let votes_with_desc,_ = Misc.split_responses vote responses in
      let string_id (id, desc) = (string_of_int id, desc) in
      Var.V.selected#set "";
      Var.V.list#set (List.map string_id votes_with_desc)
    )
  with Display_error str -> error str

let fill_question_area srv =
  fill_question_list srv;

  let q_id = Var.Q.selected#get in
  let last_q_id = Var.Q.previous#get in
  if q_id <> last_q_id then (
    Var.Q.previous#set q_id;
    Var.Bool.add_ans#set false;
    Var.Bool.tag_question#set false;
    if q_id <> "" then (Var.Bool.add_question#set false);
    try
      let q_id = int_of_string Var.Q.selected#get in
      fill_question_details q_id srv
    with Failure("int_of_string") -> ()
  )
```

```
let submit_vote () =
  let vote_ids = List.map (fun (id,_) -> int_of_string id) Var.V.list#get in
  try do_in_server (fun srv ->
    let q_id = int_of_string Var.Q.selected#get in
    ppe (s_"Vote failed: ") srv#vote q_id (Array.of_list vote_ids);
    fill_question_details q_id srv)
  with Display_error msg -> error msg

let submit_tags () =
  let tags = List.map (fun (x,_) -> int_of_string x) Var.Q.tags#get in
  let q_id = int_of_string Var.Q.selected#get in
  try do_in_server (fun srv ->
    let tag id = ppe (s_"Cannot tag question: ") srv#tag_question q_id id in
    let untag id = ppe (s_"Cannot untag question: ") srv#untag_question q_id id in
    let old_tags = srv#get_question_tags q_id in
    Array.iter untag old_tags;
    List.iter tag tags)
  with Display_error msg -> error msg


let prepare_question_addition () =
  let change_assoc key v al =
    let rec f = function
      | [] -> assert false
      | (k,x)::tl -> if k = key then (k,v) :: tl else (k,x) :: f tl
    in f al
  in
  let set var id v = var#set (change_assoc id v var#get) in
  let ids = List.map fst Var.Add.ans_desc_list#get in
  let f id =
    let a = Norm.normalize_response (List.assoc id Var.Add.ans_desc_list#get) in
    let u = Norm.normalize_link (List.assoc id Var.Add.ans_url_list#get) in
    set Var.Err.ans_desc_list id ""; set Var.Err.ans_url_list id "";
    Var.Err.general#set "";
    let check f v errvar =
      try f v with Norm.Invalid_format ->
        set errvar id "E";
        error (s_"Invalid response or link format")
    in
    if a <> "" then check (fun x -> Norm.check_response x) a Var.Err.ans_desc_list;
    check Norm.check_link u Var.Err.ans_url_list;
    (a, u)
  in
  let question = Norm.normalize_question Var.Add.q_desc#get in
  (try Norm.check_question question with Norm.Invalid_format ->
    Var.Err.q_desc#set "E";
    Var.Err.general#set (s_"Invalid question format"));
  (question, List.map f ids)
```

74 ⟨*pages.ml* 67a⟩+≡ ◁73

```
module Login = struct
  open Factory
  open Page
  open Session
  let e name ?attr body = Xml.mk_elem ~name ?attr body

  let pageref = ref Page.empty
  let button = Page.button ~pageref

  (* buttons *)
  let style = Button_style "height:100%"
  let login = button ~label:(s_"Login") ~style (fun () ->
    try
      if Var.login#get = ""
      then error (s_"Please fill in a user name")
      else (
        do_in_server ignore;
        (*Var.Bool.nav#set true;*)
        Session.update Var.session_id#get (fun r ->
          { r with login = Var.login#get; pass = Var.password#get });
        change_page Browse
      )
    with
    | ServerConnection.Protocol_warning (_, _, _) ->
        (* todo: we don't display warning message. We should show it *)
        Var.Bool.nav#set true;
        change_page Browse
    | ServerConnection.Login_error (msg, _, _) -> error msg;
    | other_exception ->
        let msg =
          Printf.sprintf
            (f_"Unknown error message (%s). Please report it to demexp-dev@nongnu.org.")
            (Printexc.to_string other_exception) in
        error msg
  )
  (*let new_user = button ~label:(s_"New user") (fun () ->
    Session.update Var.session_id#get (fun r ->
      { r with login = "" });
    Var.password#set "";
    Var.password_confirm#set "";
    Var.Bool.reg_success#set false;
    change_page Register
  )*)

  let prepare dlg =
    Session.update Var.session_id#get (fun r ->
      { r with pass = "" });
    Var.password#set ""; (* they are not automatically synched *)
    ignore dlg;
    let title = s_"demexp login" in
    let body =
      e "p" (s_"Please enter your login and password:") ^
      e "table" (
        e "tr" (
          e "td" (
            e "table" (
              e "tr" (
```

```
              e "td" (s_"Username: ") ^ e "td" (Page.text ~var:Var.login ())
            ) ^
            e "tr" (
              e "td" (s_"Password: ") ^ e "td" (Page.password ~var:Var.password ())
            )
          )
        ) ^
        e "td" (login#xml ())
      )
    ) ^
    if Conf.has_admin_account#get
    then (
      Html.br ^
      "Don't have an account? Sign up at the " ^
      Html.a ~href:(goto_page ~output:'URI Register) "register page" ^ "."
    )
    else ""
  in
  Var.body#set (Page.std ~title body)

let handle dlg = Page.handle ~page:!pageref dlg#event
end


module Register = struct
  open Factory
  open Page
  let e name ?attr body = Xml.mk_elem ~name ?attr body

  let pageref = ref Page.empty
  let button = Page.button ~pageref

  (* buttons *)
  let register = button ~label:(s_"Register") (fun () ->
    Var.Err.general#set "";
    let login = Conf.admin_login#get
    and pass = Conf.admin_password#get in
    try do_in_server ~login ~pass (fun srv ->
      raise_if (Var.password#get <> Var.password_confirm#get)
        (Display_error (s_"The passwords are not identical."));
      ignore (srv#add_participant Var.login#get Var.password#get
                (Array.of_list Conf.default_new_user_groups#get));
      Var.Bool.reg_success#set true)
    with Display_error msg -> error msg
  )

  let prepare dlg =
    (* TODO: variables shared with login page must be cleaned *)
    Var.password_confirm#set "";
    ignore dlg;
    let title = s_"demexp register" in
    let body =
      if not Var.Bool.reg_success#get then (
        e "p" (s_"Please enter your desired login and password:") ^
        e "table" (
          e "tr" (
            e "td" (s_"Login: ") ^ e "td" (Page.text ~var:Var.login ())
          ) ^
```

```
          e "tr" (
            e "td" (s_"Password: ") ^ e "td" (Page.password ~var:Var.password ())
          ) ^
          e "tr" (
            e "td" (s_"Repeat password: ") ^
            e "td" (Page.password ~var:Var.password_confirm ())
          )
        ) ^
        Html.br ^
        register#xml ()
      ) else (
        e "p" (e "b" (
          s_"Account creation was successfull." ^ Html.br ^
          s_"You can now go back to the login page to login." ^ Html.br
        ))
      )
    in
    Var.body#set (Page.std ~title body)

  let handle dlg = Page.handle ~page:!pageref dlg#event
end


module Personal = struct
  open Factory
  open Page
  let e name ?attr body = Xml.mk_elem ~name ?attr body

  let string_of_timestamp ts =
    (* hehe, lets simplify the year some, why wont we? *)
    let ts = Int32.to_int ts in
    let calc a b = (a / b, a mod b) in
    let (years,ts) = calc ts (365 * 24 * 60 * 30) in
    let (fakemonths,ts) = calc ts (30 * 24 * 60 * 30) in
    let (days,ts) = calc ts (24 * 60 * 30) in
    let (h,ts) = calc ts (60 * 30) in
    let (m,_) = calc ts 30 in
    Printf.sprintf "%i-%02i-%02i %02i:%02i"
      (2005 + years) fakemonths days h m

  let all_questions srv =
    let max_id = srv#max_question_id in
    srv#question_info 0 max_id

  let questions_with_my_vote srv q_ids =
    let login = Var.login#get in
    let f l id = if srv#get_vote id login = [||] then l else id :: l in
    List.fold_left f [] q_ids

  let update_since_vote srv voted =
    let vote_stamps = (Session.get Var.session_id#get).Session.vote_timestamps in
    let f l (id,x) =
      try
        let vstamp = Hashtbl.find vote_stamps id in
        if Cache.timestamp srv#cache (Cache.Question id) > vstamp
        then (id,x)::l
        else l
      with Not_found -> (id,x)::l
```

```
  in List.fold_left f [] voted

let updates srv all_q =
  let f l (id,x) =
    try
      let timestamp = Cache.timestamp srv#cache (Cache.Question id) in
      (timestamp,(id,x))::l
    with Not_found -> raise Not_found
  in
  let unsorted = List.fold_left f [] all_q in
  List.sort (fun a b -> compare (fst b) (fst a)) unsorted


type ('a,'b) data =
  { unvoted : 'a; update_since_vote : 'a;
    updates : 'b }

let get_data srv =
  let f (id, (desc, tags)) = (id, (desc, List.map srv#name_of_tag tags)) in
  let all_q = List.map f srv#all_questions in
  let all_ids = List.map fst all_q in
  let voted_on = questions_with_my_vote srv all_ids in
  let rec split a_list b_list = function
    | [] -> (a_list, b_list)
    | (id,desc)::tl -> if List.mem id voted_on
                       then split ((id,desc)::a_list) b_list tl
                       else split a_list ((id,desc)::b_list) tl
  in
  let (voted, unvoted) = split [] [] all_q in
  { unvoted = unvoted;
    update_since_vote = update_since_vote srv voted;
    updates = updates srv all_q }

let prepare () =
  assert Var.logged_in#get;

  let data = do_in_server get_data in

  let disp_tags tags =
    let f str tag =
      if Kludge.display_tag tag then str ^ " - " ^ tag else str
    in
    List.fold_left f "" tags
  in
  let disp_vote (id, (desc,tags)) =
    Html.tr (
      Html.td (disp_tags tags ^ " - ") ^
      Html.td (
        Html.a ~href:(goto ~output:'URI ~question:id ()) desc ^ Html.br
      )
    )
  in
  (*let disp_vote_w_ts (ts, (id, (desc,tags))) =
    Html.tr (
      Html.td (string_of_timestamp ts) ^
      Html.td (disp_tags tags ^ " - ") ^
      Html.td (
        Html.a ~href:(goto ~output:'URI ~question:id ()) desc ^ Html.br
```

```
        )
      )
    in*)

    let maptoxml f l = String.concat "" (List.map f l) in

    let title = Printf.sprintf (f_"Personal page for %s") Var.login#get in
    let body =
      (*"let's pretend now is: " ^ string_of_timestamp (Timestamp.current ()) ^*)
      (*e "h2" "Latest updates:" ^
      Html.table (maptoxml disp_vote_w_ts data.updates) ^*)
      e "h2" "Questions with updates since you voted in them:" ^
      e "i" "Let's pretend this lists only questions with updates and not all you have voted in
      Html.table (maptoxml disp_vote data.update_since_vote) ^
      e "h2" "Questions you have not (yet) voted in:" ^
      Html.table (maptoxml disp_vote data.unvoted)
    in
    Var.body#set (Page.std ~title body)

  let handle () = ()
end


module Logout_confirm = struct
  let prepare () =

    (* do the "logging out" *)
    Session.update Var.session_id#get (fun r ->
      { r with Session.pass = "" });

    Var.body#set (Page.std ~title:(s_"You are now logged out") "")
  let handle () = ()
end


module Browse = struct
  open Factory
  open Page
  let e name ?attr body = Xml.mk_elem ~name ?attr body

  let pageref = ref Page.empty
  let button = Page.button ~pageref
  let x_button = Page.x_button ~pageref

  (* buttons *)
  let submit_new_response = button ~label:"Submit" (fun () ->
    let desc = Var.Add.ans_desc#get in
    let link = Var.Add.ans_url#get in
    let q_id = int_of_string Var.Q.selected#get in
    try
      do_in_server (fun srv -> srv#add_response q_id desc link);
      Var.Bool.add_ans#set false;
      Var.Add.ans_desc#set "";
      Var.Add.ans_url#set "";
      Var.Q.previous#set ""; (* ugly *)
    with Display_error msg ->
      error (s_"Error while submitting response: "^msg)
  )
```

```
let cancel_new_response = button ~label:(s_"Cancel") (fun () ->
  Var.Add.ans_desc#set "";
  Var.Bool.add_ans#set false
)
let move_vote_up = x_button ~label:(s_"up") ~style:Link (fun id ->
  let rec f = function
    | [] -> assert false
    | v::[] -> [v]
    | v::(i,desc)::tl when i = id -> (i,desc)::v::tl
    | v::tl -> v :: f tl
  in Var.V.list#set (f Var.V.list#get);
     submit_vote ()
)
let move_vote_down = x_button ~label:(s_"down") ~style:Link (fun id ->
  let rec f = function
    | [] -> assert false
    | v::[] -> [v]
    | (i,desc)::v::tl when i = id -> v::(i,desc)::tl
    | v::tl -> v :: f tl
  in Var.V.list#set (f Var.V.list#get);
     submit_vote ()
)
let remove_vote = x_button ~label:(s_"rem") ~style:Link (fun id ->
  Var.V.list#set (List.filter (fun (x,_) -> x<>id) Var.V.list#get);
  Var.V.selected#set "";
  submit_vote ()
)
let add_vote = x_button ~label:(s_"vote") ~style:Link (fun vote_id ->
  let responses = Var.Q.responses#get in
  let votes = Var.V.list#get in
  let vote = List.find (fun (id,_) -> id = vote_id) responses in
  let rec f = function
    | [] -> [vote]
    | v::[] -> if v = vote then [v] else v::[vote]
    | v1::v2::tl -> if v1 = vote then v1::v2::tl else
                    if v2 = vote then v2::v1::tl else
                                      v1 :: f (v2::tl)
  in
  Var.V.list#set (f votes);
  Var.V.selected#set vote_id;
  submit_vote ()
)
let logout = button ~label:(s_"Logout") (fun () ->
  (* password is reset at load of login page *)
  Session.update Var.session_id#get (fun r ->
    { r with Session.pass = "" });
  Var.password#set ""; (* needs to get synched *)
  Var.logged_in#set false; (* also needs updating *)
  Var.V.list#set [] (* and this... *)
)
let submit_new_tag = button ~label:(s_"Submit") (fun () ->
  try do_in_server (fun srv ->
    ignore (srv#create_tag Var.T.for_addition#get);
    Var.Bool.add_tag#set false)
   with Display_error msg -> error msg
)
let cancel_new_tag = button ~label:(s_"Cancel") (fun () -> Var.Bool.add_tag#set false)
let add_tag_to_question = button ~label:(s_"Add") (fun () ->
```

```
  let applied = Var.Q.tags#get in
  let selected = Var.Add.tag#get in
  let available = Var.T.list#get in
  if List.exists (fun (x,_) -> x = selected) applied then
    error (s_"Tag already in use.")
  else (
    let name = List.assoc selected available in
    Var.Q.tags#set (applied @ [selected, name]);
    submit_tags ()
  )
)
let remove_tag_from_question = x_button ~label:(s_"rem") ~style:Link (fun id ->
  let applied = Var.Q.tags#get in
  if not (List.exists (fun (x,_) -> x = id) applied) then
    error (s_"Can't remove tag #"^id);
  Var.Q.tags#set (List.filter (fun (x,_) -> x <> id) applied);
  submit_tags ()
)
let add_response_to_question = button ~label:(s_"add new") ~style:Link (fun () ->
  Var.Add.ans_desc#set "";
  Var.Add.ans_url#set "";
  Var.Bool.add_ans#set true
)
let start_tagging = button ~label:(s_"manage tags") ~style:Link (fun () ->
  Var.Bool.add_tag#set false; (* we use the same global variable for both *)
  Var.Bool.tag_question#set true
)
let done_tagging = button ~label:(s_"Done tagging") (fun () ->
  Var.Bool.tag_question#set false
)
let submit_new_question = button ~label:(s_"Submit") (fun () ->
  let question, answers = prepare_question_addition () in
  let tag_id = int_of_string Var.T.selected#get in
  try
    if Var.Err.general#get <> "" then raise (Failure "won't commit this");
    do_in_server (fun srv ->
      let q_id = ppe (s_"Cannot add question: ") srv#new_question question in
      let f (desc, url) =
        if desc <> "" then
          ppe (s_"Cannot add response: ") srv#add_response q_id desc url
      in List.iter f answers;
         ppe (s_"Cannot set question status: ") srv#set_question_status q_id 2;
         ppe (s_"Cannot tag question: ") srv#tag_question q_id tag_id;
    );
    Var.Bool.add_question#set false
  with Misc.Display_error msg -> error msg;
     | Failure "won't commit this" -> ()
)
let cancel_new_question = button ~label:(s_"Cancel") (fun () ->
  Var.Bool.add_question#set false
)
let more_answers_while_adding_question = button ~label:(s_"More answers") (fun () ->
  let f x = x#set (x#get @ [string_of_int (List.length x#get + 1), ""]) in
  List.iter f [Var.Add.ans_desc_list; Var.Add.ans_url_list;
               Var.Err.ans_desc_list; Var.Err.ans_url_list]
)
(*let show_nav_area = button ~label:(s_"show") ~style:Link (fun () ->
  Var.Bool.nav#set true
```

```
)
let hide_nav_area = button ~label:(s_"hide") ~style:Link (fun () ->
  Var.Bool.nav#set false
)*)
let add_new_tag = button ~label:(s_"add new") ~style:Link (fun () ->
  Var.Bool.tag_question#set false; (* we use the same global variable *)
  Var.T.for_addition#set "";
  Var.Bool.add_tag#set true
)
let add_new_question = button ~label:(s_"add new") ~style:Link (fun () ->
  Var.Add.q_desc#set "";
  Var.Err.q_desc#set "";
  Var.Q.selected#set ""; (* ugly *)

  let x = ["1",""; "2",""; "3",""] in
  Var.Add.ans_desc_list#set x;
  Var.Add.ans_url_list#set x;
  Var.Err.ans_desc_list#set x;
  Var.Err.ans_url_list#set x;

  Var.Bool.add_question#set true
)


let prepare dlg =
  let showif b str = if b then str else "" in
  let ifloggedin = showif Var.logged_in#get in

  do_in_server (fun srv ->
    fill_tag_list srv;
    fill_question_area srv);

  let navigation_area =
    Html.table ~cellpadding:"5" ~style:"margin-left:10%;margin-right:auto" (
      e "tr" (
        e "td" (
          s_"Tags:" ^
          ifloggedin (" (" ^ add_new_tag#xml () ^ ")") ^ Html.br ^
          let clickf (id,_) = goto ~tag:id () in
          select ~list:Var.T.list ~var:Var.T.selected ~size:10 ~clickf ()
        ) ^
        e "td" (
          s_"Questions: " ^
          showif (Var.T.selected#get <> "" && Var.logged_in#get)
            ("(" ^ add_new_question#xml () ^ ")") ^
          Html.br ^
          let clickf (id,_) = goto ~question:(int_of_string id) () in
          select ~list:Var.Q.list ~var:Var.Q.selected ~size:10 ~clickf ()
        )
      )
    )
  in

  let add_response_area =
    s_"Add a new response:" ^ Html.br ^
    text ~var:Var.Add.ans_desc () ^ Html.br ^
    s_"With this link:" ^ Html.br ^
    text ~var:Var.Add.ans_url () ^ Html.br ^
```

```
  submit_new_response#xml () ^ " " ^ cancel_new_response#xml ()
in

let vote_area =
  Html.table ~style:"background:orange" (
    e "tr" (
      e "td" (
        let rec f n = function
          | [] -> ""
          | (id,desc) :: tl ->
              string_of_int n ^ ". " ^ e "b" desc ^ " (" ^
              move_vote_up#xml id ^ " " ^ move_vote_down#xml id ^ " " ^
              remove_vote#xml id ^ ")" ^ Html.br ^ f (n+1) tl
        in
        s_"Your vote(s):" ^ Html.br ^
        f 1 Var.V.list#get
      )
    )
  )
in

let self_url = dlg#environment.Wd_types.self_url in

let title = s_"Browse position base" in
let body =
  showif (not Var.logged_in#get) (
    Html.p (
      (* This is not translatable. (TODO) *)
      let link = Html.a ~href:(goto_page ~output:‘URI Login) "login" in
      Html.span ~style:"color:red" "You are not logged in." ^
      " Please " ^ link  ^ " to be able to vote. "
    )
  ) ^
  (* navigation area *)
  e "div" ~attr:["style","background:pink"] (
    e "b" (s_"Navigation area ") ^
    if Var.Bool.nav#get then (
      "(" ^ Html.a ~href:(goto ~output:‘URI ~nav:false ()) (s_"hide") ^ ")" ^
      e "div" ~attr:["style","text-align:center"] navigation_area
    ) else (
      "(" ^ Html.a ~href:(goto ~output:‘URI ~nav:true ()) (s_"show") ^ ")" ^ Html.br ^ Html
    )
  ) ^
  (* add tag area *)
  (if Var.Bool.add_tag#get then
    e "p" (
      s_"Name of new tag:" ^ Html.br ^
      text ~var:Var.Add.tag () ^ " " ^
      submit_new_tag#xml () ^ " " ^ cancel_new_tag#xml ()
    )
  else "") ^
  (* question details area *)
  (if Var.Q.selected#get <> "" then (
    e "p" (
      e "i" (s_"Title") ^ ": (#" ^ Var.Q.selected#get ^ ") " ^
      e "b" Var.Q.title#get ^ " (" ^
      Html.a ~href:(self_url^"?question_id="^Var.Q.selected#get) (s_"permlink") ^
      ")" ^ Html.br ^
```

```
    e "i" (s_"Limit date") ^ ": " ^ Var.Q.limit_date#get ^ Html.br ^
    e "table" (
      if not Var.Bool.tag_question#get then (
        e "tr" (
          e "td" (e "i" "Tags" ^ ": ") ^
          e "td" (
            " - " ^ String.concat " - " (List.map snd Var.Q.tags#get) ^ " - " ^
            ifloggedin ("(" ^ start_tagging#xml () ^ ")")
          )
        )
      ) else ( (* manage tags area *)
        e "tr" (
          e "td" (e "i" (s_"Applied tags") ^ ": ") ^
          let rec f = function [] -> "" | (a,b)::tl ->
            b ^ " (" ^ remove_tag_from_question#xml a ^ ")" ^
            Html.br ^ f tl in
          e "td" (f Var.Q.tags#get)
        ) ^
        e "tr" (
          e "td" (e "i" (s_"Available tags") ^ ": ") ^
          e "td" (
            select ~list:Var.T.list ~var:Var.Add.tag () ^ " " ^
            add_tag_to_question#xml () ^ " " ^ done_tagging#xml ()
          )
        )
      )
    ) ^
    e "i" (s_"Responses") ^ ":" ^
    ifloggedin (" (" ^ add_response_to_question#xml () ^ ")") ^
    e "table" (
      e "tr" (
        e "td" (
          e "ul" (
            let rec f = function [] -> "" | (a,b)::tl ->
              e "li" (b ^ ifloggedin (" (" ^ add_vote#xml a ^ ")")) ^
              f tl in
            f Var.Q.responses#get
          )
        ) ^
        e "td" ( (* vote area *)
          if not Var.Bool.add_ans#get && Var.V.list#get <> []
          then vote_area else ""
        ) ^
        e "td" ( (* add response area *)
          if Var.Bool.add_ans#get then add_response_area else ""
        )
      )
    ) ^
    e "i" (s_"Winning response(s)") ^ ": " ^ e "b" Var.Q.winning_responses#get ^
    Html.br ^
    e "i" (s_"Number of votes") ^ ": " ^ string_of_int Var.Q.number_of_votes#get
  )
) else "") ^
(* add question area *)
let error = e "font" ~attr:["color","red"] (e "b" (s_"Invalid format")) in
if Var.Bool.add_question#get then (
  e "p" (
    e "h2" (s_"New question") ^
```

```
          s_"Your question: " ^ text ~var:Var.Add.q_desc ~size:40 () ^
          (if Var.Err.q_desc#get <> "" then error else "") ^ Html.br ^ Html.br ^
          let rec f n max =
            let index = string_of_int n in
            if n <= max then
            e "div" ~attr:["style","background:lightblue;border:none"] (
              e "table" (
                e "tr" (
                  e "td" (s_"Response " ^ index ^ ": ") ^
                  e "td" (
                    text ~var:Var.Add.ans_desc_list ~index ~size:40 () ^
                    if List.assoc index Var.Err.ans_desc_list#get <> ""
                    then error else ""
                  )
                ) ^
                e "tr" (
                  e "td" (s_"With this link: ") ^
                  e "td" (
                    text ~var:Var.Add.ans_url_list ~index ~size:40 () ^
                    if List.assoc index Var.Err.ans_url_list#get <> ""
                    then error else ""
                  )
                )
              )
            ) ^ Html.br ^ f (n+1) max
            else ""
          in f 1 (List.length Var.Add.ans_desc_list#get) ^
          more_answers_while_adding_question#xml () ^ " " ^
          submit_new_question#xml () ^ " " ^ cancel_new_question#xml ()
        )
      ) else ""
    in
    Var.body#set (Page.std ~title body)

  let handle dlg =
    let event = dlg#event in
    Page.handle ~page:!pageref event
end
```

Chapter 18

demexpweb page

18.1 demexpweb.ui aggregation file

This file includes all the other dialogs.

85 ⟨*demexpweb.ui* 85⟩≡

```
<?xml version="1.0"?>

<!-- copyright 2005-2006 David MENTRE -->
<!-- this software is under GNU GPL. See COPYING.GPL file for details -->

<!DOCTYPE ui:application
          PUBLIC "-//NPC//DTD WDIALOG 2.1//EN" "wd_application_2.dtd" [

<!ENTITY include_variables       SYSTEM "variables.ui">

<!ENTITY false "<ui:string-value>0</ui:string-value>">
<!ENTITY true "<ui:string-value>1</ui:string-value>">

]>

<ui:application start-dialog="main">

  <!--<?wd-debug-mode partially-encoded?>-->
  <?wd-prototype-mode?>

  &include_variables;

  <ui:template name="p" from-caller="name body">
    <ui:ifvar variable="cur-page" op="eq" value="$name">
      $body
    </ui:ifvar>
  </ui:template>

  <ui:dialog name="main" start-page="main" lang-variable="lang">
    <ui:variable name="lang"/>
    <ui:variable name="cur-page"/>
    <ui:variable name="error-message"/>
    <ui:variable name="body"/>

    <!-- dialog variables -->
    <ui:variable name="session" type="dialog"/>
    <ui:variable name="tag" type="dialog"/>
```

```
    <ui:variable name="question" type="dialog"/>
    <ui:variable name="vote" type="dialog"/>
    <ui:variable name="bool" type="dialog"/>

    <ui:page name="main">
      <ui:special>$[body]</ui:special>
    </ui:page>
  </ui:dialog>

</ui:application>
```

18.2 OCaml code of demexpweb page

87 ⟨*demexpweb.ml* 87⟩≡

```
open GlobalState
open Session
open Pages

let give_cookie session_id response_header cgi =
  let cookie =
    { Netcgi_types.cookie_name = "session id";
      cookie_value = session_id;
      cookie_expires = None; (* with browser session *)
      cookie_domain = None; (* auto *)
      cookie_path = None; (* something error prone *)
      cookie_secure = cgi#environment#cgi_https }
  in response_header.Wd_types.rh_set_cookie <- [cookie]

let get_session_id cgi response_header =
  let id =
    try let id = List.assoc "session id" cgi#environment#cookies in
        if Session.exists id then Some id else None
    with Not_found -> None
  in
  match id with Some id -> id
   | None -> let id = Session.new_id () in
             give_cookie id response_header cgi; id

let sync_vars session =
  Var.login#set session.login;
  Var.password#set session.pass;
  Var.lang#set session.lang;
  Var.logged_in#set (session.pass <> "")

let main universe name env = object (self)
  inherit Wd_dialog.dialog universe name env

  val get_session_id =
    fun () -> get_session_id env.Wd_types.cgi env.Wd_types.response_header

  initializer
    Var.dlg := Some self;

  method prepare_page () =

    if self#dialog_variable "session" = None then (
      (* this is a fresh connection, we need to initialize some stuff *)
      let init dlg_name =
        let dlg = universe#create env dlg_name in
        self#set_variable dlg_name (Wd_types.Dialog_value (Some dlg))
      in
      List.iter init ["session"; "tag"; "question"; "vote"; "bool"];
    );

    (* TODO: we should give an error on invalid cgi arguments *)
    let cgi = env.Wd_types.cgi in
    let q_id = cgi#argument_value "question_id" in
    let tag_id = cgi#argument_value "tag_id" in
    let () = match cgi#argument_value "navigation" with
```

```
    | "1" -> Var.Bool.nav#set true
    | "0" -> Var.Bool.nav#set false
    | "" -> ()
    | _ -> (* here we should raise an exception *) ()
  in

  let session_id = get_session_id () in
  let session =
    try let s = Session.get session_id in
        try do_in_server ~login:s.login ~pass:s.pass ignore; s
        with ServerConnection.Login_error _ -> Session.default ()
    with Invalid_id -> Session.default ()
  in
  Session.set session_id session;
  Var.session_id#set session_id;
  assert (Session.exists session_id);
  assert (Var.session_id#get = session_id);
  sync_vars session;

  let lang = cgi#argument_value "lang" in
  if lang <> "" then Var.lang#set lang;
  (*if Var.lang#get = "" then Var.lang#set "en";*)
  Unix.putenv "LC_ALL" (match Var.lang#get with
    | "en" -> "en_US.UTF-8"
    | "fr" -> "fr_FR.UTF-8"
    | "sv" -> "sv_SE.UTF-8"
    | "" -> assert false
    | _ -> failwith "unrecognized language code, sorry");

  let () =
    try change_page (page_of_string (cgi#argument_value "page"))
    with InvalidPageName _ -> ()
  in

  Session.update session_id (fun r -> { r with lang = Var.lang#get });

  let () =
    if tag_id <> "" then Var.T.selected#set tag_id;
    if q_id <> "" then Var.Q.selected#set q_id;
    if tag_id = "" && q_id <> "" && cgi#argument_value "navigation" <> "1"
    then Var.Bool.nav#set false
  in

  match cur_page ~default:Browse () with
    | Login -> Login.prepare self
    | Browse -> Browse.prepare self
    | Register -> Register.prepare self
    | Personal -> Personal.prepare ()
    | Logout_confirm -> Logout_confirm.prepare ()


method handle () =
  let session_id = get_session_id () in
  Var.session_id#set session_id;

  (* temporary hack *)
  if Var.password#get = "" then sync_vars (Session.get session_id);
```

```
    assert (Var.session_id#get <> "");

    match cur_page () with
      | Login -> Login.handle self
      | Browse -> Browse.handle self
      | Register -> Register.handle self
      | Personal -> Personal.handle ()
      | Logout_confirm -> Logout_confirm.handle ()

end

let _ =
  let (args,blabla) = DemexpWebGettext.Gettext.init in
  Arg.parse (Arg.align args) ignore blabla;
  let session_file = "demexp_sessions" in (* this is such a bad solution it is
  not worth to make the file a configurable option ^^ *)
  if Sys.file_exists session_file then Session.load session_file;
  at_exit (fun () -> Session.store session_file);
  Wd_run_cgi.run
    ~charset:‘Enc_utf8
    ~reg:(fun universe -> universe#register "main" main)
    ~uifile:"demexpweb.ui"
    ()
```

Part IV

Client (`gtk2-clnt/`)

Chapter 19

Definition of command line flags

Module `Clntflags` defines command line option that are set when the client is launched.

91a ⟨*clntflags.ml* 91a⟩≡ 91b ▷
```
(* copyright 2004 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Config
```

91b ⟨*clntflags.ml* 91a⟩+≡ ◁91a 91c ▷
```
let default_config_dirname = ".demexp"

let flag_dall_dialogs = ref false                  (* --dall-dialogs *)
and flag_dexceptions = ref false                   (* --dexceptions *)
and flag_log = ref false                           (* -l --log *)
and flag_autotests = ref false                     (* --autotests *)
and flag_pref_dir = ref default_config_dirname     (* --preference-dir *)
and flag_url_list : string list ref = ref []       (* command line urls *)
and flag_update_voted_state = ref false            (* --update-voted-state *)
```

We also define the help function `log` which print on output its arguments only if `flag_log` is `true`. `log` can be used in the same way as `printf`.

The trick here is to call `kprintf` as last expression in the function: http://caml.inria.fr/archives/200405/msg00355.html.

91c ⟨*clntflags.ml* 91a⟩+≡ ◁91b
```
let log fmt =
  let print_if_necessary str =
    if !flag_log then (
      Format.printf "%s" str;
      Format.print_newline ()
     ) in
  Format.kprintf print_if_necessary fmt
```

Chapter 20

Miscellaneous GUI (`MiscUI`)

Module `MiscUI` implements miscellaneous helper functions used by other modules in the client.

92a ⟨*miscUI.ml* 92a⟩≡ 92b ▷
```
(* copyright 2005-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open DemexpGettext.Gettext
```

20.1 Display messages to user

We store a global tooltips group for the whole application in this module.

92b ⟨*miscUI.ml* 92a⟩+≡ ◁92a 92c ▷
```
let tooltips_group = GData.tooltips ()
```

Helper function `display_message` opens a dialog to show `message` to the user.

92c ⟨*miscUI.ml* 92a⟩+≡ ◁92b 92d ▷
```
let display_message message =
  let md = GWindow.message_dialog
      ~message
      ~message_type:`INFO
      ~buttons:GWindow.Buttons.ok
      ~modal:true () in
  ignore(md#run ());
  md#destroy ()
```

Helper function `display_error` opens a dialog to show error `message` to the user.

92d ⟨*miscUI.ml* 92a⟩+≡ ◁92c 93a ▷
```
let display_error message =
  let md = GWindow.message_dialog
      ~message
      ~message_type:`ERROR
      ~buttons:GWindow.Buttons.ok
      ~modal:true () in
  ignore(md#run ());
  md#destroy ()
```

Helper function `create_user_msg` create a simple `user_msg` function that, when called with a `str` string parameter, displays it in Gtk `statusbar` identified with `str_id` string identifier.

93a ⟨*miscUI.ml* 92a⟩+≡ ◁92d 93b▷
```
let create_user_msg_fun statusbar str_id =
  let sb_context = statusbar#new_context ~name:str_id in
  let user_msg str =
    sb_context#pop ();
    ignore(sb_context#push str) in
  user_msg
```

20.2 Progress bar

We create a simple interface to display and update a progress bar. It is made of glade window (`ui`) with the progress bar inside it, going from `min` to `max` value.

93b ⟨*miscUI.ml* 92a⟩+≡ ◁93a 93c▷
```
type progress_bar = {
    min : float;
    max : float;
    ui : Demexp_gladeui.progress_bar_window;
  }
```

Helper function `pump_glib_events` refreshes the GUI by letting GLib process all GTK events (refresh, etc.)[1].

93c ⟨*miscUI.ml* 92a⟩+≡ ◁93b 93d▷
```
let pump_glib_events () = while Glib.Main.iteration false do () done
```

Function `open_progress_bar` opens a window with a progress bar inside it. The progress bar goes from `min` to `max`.

93d ⟨*miscUI.ml* 92a⟩+≡ ◁93c 93e▷
```
let open_progress_bar ?(min=0) ?(max=1) ~title () =
  let ui = new Demexp_gladeui.progress_bar_window () in
  ui#toplevel#set_title title;
  ui#the_progress_bar#set_text title;
  ui#toplevel#show ();
  pump_glib_events ();
  { min = float_of_int min; max = float_of_int max; ui = ui}
```

Function `update_progress_bar` updates the progress bar `pb` by setting its current value to `v`.

93e ⟨*miscUI.ml* 92a⟩+≡ ◁93d 93f▷
```
let update_progress_bar ~pb ~v =
  let frac = ((float_of_int v) -. pb.min) /. (pb.max -. pb.min) in
  if frac >= 0.0 && frac <= 1.0 then
    pb.ui#the_progress_bar#set_fraction frac;
  pump_glib_events ()
```

Function `close_progress_bar` destroys the progress bar `pb`.

93f ⟨*miscUI.ml* 92a⟩+≡ ◁93e 94▷
```
let close_progress_bar ~pb = pb.ui#toplevel#destroy ()
```

[1] Many thanks to Olivier ANDRIEU for helping on this.

20.3 Network related code

Function `handle_network_error` calls the function `f` with an `arg`ument and handles any network related exceptions that might be generated by this call. In case of error, it displays the error message and returns `err_ret`.

This function allows to make a systematic handling of error cases in all functions calling the server. The handling of error can optionally be bypassed if `Clntflags.flag_dexceptions` is set to true.

94 ⟨*miscUI.ml* 92a⟩+≡ ◁93f

```
let handle_network_error f arg err_ret =
  let user_msg str =
    Printf.eprintf "%s" str;
    display_error str in
  if !Clntflags.flag_dexceptions then
    f arg
  else
    let print_error_and_continue msg =
      Printf.eprintf "%s" msg;
      display_error msg;
      err_ret in
    try
      f arg
    with
    | Failure "Xdr.pack_xdr_value_as_string" ->
        print_error_and_continue
          (s_ "One of input string is too long. Make it shorter.")
    | Rpc_client.Client_is_down ->
        print_error_and_continue
          (s_ "Connection lost with server. Try to restart the client.")
    | Rpc_client.Message_lost ->
        print_error_and_continue
          (s_ "Unable to connect to the server. Wait or try to restart the client.")
    | Rpc.Rpc_server server_error ->
        let err_msg = Misc.string_of_server_error server_error in
        print_error_and_continue
          (Printf.sprintf (f_ "Connection lost with the server (cause: Rpc.Rpc_server(%s)). Try
    | Misc.Display_error str ->
        print_error_and_continue str
    | Unix.Unix_error (Unix.ECONNRESET, _, _) ->
        print_error_and_continue
          (s_ "TCP connection to server reseted by peer. Try to restart the client.")
    | any ->
        print_error_and_continue
          (Printf.sprintf (f_ "Unknown error message (%s). Please report it to demexp-dev@nongn
             (Printexc.to_string any))
```

Chapter 21

"Preferences" window (`Pref`)

Module `Pref` stores and displays user preferences. It uses module `Config_file` to store them on disk, in file `HOME/.demexp/config`.

This module also state of each question for each accessed server: question seen, voted and timestamp when last seen.

95a ⟨*pref.ml* 95a⟩≡ 95b ▷
```
(* copyright 2004-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Config
open Printf
open Config_file
open DemexpGettext.Gettext
```

Function `to_int_or_default` is a simple helper function that converts its `str` argument in an integer or returns the `default` value.

95b ⟨*pref.ml* 95a⟩+≡ ◁95a 95c ▷
```
let to_int_or_default str default =
  try int_of_string str
  with Failure "int_of_string" -> default
```

The login of a delegate is "`delegate_`" concatenated with the individual login.

95c ⟨*pref.ml* 95a⟩+≡ ◁95b 96a ▷
```
let delegate_prefix = "delegate_"
```

21.1 Wrappers for specific data types

In order to use `Config_file`, we need to define a specific wrapper for our server parameters.

96a ⟨*pref.ml* 95a⟩+≡ ◁95c 96b▷

```
let server_wrappers =
  {to_raw = (fun (server, port, login, password) ->
               Raw.Section [("server", string_wrappers.to_raw server);
                            ("port", int_wrappers.to_raw port);
                            ("login", string_wrappers.to_raw login);
                            ("password", string_wrappers.to_raw password)]);
   of_raw = fun arg ->
     let host = ref default_server_name and port = ref default_server_port
     and login = ref "" and password = ref "" in
     match arg with
     | Raw.Section l ->
         List.iter
           (fun (name,value) -> match name with
              | "server" -> host := string_wrappers.of_raw value
              | "port" -> port := int_wrappers.of_raw value
              | "login" -> login := string_wrappers.of_raw value
              | "password" -> password := string_wrappers.of_raw value
              | s -> Printf.eprintf "Unknown field %s\n%!" s)
           l;
         (!host, !port, !login, !password)
     | r -> raise (Wrong_type
                     (fun outchan -> Printf.fprintf outchan
                         "Raw.Section expected, got %a\n%!" Raw.to_channel r))}
```

We also define a wrapper for `Timestamp` data type.

96b ⟨*pref.ml* 95a⟩+≡ ◁96a 97a▷

```
let timestamp_wrappers = {
  to_raw = (fun v -> Raw.String (Timestamp.to_string v));
  of_raw = function
    | Raw.Int v -> Timestamp.of_int v
    | Raw.Float v -> Timestamp.of_float v
    | Raw.String v -> Timestamp.of_string v
    | r -> raise (Wrong_type
                    (fun outchan -> Printf.fprintf outchan
                        "Raw.Int expected, got %a\n%!" Raw.to_channel r))}
```

21.2 Preference directory

Preferences are stored in a `demexp_directory` (see code chunk 91b for its exact name), in different files.

This directory is created when the preferences object is created (see code chunk 98d).

Function `check_or_create_demexp_directory` creates the `demexp_directory` if it not already exists.

97a ⟨*pref.ml* 95a⟩+≡ ◁96b 97b▷

```
let check_or_create_demexp_directory demexp_directory =
  try
    let stats = Unix.stat demexp_directory in
    match stats.Unix.st_kind with
    | Unix.S_DIR -> ()
    | _ ->
        MiscUI.display_error
          (Printf.sprintf
             (f_ "%s exists but is not a directory, please remove it.")
             demexp_directory);
        exit 2
  with Unix.Unix_error (Unix.ENOENT, _, _) ->
    Unix.mkdir demexp_directory 0o700
```

21.2.1 Format of `config` file

File "`config`" stores the client preferences.

97b ⟨*pref.ml* 95a⟩+≡ ◁97a 98a▷

```
let config_filename = "config"
```

This file is made of a section *servers* that stores the set of servers, with corresponding login and password.

The second section named *browser_state* stores, for each server identified by its server name and port, the state of each question (seen, voted and latest timestamp).

Here is an example of `config` file content:

```
(* List of user servers *)
servers = [{server = server  port = 1234  login = login  password = pass}]
(* Browser state. Format is [(server, port, question_state)] where
   question_state in [(q_id, seen, voted, timestamp)]. *)
browser_state = [(server, 1234, [(42, true, true, "78")])]
```

21.3 Object storing user preferences

We define a class `preferences` that is able to read user preferences from disk and to store any modification back on disk.

In this object, we use `login` data structure type to store complete login information (see code chunk 99b). We also use `question_state` data structure to store the browser state of each question.

98a ⟨*pref.ml* 95a⟩+≡ ◁97b 98b▷

```
type login = {
    mutable user_login: string;
    mutable user_password: string;
  }

type question_state = {
    mutable seen: bool;
    mutable voted: bool;
    mutable timestamp: Timestamp.t;
  }
```

By default, the preference files are stored in user HOME directory, under `dirname`. The path of this `dirname` can be absolute if `relative` argument is set to false, otherwise the `dirname` is relative to the home directory.

Variable `demexp_directory` contains the full path name of the `.demexp` directory.

98b ⟨*pref.ml* 95a⟩+≡ ◁98a 98c▷

```
class preferences user_msg ?(relative = true) dirname () =
  let demexp_directory =
    if relative then (
      let home_path =
        try Sys.getenv "HOME"
        with Not_found ->
          Printf.eprintf (f_ "Warning: HOME environment variable not found, use current directo
          "." in
      home_path ^ "/" ^ dirname
     ) else
      dirname in
```

We also define the full path of the "`config`" file.

98c ⟨*pref.ml* 95a⟩+≡ ◁98b 98d▷

```
  let config_fullpath = demexp_directory ^ "/" ^ config_filename in
```

At login creation, we check that the directory exists.

98d ⟨*pref.ml* 95a⟩+≡ ◁98c 99a▷

```
  let _ = check_or_create_demexp_directory demexp_directory in
```

We define a group and its associated set of configuration parameters that we will need to save/load from the configuration file.

99a ⟨*pref.ml* 95a⟩+≡ ◁98d 99b▷

```
let group = new group in
(* list of servers *)
let help = s_ "List of user servers" in
let servers_cp = new list_cp server_wrappers ~group ["servers"] [] help in
(* browser state *)
let help =
  s_ "Browser state. Format is [(server, port, question_state)] where question_state is [(q_i
let question_wrapper =
  tuple4_wrappers
    int_wrappers bool_wrappers bool_wrappers timestamp_wrappers in
let state_for_a_server_wrappers =
  tuple3_wrappers string_wrappers int_wrappers
    (list_wrappers question_wrapper) in
let browser_state_cp =
  new list_cp state_for_a_server_wrappers ~group ["browser_state"] [] help in
```

In the preference object, information is stored as a hash table `login_table`, associating (server, port) to `login` information.

We also have a hash table pointing from the couple (server, port) to another hash table, itself pointing from question number to corresponding question state.

99b ⟨*pref.ml* 95a⟩+≡ ◁99a 99c▷

```
object (self)
  method preference_dir_name = demexp_directory

  val login_table : (string * int,    login) Hashtbl.t = Hashtbl.create 3
                    (* server * port -> login *)

  val browser_state_table
      : (string * int,    (int,    question_state) Hashtbl.t) Hashtbl.t
      (* server * port -> (q_id -> question_state) *)
      = Hashtbl.create 3
```

Method `set_server_name` and `set_server_port` are used to chose the server that we use for logins. Get methods returns empty strings if the login information do not exist for the chosen server and port.

99c ⟨*pref.ml* 95a⟩+≡ ◁99b 100▷

```
  val mutable server_name = default_server_name
  val mutable server_port = default_server_port

  method server_name = server_name
  method server_port = server_port

  method set_server_name v = server_name <- v
  method set_server_port v = server_port <- v
```

We define methods to access user login and password.

100 ⟨*pref.ml* 95a⟩+≡ ◁99c 101▷

```
method user_info =
  try
    Hashtbl.find login_table (server_name, server_port)
  with Not_found ->
    let l = { user_login = ""; user_password = "" } in
    Hashtbl.add login_table (server_name, server_port) l;
    l

method user_login = (self#user_info).user_login

method user_password = (self#user_info).user_password

method delegate_login = delegate_prefix ^ (self#user_info).user_login

method delegate_password = (self#user_info).user_password

method set_user_login v = (self#user_info).user_login <- v

method set_user_password v = (self#user_info).user_password <- v
```

We define methods to access browser state.

101 ⟨*pref.ml* 95a⟩+≡ ◁100 102a▷

```
method question_state server port q_id =
  try
    let h = Hashtbl.find browser_state_table (server, port) in
    try
      Hashtbl.find h q_id
    with Not_found ->
      let s = {seen=false; voted=false; timestamp=Timestamp.of_int 0} in
      Hashtbl.add h q_id s;
      s
  with Not_found ->
    let s = {seen=false; voted=false; timestamp=Timestamp.of_int 0} in
    let h = Hashtbl.create 3 in
    Hashtbl.add h q_id s;
    Hashtbl.add browser_state_table (server, port) h;
    s

method question_seen server port q_id =
  (self#question_state server port q_id).seen

method question_voted server port q_id =
  (self#question_state server port q_id).voted

method question_timestamp server port q_id =
  (self#question_state server port q_id).timestamp

method set_question_seen server port q_id new_seen =
  (self#question_state server port q_id).seen <- new_seen

method set_question_voted server port q_id new_voted =
  (self#question_state server port q_id).voted <- new_voted

method set_question_timestamp server port q_id new_timestamp =
  (self#question_state server port q_id).timestamp <- new_timestamp

method iter_question server port f =
  try
    let h = Hashtbl.find browser_state_table (server, port) in
    Hashtbl.iter f h
  with Not_found -> ()
```

The method `load` retrieves the preferences from disk.

102a ⟨*pref.ml* 95a⟩+≡ ◁101 102b▷

```
method load =
  (* read from file *)
  group#read config_fullpath;
  (* copy list of servers into our hash table *)
  Hashtbl.clear login_table;
  let add_server (server, port, login, password) =
    Hashtbl.add login_table (server, port)
      {user_login=login; user_password=password} in
  List.iter add_server servers_cp#get;
  (* copy browser state into our hash table *)
  Hashtbl.clear browser_state_table;
  let add_question_state h (q_id, seen, voted, timestamp) =
    Hashtbl.add h q_id {seen=seen; voted=voted; timestamp=timestamp} in
  let add_server_state (server, port, question_states) =
    let h = Hashtbl.create 3 in
    List.iter (add_question_state h) question_states;
    Hashtbl.add browser_state_table (server, port) h in
  List.iter add_server_state browser_state_cp#get
```

Methode `save` saves the preferences to disk.

102b ⟨*pref.ml* 95a⟩+≡ ◁102a 102c▷

```
method save =
  (* get list of servers *)
  let server_fold (server, port) login l =
    (server, port, login.user_login, login.user_password) :: l in
  servers_cp#reset;
  servers_cp#set (Hashtbl.fold server_fold login_table []);
  (* get browser state *)
  let question_fold q_id q_state l =
    (q_id, q_state.seen, q_state.voted, q_state.timestamp) :: l in
  let server_state_fold (server, port) h l =
    let h_as_list = Hashtbl.fold question_fold h [] in
    (server, port, h_as_list) :: l in
  browser_state_cp#reset;
  browser_state_cp#set
    (Hashtbl.fold server_state_fold browser_state_table []);
  (* write it to disk *)
  group#write config_fullpath
```

We finally finish the definition of the `preferences` object (cf. code chunk 98b).

102c ⟨*pref.ml* 95a⟩+≡ ◁102b 103a▷

```
end
```

21.4 Graphical preferences window

todo: Several preferences windows can be open at the same time. Probably confusing for the user. TO DO

In all code of this section, `pref` is an object of `preferences` as defined in previous section and `ui_pref` is an object of class `Demexp_gladeui.preferences`.

Function `save_changes` is called when the user clicks on "Save changes" button. It gets the text widget contents, check for correctness of user input. If entries are valid, it updates the `pref` object with them and then save them on disk.

103a ⟨*pref.ml* 95a⟩+≡ ◁102c 103b▷

```
let save_changes pref ui_pref () =
  (* normalize logins *)
  let login = Norm.normalize_login ui_pref#entry_login5#text in
  ui_pref#entry_login5#set_text login;
  (* save if correct *)
  try
    Norm.check_login login;
    pref#set_server_name ui_pref#entry_server_name9#text;
    pref#set_server_port
      (to_int_or_default ui_pref#entry_server_port10#text pref#server_port);
    pref#set_user_login login;
    pref#set_user_password ui_pref#entry_password6#text;
    pref#save;
    ui_pref#toplevel#destroy ()
  with Norm.Invalid_format ->
    MiscUI.display_error (Printf.sprintf (f_ "Invalid user login"))
```

Function `ui_preferences` is called to display the preferences setting window. It just setup the dialog with content of `pref` object and then wait for user input.

103b ⟨*pref.ml* 95a⟩+≡ ◁103a 104▷

```
let ui_preferences pref () =
  let ui_pref = new Demexp_gladeui.preferences () in
  ignore(ui_pref#save_changes12#connect#clicked
           ~callback:(save_changes pref ui_pref));
  ignore(ui_pref#cancel13#connect#clicked
           ~callback:ui_pref#toplevel#destroy);
  (* setup dialog content *)
  pref#load;
  ui_pref#entry_login5#set_text pref#user_login;
  ui_pref#entry_password6#set_text pref#user_password;
  ui_pref#entry_server_name9#set_text pref#server_name;
  ui_pref#entry_server_port10#set_text (string_of_int pref#server_port);
  ui_pref#toplevel#show ()
```

21.5 Autotests

todo: It would be better to generate `dirname` similarly to `Filename.temp_file`. TO DO

104 ⟨*pref.ml* 95a⟩+≡ ◁103b

```
let _ =
  if Config.do_autotests then begin
    Printf.printf "  pref autotests...";
    let dirname = "/tmp/test-demexprc" in
    let loginname = dirname ^ "/" ^ config_filename in
    if Sys.file_exists loginname then Sys.remove loginname;
    let user_msg str = print_string str in
    let p = new preferences user_msg ~relative:false dirname () in
    (* load empty prefs, set value and store back to disk *)
    p#load;
    p#set_server_name "server";
    p#set_server_port 1234;
    assert(p#user_login = "");
    assert(p#question_seen "s" 1 42 = false);
    p#set_user_login "login";
    p#set_user_password "pass";
    p#set_question_seen "s" 1 42 true;
    p#set_question_voted "s" 1 42 true;
    p#set_question_timestamp "s" 1 42 (Timestamp.of_int 0x45);
    p#save;
    (* reload prefs and check everything is correct *)
    let p2 = new preferences user_msg ~relative:false dirname () in
    p2#load;
    assert(p2#user_login = "");
    p2#set_server_name "server";
    p2#set_server_port 1234;
    assert(p2#user_login = "login");
    assert(p2#user_password = "pass");
    assert(p2#delegate_login = delegate_prefix ^ "login");
    assert(p2#delegate_password = "pass");
    assert(p2#question_seen "s" 1 42 = true);
    assert(p2#question_voted "s" 1 42 = true);
    assert(p2#question_timestamp "s" 1 42 = Timestamp.of_int 0x45);
    (* erase test logins file *)
    Sys.remove loginname;
    Printf.printf "done\n"
  end
```

Chapter 22

Keeping records of user actions (`Clerk`)

Module `Clerk` is an interface to the `Pref` object and is used to store and check that a given question has been seen by the user or voted on. The `Pref` object is responsible of storing that information on disk.

105a ⟨*clerk.ml* 105a⟩≡ 105b ▷
```
(* copyright 2005-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Perf
open Str
open Printf
open Messages_aux
open Messages_clnt
open DemexpGettext.Gettext
```

22.1 Data structure

We define the data structure that stores the state of each question.

105b ⟨*clerk.ml* 105a⟩+≡ ◁ 105a 105c ▷
```
type t = {
    cache : Cache.t;
    pref : Pref.preferences;
    server_name : string;
    server_port : int;
  }
```

22.2 Usage

In all following code, `clerk` is of type `Clerk.t`.

Function `save` saves onto disk the state of `clerk`.

105c ⟨*clerk.ml* 105a⟩+≡ ◁ 105b 106a ▷
```
let save ~clerk =
  clerk.pref#save
```

Function `create` makes a new clerk, loading its data from disk if they exist.

106a ⟨*clerk.ml* 105a⟩+≡ ◁105c 106b▷

```
let create cache pref server_name server_port =
  pref#load;
  { cache = cache;
    pref = pref;
    server_name = server_name;
    server_port = server_port; }
```

Function `voted` tells if question of identifier `q_id` is marked as voted.

106b ⟨*clerk.ml* 105a⟩+≡ ◁106a 106c▷

```
let voted ~clerk ~q_id =
  clerk.pref#question_voted clerk.server_name clerk.server_port q_id
```

Function `seen` tells if question of identifier `q_id` is marked as having been seen.

106c ⟨*clerk.ml* 105a⟩+≡ ◁106b 106d▷

```
let seen ~clerk ~q_id =
  clerk.pref#question_seen clerk.server_name clerk.server_port q_id
```

Helper function `update_question_timestamp` update the value of the saved timestamp to the value currently in the cache. We do nothing of the searched timestamp is not found in the cache (**todo**: Should we fix this? Why this behavior?). TO DO

106d ⟨*clerk.ml* 105a⟩+≡ ◁106c 106e▷

```
let update_question_timestamp ~clerk ~q_id =
  try
    let cache_timestamp = Cache.timestamp clerk.cache (Cache.Question q_id) in
    clerk.pref#set_question_timestamp clerk.server_name clerk.server_port q_id
      cache_timestamp
  with Not_found -> ()
```

Function `mark_as_seen` registers that we have seen question of identifier `q_id`.

106e ⟨*clerk.ml* 105a⟩+≡ ◁106d 106f▷

```
let mark_as_seen ~clerk ~q_id =
  clerk.pref#set_question_seen clerk.server_name clerk.server_port q_id true;
  update_question_timestamp ~clerk ~q_id
```

Function `mark_as_not_seen` registers that we have *not* seen question of identifier `q_id`.

106f ⟨*clerk.ml* 105a⟩+≡ ◁106e 106g▷

```
let mark_as_not_seen ~clerk ~q_id =
  clerk.pref#set_question_seen clerk.server_name clerk.server_port q_id false;
  update_question_timestamp ~clerk ~q_id
```

Function `mark_as_voted` registers that we have voted on question of identifier `q_id`.

106g ⟨*clerk.ml* 105a⟩+≡ ◁106f 107a▷

```
let mark_as_voted ~clerk ~q_id =
  clerk.pref#set_question_voted clerk.server_name clerk.server_port q_id true;
  update_question_timestamp ~clerk ~q_id
```

Function `update_seen_questions_from_cache` checks if some seen questions have been updated in the `cache` and in that case mark them as not seen. This function should be called each time a new clerk is created (i.e. loaded).

107a ⟨*clerk.ml* 105a⟩+≡ ◁106g 107b▷

```
let update_seen_questions_from_cache ~clerk =
  let mark_not_seen_if_updated q_id _ =
    try
      let cache_timestamp = Cache.timestamp clerk.cache (Cache.Question q_id)
      and saved_timestamp =
        clerk.pref#question_timestamp clerk.server_name
          clerk.server_port q_id in
      if cache_timestamp > saved_timestamp then mark_as_not_seen ~clerk ~q_id
    with Not_found ->
      (* cache don't have the timestamp, so probably new or updated *)
      mark_as_not_seen ~clerk ~q_id in
  clerk.pref#iter_question clerk.server_name clerk.server_port
    mark_not_seen_if_updated
```

Function `determine_voted_state` should be called when we want to force the voted state of questions from server state. This function gets for each question the voted state and update the clerk state accordingly.

107b ⟨*clerk.ml* 105a⟩+≡ ◁107a

```
let determine_voted_state ~clerk ~client ~cookie =
  let mark_voted_if_voted_on_server q_id _ =
    let ret =
      Demexp.V1.get_vote client (cookie, q_id, clerk.pref#user_login) in
    if ret.get_vote_rc <> rt_ok then
      raise (Misc.Display_error
               (Printf.sprintf (f_ "Cannot get my own vote. Error: %s")
                  (Misc.string_of_return_code ret.get_vote_rc)));
    if Array.to_list ret.get_vote <> [] then
      mark_as_voted ~clerk ~q_id in
  let timer = Perf.timer_start () in
  clerk.pref#iter_question clerk.server_name clerk.server_port
    mark_voted_if_voted_on_server;
  Perf.timer_stop_and_record "Clerk.determine_voted_state" timer
```

Chapter 23

"Manage users" window (`Users`)

Module `Users` stores and displays participants. The main widget in the window is a Gtk Tree View. This view is composed of the graphical view itself and a model that stores the data for the view.

108a ⟨*users.ml* 108a⟩≡ 108b ▷
```
(* copyright 2004-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Messages_aux
open Messages_clnt
open DemexpGettext.Gettext
```

23.1 Helper functions

Helper function `display_statusbar_message` display in status bar defined by context `sb_context` the error message corresponding to `code`.

108b ⟨*users.ml* 108a⟩+≡ ◁ 108a 108c ▷
```
let display_statusbar_message sb_context code =
  sb_context#pop ();
  ignore(sb_context#push (Misc.string_of_return_code code))
```

Helper function `string_of_group_array` transforms an array of strings into a string where previous array elements are separated by commas.

108c ⟨*users.ml* 108a⟩+≡ ◁ 108b 109a ▷
```
let string_of_group_array a =
  let s = Array.fold_left (fun str group -> str ^ group ^ ",") "" a in
  let len = String.length s in
  if len >= 1 then
    String.sub s 0 (len - 1) (* brute force removal of last ',' *)
  else s
```

23.2 Model

The model itself is defined as an object of class `users_model`. This class contains principally static variables that can be returned by calling the method of the same name.

109a ⟨*users.ml* 108a⟩+≡ ◁108c 109b▷

```
class users_model sb_context client cookie cache =
  let columns = new GTree.column_list in
  let col_id = columns#add Gobject.Data.int in
  let col_login = columns#add Gobject.Data.string in
  let col_password = columns#add Gobject.Data.string in
  let col_groups = columns#add Gobject.Data.string in
  let store = GTree.list_store columns in
```

By default, sorting is done on by user id.

109b ⟨*users.ml* 108a⟩+≡ ◁109a 109c▷

```
  let _ = store#set_sort_column_id col_id.GTree.index 'ASCENDING in
```

At initialization time, the model gets from the server the list of participants with corresponding information. At first, we get the maximum participant identifier.

109c ⟨*users.ml* 108a⟩+≡ ◁109b 110a▷

```
  let max_id : int =
    let f () =
      let ret = Demexp.V1.max_participant_id client cookie in
      if ret.max_participant_id_rc = rt_ok then
        ret.max_participant_id
      else (
        display_statusbar_message sb_context ret.max_participant_id_rc;
        -1
       ) in
    MiscUI.handle_network_error f () (-1) in
```

Then, we get the participant info per group of `number` by calling helper function `fill_store` and we add them to the model by calling `fill_one`.

110a ⟨*users.ml* 108a⟩+≡ ◁109c 110b▷

```
let _ =
  let fill_one info =
    let iter = store#append () in
    store#set ~row:iter ~column:col_id info.info_id;
    store#set ~row:iter ~column:col_login info.info_login;
    store#set ~row:iter
      ~column:col_password info.info_password;
    store#set ~row:iter
      ~column:col_groups (string_of_group_array info.info_groups) in
  let max_number = Rtypes.int_of_uint4 max_number_ids in
  let f () =
    let rec fill_store base =
      let number_to_get =
        if max_id - base + 1 <= max_number then max_id - base + 1
        else max_number in
      if base <= max_id then (
        let ret =
          Cache.participant_info cache client
            (cookie, base, number_to_get) in
        if ret.participant_info_rc = rt_ok then (
          Array.iter fill_one ret.participant_info;
          fill_store (base + number_to_get)
         ) else
          display_statusbar_message sb_context ret.participant_info_rc
       ) in
    fill_store 0 in
  MiscUI.handle_network_error f () () in
```

We finally define the methods of class `users_model`.

110b ⟨*users.ml* 108a⟩+≡ ◁110a 111a▷

```
object
  method col_id = col_id
  method col_login = col_login
  method col_password = col_password
  method col_groups = col_groups
  method store = store
end
```

23.3 Graphical management of users

Function `update_view` adds needed columns to `view` and connects it to the `model`.

111a ⟨*users.ml* 108a⟩+≡ ◁110b 111b▷

```
let update_view ~view ~model () =
  (* id column *)
  let col = GTree.view_column ~title:(s_ "id")
      ~renderer:(GTree.cell_renderer_text [], ["text", model#col_id]) () in
  col#set_sort_column_id model#col_id.GTree.index;
  col#set_sort_indicator true;
  ignore(view#append_column col);
  (* login column *)
  let col = GTree.view_column ~title:(s_ "login")
      ~renderer:(GTree.cell_renderer_text [], ["text", model#col_login]) () in
  col#set_sort_column_id model#col_login.GTree.index;
  col#set_sort_indicator true;
  ignore(view#append_column col);
  (* password column *)
  let col = GTree.view_column ~title:(s_ "login")
      ~renderer:(GTree.cell_renderer_text [], ["text", model#col_password]) () in
  col#set_sort_column_id model#col_password.GTree.index;
  col#set_sort_indicator true;
  ignore(view#append_column col);
  (* groups column *)
  let col = GTree.view_column ~title:(s_ "groups")
      ~renderer:(GTree.cell_renderer_text [], ["text", model#col_groups]) () in
  col#set_sort_column_id model#col_groups.GTree.index;
  col#set_sort_indicator true;
  ignore(view#append_column col);
  view#set_rules_hint true;
  view#selection#set_mode `SINGLE;
  view#set_model (Some model#store#coerce)
```

Callback `modified_selection` is called each time a row is selected or unselected in the Tree View of participants. When a row is selected, corresponding row items are made editable in entry boxes.

Note: The typing of `gtk_model` forced to `#GTree.model` is black magic to me (I copied it from Gtk Tree View tutorial). But without that, it does not compile.

111b ⟨*users.ml* 108a⟩+≡ ◁111a 112▷

```
let modify_selection ui_users model path currently_selected =
  let gtk_model : #GTree.model = model#store in
  let row = gtk_model#get_iter path in
  if not currently_selected then (
    let login = gtk_model#get ~row ~column:model#col_login
    and password = gtk_model#get ~row ~column:model#col_password
    and groups = gtk_model#get ~row ~column:model#col_groups in
    ui_users#entry_login14#set_text login;
    ui_users#entry_password15#set_text password;
    ui_users#entry_groups19#set_text groups
   );
  true (* allow selection state to change *)
```

Callback `remove_callback` is called when user clicks on button "Remove". Firstly, we get user confirmation that selected participants should be deleted from server. Then, for each selected row, this function gets the row login, and it removes the login on the server. If successful, the corresponding row on display is removed.

112 ⟨*users.ml* 108a⟩+≡ ◁111b 113▷

```
let remove_callback ui_users model sb_context client cookie cache () =
  let f () =
    let confirm_removal () =
      let md = GWindow.message_dialog
          ~message:(s_ "Are you sure you want to remove selected participant(s)?")
          ~message_type:`QUESTION
          ~buttons:GWindow.Buttons.ok_cancel
          ~modal:true () in
      let res = md#run () = `OK in
      md#destroy () ;
      res in
    let gtk_model : #GTree.model = model#store in
    let paths = ui_users#treeview_users11#selection#get_selected_rows in
    let remove_row path =
      let row = gtk_model#get_iter path in
      let login = gtk_model#get ~row ~column:model#col_login in
      let p_id = gtk_model#get ~row ~column:model#col_id in
      let ret = Demexp.V1.remove_participant client (cookie, login) in
      if ret = rt_ok then (
        ignore(model#store#remove row);
        Cache.invalidate cache (Cache.Participant p_id)
      ) else
        display_statusbar_message sb_context ret in
    if confirm_removal () then
      List.iter remove_row paths in
  MiscUI.handle_network_error f () ()
```

Callback `add_callback` is called when user clicks the "Add" button. It gets participant information from editable fields and tries to add it onto the server. If successful, it gets the new record from the server and update display accordingly.

113 ⟨*users.ml* 108a⟩+≡ ◁112 114▷

```
let add_callback ui_users model sb_context client cookie () =
  let f () =
    (* get fields data *)
    let login = ui_users#entry_login14#text
    and password = ui_users#entry_password15#text
    and str_groups = ui_users#entry_groups19#text in
    let groups = Str.split (Str.regexp ",") str_groups in
    (* add participant on server *)
    let ret =
      Demexp.V1.add_participant client (cookie, login, password,
                                        Array.of_list groups) in
    if ret.add_participant_rc = rt_ok then (
      (* get new record from server *)
      let ret2 = Demexp.V1.participant_info client (cookie,
                                                    ret.add_participant_id, 1) in
      if ret2.participant_info_rc = rt_ok then (
        (* update display *)
        let store : GTree.list_store = model#store in
        let append_row info =
          let iter = model#store#append () in
          store#set ~row:iter ~column:model#col_id info.info_id;
          store#set ~row:iter ~column:model#col_login info.info_login;
          store#set ~row:iter
            ~column:model#col_password info.info_password;
          store#set ~row:iter
            ~column:model#col_groups (string_of_group_array info.info_groups) in
        Array.iter append_row ret2.participant_info;
        (* erase entry boxes *)
        ui_users#entry_login14#set_text "";
        ui_users#entry_password15#set_text "";
        ui_users#entry_groups19#set_text ""
       ) else
        display_statusbar_message sb_context ret2.participant_info_rc
     ) else
      display_statusbar_message sb_context ret.add_participant_rc in
  MiscUI.handle_network_error f () ()
```

Callback `update_callback` is called when user clicks on "Update" button. Firstly, it gets the currently selected display row and its identifier. Then it gets the information from editable text fields and use it to update participant record on server. If successful, it gets the participant record from the server to update the display.

Note: In function `update_row`, we could theoretically get several records and always update the same `iter` row. However, as we request only one record, this cannot happen.

114 ⟨*users.ml* 108a⟩+≡ ◁113 115a▷

```
let update_callback ui_users (model : users_model) sb_context
    client cookie cache () =
  (* get current selected row *)
  let paths = ui_users#treeview_users11#selection#get_selected_rows in
  let gtk_model : #GTree.model = model#store in
  match paths with
  | [] -> ()
  | path :: _ ->
      let f () =
        let iter = gtk_model#get_iter path in
        let id = gtk_model#get ~row:iter ~column:model#col_id in
        (* get fields data *)
        let login = ui_users#entry_login14#text
        and password = ui_users#entry_password15#text
        and str_groups = ui_users#entry_groups19#text in
        let groups = Str.split (Str.regexp ",") str_groups in
        (* update on server *)
        let rc =
          Demexp.V1.update_participant client (cookie, login, password,
                                               Array.of_list groups) in
        if rc = rt_ok then (
          Cache.invalidate cache (Cache.Participant id);
          (* get new record from server *)
          let ret2 = Demexp.V1.participant_info client (cookie, id, 1) in
          if ret2.participant_info_rc = rt_ok then (
            (* update display *)
            let store : GTree.list_store = model#store in
            let update_row info =
              store#set ~row:iter ~column:model#col_id info.info_id;
              store#set ~row:iter ~column:model#col_login info.info_login;
              store#set ~row:iter
                ~column:model#col_password info.info_password;
              store#set ~row:iter
                ~column:model#col_groups
                (string_of_group_array info.info_groups) in
            Array.iter update_row ret2.participant_info
           ) else
            display_statusbar_message sb_context ret2.participant_info_rc
         ) else
          display_statusbar_message sb_context rc in
      MiscUI.handle_network_error f () ()
```

Helper function `setup_window` prepares the widgets and sets needed callbacks.

115a ⟨*users.ml* 108a⟩+≡ ◁114 115b ▷

```
let setup_window ui_users sb_context client cookie cache =
  let model = new users_model sb_context client cookie cache in
  update_view ~view:ui_users#treeview_users11 ~model ();
  ignore(ui_users#button_remove20#connect#clicked
           ~callback:(remove_callback ui_users model sb_context
                        client cookie cache));
  ignore(ui_users#button_add49#connect#clicked
           ~callback:(add_callback ui_users model sb_context client cookie));
  ignore(ui_users#button_update50#connect#clicked
           ~callback:(update_callback ui_users model sb_context
                        client cookie cache));
  ui_users#treeview_users11#selection#set_select_function
    (modify_selection ui_users model)
```

Function `ui_manage_users` is called to display the user management window.

115b ⟨*users.ml* 108a⟩+≡ ◁115a

```
let ui_manage_users client cookie cache () =
  let ui_users = new Demexp_gladeui.manage_users () in
  let sb_context = ui_users#statusbar2#new_context ~name:"manage users sb" in
  (* we show dialog immediately so we can tell what we are doing in
     the status bar *)
  ui_users#toplevel#show ();
  try
    setup_window ui_users sb_context client cookie cache
  with Failure msg -> ignore(sb_context#push msg)
```

Chapter 24

"Manage tags" window (`Tags`)

Module `Tags` allow an administrator to manage tags: add new ones and rename existing ones.

116a ⟨*tags.ml* 116a⟩≡ 116b ▷
```
(* copyright 2004-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Messages_aux
open Messages_clnt
open DemexpGettext.Gettext
```

24.1 Tags backend and view

We define the `tags_backend` that contains the data displayed in the list of tags widget.

116b ⟨*tags.ml* 116a⟩+≡ ◁ 116a 116c ▷
```
class tags_backend () =
  let columns = new GTree.column_list in
  let col_id = columns#add Gobject.Data.int in
  let col_new = columns#add Gobject.Data.boolean in
  let col_voted = columns#add Gobject.Data.boolean in
  let col_label = columns#add Gobject.Data.string in
  let store = GTree.tree_store columns in

  let _ = store#set_sort_column_id col_label.GTree.index 'ASCENDING in

  object
    method col_id = col_id
    method col_new = col_new
    method col_voted = col_voted
    method col_label = col_label
    method store = store
  end
```

In order to present the tags in a hierarchy, we will sort them in a `tag_tree`.

116c ⟨*tags.ml* 116a⟩+≡ ◁ 116b 117a ▷
```
type tag_tree = Tag_node of char * tag_tree list  * int *  string
              (* hierarchy_level *    children    *  id * tag_name  *)
```

Helper function `char_list_of_string` return a string `s` as a list of its character (copied from http://caml.inria.fr/pub/ml-archives/caml-list/2005/02/1b5f499bf035a8a9b2a52d0ce0357640.en.html).

117a ⟨*tags.ml* 116a⟩+≡ ◁116c 117b▷

```
let char_list_of_string s =
  let l = ref [] in
  String.iter (fun c -> l := c :: !l) s;
  List.rev !l
```

Helper function `extract_hierarchy_level` extracts, from a tag `label`, the hierarchy level of the tag as a list of characters. For example, tag `aa: Rennes` gives level `['a'; 'b']`.

117b ⟨*tags.ml* 116a⟩+≡ ◁117a 117c▷

```
let hierarchy_level_re = Str.regexp "^\\([a-zA-Z0-9]+\\):"

let extract_hierarchy_level (id, label) =
  if Str.string_match hierarchy_level_re label 0 then
    (char_list_of_string (Str.matched_group 1 label), id, label)
  else ([], id, label)
```

Function `insert_into_hierarchy` takes a new tag with its current level (`tag_hier`) and inserts it into the `hierarchy_list`. This is done by finding, at each level of the hierarchy, the tag having the same level and inserting the tag in its `node_children`.

117c ⟨*tags.ml* 116a⟩+≡ ◁117b 118a▷

```
let rec insert_into_hierarchy hierarchy_list (tag_hier, id, label) =
  match tag_hier with
  | [ lvl ] -> Tag_node(lvl, [], id, label) :: hierarchy_list
  | [] -> Tag_node(' ', [], id, label) :: hierarchy_list
  | lvl_hd :: lvl_tail ->
      let same_level, other_level =
        List.partition
          (fun (Tag_node(c, _, _, _)) -> c = lvl_hd) hierarchy_list in
      match same_level with
      | [] ->
          let children =
            insert_into_hierarchy [] (lvl_tail, id, label) in
          [Tag_node(lvl_hd, children, -1, "")]
            @ other_level
      | [Tag_node(node_lvl, node_children, node_id, node_label)] ->
          let new_node_children =
            insert_into_hierarchy node_children (lvl_tail, id, label) in
          [Tag_node(node_lvl, new_node_children, node_id, node_label)]
            @ other_level
      | _ -> assert(false)
```

Function `tree_of_tag_hash` converts a hash-table of tags indexed by their ids into a hierarchy of tags.

118a ⟨*tags.ml* 116a⟩+≡ ◁117c 118b▷

```
let tree_of_tag_hash tag_hash =
  (* gather tags in a list *)
  let add_to_list id label tl = (id, label) :: tl in
  let tag_list = Hashtbl.fold add_to_list tag_hash [] in
  (* sort it *)
  let string_order (_, label1) (_, label2) = compare label1 label2 in
  let sorted_tags = List.sort string_order tag_list in
  (* add the hierarchy level to each tag in the list *)
  let add_hierarchy_level (id, label) =
    let hl = extract_hierarchy_level (id, label) in
    (hl, id, label) in
  let tags_with_level = List.map extract_hierarchy_level sorted_tags in
  (* and finally build the hierarchy tree *)
  List.fold_left insert_into_hierarchy [] tags_with_level
```

Function `add_tag_row` add a new row with containing `id` and `label` to given tag `backend`. Optional argument `skip_question_tags` is set to true if we do not want to display question's specific tags. Optional argument `row_callback` is a function that is call with the newly created row iter as argument.

118b ⟨*tags.ml* 116a⟩+≡ ◁118a 119a▷

```
let add_tag_row ?(skip_question_specific_tags=false)
    ?(row_callback=(fun _row -> ())) backend id label =
  if not (skip_question_specific_tags
            && (Norm.is_question_specific_tag label)) then (
    let store : GTree.tree_store = backend#store in
    let row = store#append () in
    store#set ~row ~column:backend#col_id id;
    store#set ~row ~column:backend#col_label
      (Misc.add_line_splits 30 label);
    row_callback row
   )
```

Function `fill_tag_store` fill up a `store` for tag `backend` with set of `tags` (given as a hash table, from id to label).

Optional argument `skip_question_tags` is set to true if we do not want to display question's specific tags. Optional argument `row_callback` is a function that is call with the newly created row iter as argument.

119a ⟨*tags.ml* 116a⟩+≡ ◁118b 119b▷

```
let fill_tag_store ?(skip_question_specific_tags=false)
    ?(row_callback=(fun _row -> ())) backend tag_hash =
  let store : GTree.tree_store = backend#store in
  store#clear ();
  let tree = tree_of_tag_hash tag_hash in
  let rec add_tree_node ?(parent=None) level_list =
    match level_list with
    | [] -> ()
    | Tag_node(_, children, id, label) :: level_tail ->
        if not (skip_question_specific_tags
                  && (Norm.is_question_specific_tag label)) then (
          let store : GTree.tree_store = backend#store in
          let row =
            match parent with
            | None -> store#append ()
            | Some p -> store#append ~parent:p () in
          store#set ~row ~column:backend#col_id id;
          store#set ~row ~column:backend#col_label
            (Misc.add_line_splits 30 label);
          row_callback row;
          add_tree_node ~parent:(Some row) children;
          add_tree_node ~parent level_tail
         )
        else add_tree_node ~parent level_tail in
  add_tree_node tree
```

Helper function `set_column_tooltip` set the tooltip `tip` on column `col` of a Tree View. We start from the `col` widget and lookup recursively until the column button is found.

Many thanks to Arnold GUILLAUMOT <arnold.guillaumot@wanadoo.fr> for this code (see http://wwwfun.kurims.kyoto-u.ac.jp/soft/olabl/lablgtk-list/1315).

119b ⟨*tags.ml* 116a⟩+≡ ◁119a 120▷

```
let set_column_tooltip ~tip ~col =
  let rec get_button w =
    match w with
      None -> None
    | Some p ->
        begin
          let name = Gobject.Type.name (Gobject.get_type p#as_widget) in
          if name = "GtkButton"
              (* try to cast our GObj.widget to a GButton.button *)
          then
            try
              Some (new GButton.button (Gobject.try_cast p#as_widget name))
            with _ -> get_button p#misc#parent
                (* check the parent of our GObj.widget *)
          else get_button p#misc#parent
        end in
  match get_button col#widget with
    None -> ()
  | Some b ->
      let tooltips = GData.tooltips () in
      MiscUI.tooltips_group#set_tip b#coerce ~text:tip
```

Helper function `connect_tags_view` setup the view for a widget displaying tags. It setup two columns: one for tag ids and one for tag labels.

If `new_voted_updated_flags` is true, two new columns are added: "new" (`N`) and "voted" (`V`).

120 ⟨*tags.ml* 116a⟩+≡ ◁119b 121a▷

```
let connect_tag_view ?(for_browser=false) ?(mode='SINGLE) ~view ~backend () =
  let add_text_column ~title ?(tip="") ~backend_col () =
    let col = GTree.view_column ~title
        ~renderer:(GTree.cell_renderer_text [], ["text",
                                                 backend_col]) () in
    col#set_sort_column_id backend_col.GTree.index;
    col#set_sort_indicator true;
    (* set an explicit label so its widget can be used to set the tooltip *)
    let label = GMisc.label ~text:title () in
    col#set_widget (Some label#coerce);
    ignore(view#append_column col);
    if tip <> "" then set_column_tooltip ~tip ~col in
  let add_toggle_column ~title ?(tip="") ~backend_col () =
    let renderer = GTree.cell_renderer_toggle [] in
    let col = GTree.view_column ~title
        ~renderer:(renderer, ["active", backend_col]) () in
    col#set_sort_column_id backend_col.GTree.index;
    col#set_sort_indicator true;
    (* set an explicit label so its widget can be used to set the tooltip *)
    let label = GMisc.label ~text:title () in
    col#set_widget (Some label#coerce);
    ignore(view#append_column col);
    if tip <> "" then set_column_tooltip ~tip ~col in
  if not for_browser then (
    (* id column *)
    let tip = s_ "Unique identifier of the tag" in
    add_text_column ~title:(s_ "id") ~tip ~backend_col:backend#col_id ()
   );
  if for_browser then (
    (* new column *)
    let tip = s_ "New\n\nIf checked, there is new information on this question, like a new vote
    add_toggle_column ~title:(s_ "N") ~tip ~backend_col:backend#col_new ();
    (* voted column *)
    let tip = s_ "Voted\n\nIf checked, you have voted on this question." in
    add_toggle_column ~title:(s_ "V") ~tip ~backend_col:backend#col_voted ()
   );
  (* label column *)
  add_text_column ~title:(s_ "tag") ~backend_col:backend#col_label ();

  view#set_rules_hint true;
  view#selection#set_mode mode;
  view#set_model (Some backend#store#coerce)
```

24.2 Window management

Callback `tag_clicked` is called each time the user clicks in a row in the list of tags. The function simply gets the tag label and displays it in the editable text box.

121a ⟨*tags.ml* 116a⟩+≡ ◁120 121b▷

```
let tag_clicked ui backend path currently_selected =
  let model : #GTree.model = backend#store in
  let row = model#get_iter path in
  if not currently_selected then (
    let tag_label = model#get ~row ~column:backend#col_label in
    ui#entry_tag_name16#set_text tag_label
   ) else
    ui#entry_tag_name16#set_text "";
  true (* allow selection state to change *)
```

Callback `rename_callback` is called when the user clicks on "Rename" button. It gets the new label and update the tag on the server.

121b ⟨*tags.ml* 116a⟩+≡ ◁121a 121c▷

```
let rename_callback ui backend client cookie cache user_msg () =
  (* get current selected row in tags list *)
  let paths = ui#treeview_tags12#selection#get_selected_rows in
  let model : #GTree.model = backend#store in
  match paths with
  | [] -> ()
  | path :: _ -> (* only one row can be selected simultaneously *)
      let f () =
        let iter = model#get_iter path in
        let tag_id = model#get ~row:iter ~column:backend#col_id in
        let tag_label = ui#entry_tag_name16#text in
        let ret = Demexp.V1.update_tag client (cookie, tag_id, tag_label) in
        if ret = rt_ok then (
          let store : GTree.tree_store = backend#store in
          store#set iter backend#col_label tag_label;
          Cache.invalidate cache (Cache.Tag tag_id);
          user_msg "" (* success so cleanup display *)
         ) else
          user_msg (Printf.sprintf (f_ "Cannot rename tag #%d: %s.")
                      tag_id (Misc.string_of_return_code ret)) in
      MiscUI.handle_network_error f () ()
```

Callback `add_callback` is called when the user clicks on "Add" button. It gets the new label and tries to create the tag on the server.

121c ⟨*tags.ml* 116a⟩+≡ ◁121b 122a▷

```
let add_callback ui backend client cookie user_msg () =
  let f () =
    let tag_label = ui#entry_tag_name16#text in
    let ret = Demexp.V1.create_tag client (cookie, tag_label) in
    if ret.create_tag_rc = rt_ok then (
      add_tag_row backend ret.create_tag_id tag_label;
      ui#entry_tag_name16#set_text "";
      user_msg "" (* success so cleanup display *)
     ) else
      user_msg (Printf.sprintf (f_ "Cannot create tag with label \"%s\": %s.")
                  tag_label (Misc.string_of_return_code ret.create_tag_rc)) in
  MiscUI.handle_network_error f () ()
```

Function `setup_window` prepares the widgets and sets needed callbacks.

122a ⟨*tags.ml* 116a⟩+≡ ◁121c 122b▷

```
let setup_window ui backend client cookie cache user_msg =
  let tags = Hashtbl.create 3 in
  Cache.update_tags_hash tags client cookie cache;
  fill_tag_store backend tags;
  connect_tag_view ~view:ui#treeview_tags12 ~backend ();
  (* connect callbacks *)
  ui#treeview_tags12#selection#set_select_function
    (tag_clicked ui backend);
  ignore(ui#button_rename48#connect#clicked
           ~callback:(rename_callback ui backend
                        client cookie cache user_msg));
  ignore(ui#button_add47#connect#clicked
           ~callback:(add_callback ui backend client cookie user_msg))
```

Function `ui_manage_tags` is called to display the user management window.

122b ⟨*tags.ml* 116a⟩+≡ ◁122a 123▷

```
let ui_manage_tags client cookie cache () =
  let ui = new Demexp_gladeui.manage_tags () in
  let user_msg =
    MiscUI.create_user_msg_fun ui#statusbar4 "manage tags sb" in
  let backend = new tags_backend () in
  (* we show dialog immediately so we can tell what we are doing in
     the status bar *)
  ui#toplevel#show ();
  try
    setup_window ui backend client cookie cache user_msg
  with Failure msg -> user_msg msg
```

24.3 Autotests

123 ⟨*tags.ml* 116a⟩+≡ ◁122b

```
let _ =
  if Config.do_autotests then begin
    Printf.printf "  tags autotests...";
    let h = Hashtbl.create 3 in
    Hashtbl.add h 1 "a: City";
    assert(tree_of_tag_hash h =
           [Tag_node ('a', [], 1, "a: City")]);
    Hashtbl.add h 0 "zero";
    Hashtbl.add h 2 "aa: Rennes";
    Hashtbl.add h 3 "ab: New York";
    assert(tree_of_tag_hash h =
           [Tag_node (' ', [], 0, "zero");
            Tag_node ('a',
                      [Tag_node ('b', [], 3, "ab: New York");
                       Tag_node ('a', [], 2, "aa: Rennes")],
                      1, "a: City")]);
    Hashtbl.add h 4 "one";
    assert(tree_of_tag_hash h =
           [Tag_node (' ', [], 0, "zero");
            Tag_node (' ', [], 4, "one");
            Tag_node ('a',
                      [Tag_node ('b', [], 3, "ab: New York");
                       Tag_node ('a', [], 2, "aa: Rennes")],
                      1, "a: City")]);
    Hashtbl.add h 5 "aa: Rio";
    assert(tree_of_tag_hash h =
           [Tag_node (' ', [], 0, "zero");
            Tag_node (' ', [], 4, "one");
            Tag_node ('a',
                      [Tag_node ('b', [], 3, "ab: New York");
                       Tag_node ('a', [], 5, "aa: Rio");
                       Tag_node ('a', [], 2, "aa: Rennes")],
                      1, "a: City")]);
    Printf.printf "done\n"
  end
```

Chapter 25

"New question" window (Newquestion)

Module Newquestion manages the window that allows to enter a new question on the demexp server.

124a ⟨*newquestion.ml* 124a⟩≡ 124b ▷
```
(* copyright 2004-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Messages_aux
open Messages_clnt
open DemexpGettext.Gettext
```

25.1 Responses backend (aka model) and corresponding view

We define in this section the backend that supports the tree widget displaying added reponses and links. As in Users module, we define it as an object. We use an additional response_hash that contains the set of already added responses in order to check that a response is not added twice.

Note: In following callbacks, when the click on a button succeeds, we remove the previous message in the toolbar (if it exists) because we consider that previous error has been corrected.

124b ⟨*newquestion.ml* 124a⟩+≡ ◁ 124a 125a ▷
```
class responses_backend () =
  let columns = new GTree.column_list in
  let col_response = columns#add Gobject.Data.string in
  let col_link = columns#add Gobject.Data.string in
  let store = GTree.list_store columns in
  let response_hash : (string, string) Hashtbl.t = Hashtbl.create 3 in

  object
    method col_response = col_response
    method col_link = col_link
    method store = store
    method response_hash = response_hash
  end
```

Function `update_view` adds needed columns to `view` and connects them to the `backend`.

125a ⟨*newquestion.ml* 124a⟩+≡ ◁124b 125b▷

```
let update_view ~view ~backend () =
  (* response column *)
  let col = GTree.view_column ~title:(s_ "response")
      ~renderer:(GTree.cell_renderer_text [], ["text",
                                               backend#col_response]) () in
  ignore(view#append_column col);
  (* link column *)
  let col = GTree.view_column ~title:(s_ "link")
      ~renderer:(GTree.cell_renderer_text [], ["text",
                                               backend#col_link]) () in
  ignore(view#append_column col);
  view#set_rules_hint true;
  view#selection#set_mode `SINGLE;
  view#set_model (Some backend#store#coerce)
```

25.2 New question window

Callback `remove_response` is called when user clicks on button "Remove". It just gets the selected row and remove it from backend.

Note: We consider only one path because only one row can be selected at a time (see end of code chunk 125a).

125b ⟨*newquestion.ml* 124a⟩+≡ ◁125a 126a▷

```
let remove_response ui_new_question backend sb_context () =
  (* get current selected row *)
  let paths = ui_new_question#treeview23#selection#get_selected_rows in
  let model : #GTree.model = backend#store in
  match paths with
  | [] -> ()
  | path :: _ ->
      let iter = model#get_iter path in
      let response = model#get ~row:iter ~column:backend#col_response in
      ignore(model#remove iter);
      Hashtbl.remove backend#response_hash response;
      sb_context#pop ()
```

Callback `add_response` is called when the user clicks on button "Add response". It simply gets the response and link from corresponding text entry, normalize the response and put both of them in the view backend.

126a ⟨*newquestion.ml* 124a⟩+≡ ◁125b 126b▷

```
let add_response ui_new_question backend sb_context () =
  let response = Norm.normalize_response ui_new_question#entry_response22#text
  and link = ui_new_question#entry_link23#text in
  try
    Norm.check_response response;
    if not (Hashtbl.mem backend#response_hash response) then (
      let store : GTree.list_store = backend#store in
      let iter = backend#store#append () in
      store#set ~row:iter ~column:backend#col_response response;
      store#set ~row:iter ~column:backend#col_link link;
      Hashtbl.add backend#response_hash response link;
      (* cleanup entry boxes *)
      ui_new_question#entry_response22#set_text "";
      ui_new_question#entry_link23#set_text "";
      sb_context#pop ()
     ) else (
      let msg =
        Printf.sprintf
          (f_ "Can't add response, '%s' already in list") response in
      sb_context#pop (); (* remove previous message *)
      ignore(sb_context#push msg)
     )
  with Norm.Invalid_format ->
    sb_context#pop (); (* remove previous message *)
    ignore(sb_context#push (s_ "Can't add empty response"))
```

Helper function `confirm_addition` display a dialog to check that the user really wants to add a `question` on server.

126b ⟨*newquestion.ml* 124a⟩+≡ ◁126a 127a▷

```
let confirm_addition question =
  let msg = Printf.sprintf
      (f_ "Are you sure you want to add question '%s'? You won't be able to change its title la
      question in
  let md = GWindow.message_dialog
      ~message:msg
      ~message_type:`QUESTION
      ~buttons:GWindow.Buttons.ok_cancel
      ~modal:true () in
  let res = md#run () = `OK in
  md#destroy ();
  res
```

Callback `add_question` is called when the user clicks on the "Add question" button. After normalizing the question and checking its formatting correctness, we ask the user if he really wants to add this question. If yes, then we get the question content and calls needed RPC on the server.

fixme: We should add an exception handler to catch Xdr.?? exception in case of RPC error. FIXME

127a ⟨*newquestion.ml* 124a⟩+≡ ◁126b 127b▷

```
let add_question ui_new_question backend client cookie () =
  let question =
    Norm.normalize_question ui_new_question#entry_question19#text in
  ui_new_question#entry_question19#set_text question;
  try
    Norm.check_question question;
    if confirm_addition question then
      let f () =
        (* add question *)
        let ret = Demexp.V1.new_question client (cookie, question) in
        if ret.question_id_return_code <> rt_ok then (
          let msg =
            Printf.sprintf (f_ "Cannot add question '%s': %s")
              question
              (Misc.string_of_return_code ret.question_id_return_code) in
          raise (Misc.Display_error msg)
         );
        let q_id = ret.question_id_id in
        (* add responses *)
        let register_response response link =
          let ret =
            Demexp.V1.add_response client (cookie, q_id, response, link) in
          if ret <> rt_ok then
            let msg =
              Printf.sprintf (f_ "Cannot add response '%s': %s")
                response (Misc.string_of_return_code ret) in
            raise (Misc.Display_error msg) in
        Hashtbl.iter register_response backend#response_hash;
        (* and finally remove the whole "New question" window *)
        ui_new_question#new_question#destroy () in
      MiscUI.handle_network_error f () ()
  with Norm.Invalid_format ->
    MiscUI.display_error (s_ "Invalid empty question")
```

Helper function `setup_window` prepares the widgets and sets needed callbacks.

127b ⟨*newquestion.ml* 124a⟩+≡ ◁127a 128▷

```
let setup_window ui_new_question sb_context client cookie =
  let backend = new responses_backend () in
  update_view ~view:ui_new_question#treeview23 ~backend ();
  ignore(ui_new_question#button_cancel52#connect#clicked
           ~callback:ui_new_question#new_question#destroy);
  ignore(ui_new_question#button_add_response49#connect#clicked
           ~callback:(add_response ui_new_question backend sb_context));
  ignore(ui_new_question#button_remove50#connect#clicked
           ~callback:(remove_response ui_new_question backend sb_context));
  ignore(ui_new_question#button_add_question51#connect#clicked
           ~callback:(add_question ui_new_question backend client cookie))
```

Function `ui_new_question` is called to display the "New question" window.

128 ⟨*newquestion.ml* 124a⟩+≡ ◁127b

```
let ui_new_question client cookie () =
  let ui_new_question = new Demexp_gladeui.new_question () in
  let sb_context = ui_new_question#statusbar6#new_context
      ~name:"new question sb" in
  (* we show dialog immediately so we can tell what we are doing in
     the status bar *)
  ui_new_question#toplevel#show ();
  try
    setup_window ui_new_question sb_context client cookie
  with Failure msg -> ignore(sb_context#push msg)
```

Chapter 26

"Classification" window (`Clsf`)

Module `Clsf` allows demexp classifiers to do classification.

129 ⟨*clsf.ml* 129⟩≡ 130 ▷

```
(* copyright 2004-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Messages_aux
open Messages_clnt
open DemexpGettext.Gettext
```

26.1 Window backends

We define in this section the three backends that stores the data needed to display the three tree widgets of the classification window. For this, we define two kinds of backends: `tag_backend` for the two tree widgets showing tags, and `question_backend` for the tree widget displaying the set of available questions.

A `tag_backend` contains two columns: the id and the label of each tag.

Moreover, backend `tag_backend` contains a hash table `tags_hash` of all stored tags. Associated methods `add_tag` and `remove_tag` allows to manipulate this hash table. Method `clear` removes all entries in the backend.

fixme: Right now, the hash table is not managed jointly with the store. It might be more clean to manage them together. FIXME

130 ⟨*clsf.ml* 129⟩+≡ ◁129 131a▷

```
class tag_backend () =
  let columns = new GTree.column_list in
  let col_id = columns#add Gobject.Data.int in
  let col_new = columns#add Gobject.Data.boolean in
  let col_voted = columns#add Gobject.Data.boolean in
  let col_label = columns#add Gobject.Data.string in
  let store = GTree.tree_store columns in

  let _ = store#set_sort_column_id col_label.GTree.index 'ASCENDING in

  object
    method col_id = col_id
    method col_new = col_new
    method col_voted = col_voted
    method col_label = col_label
    method store = store

    val mutable tags_hash : (int, string) Hashtbl.t = Hashtbl.create 3
    method tags_hash = tags_hash
    method set_tags_hash h = tags_hash <- h
    method add_tag id label = Hashtbl.add tags_hash id label
    method remove_tag id = Hashtbl.remove tags_hash id

    method clear () =
      Hashtbl.clear tags_hash;
      store#clear ()
  end
```

A `question_backend` contains three columns: the identifier and the descriptor of each question, as well as a check box indicating if the question is public or not. It also stores the `selected_question`.

131a ⟨*clsf.ml* 129⟩+≡ ◁130 131b▷

```
class question_backend () =
  let columns = new GTree.column_list in
  let col_public = columns#add Gobject.Data.boolean in
  let col_id = columns#add Gobject.Data.int in
  let col_desc = columns#add Gobject.Data.string in
  let store = GTree.list_store columns in

  let _ = store#set_sort_column_id col_id.GTree.index 'DESCENDING in

  object
    method col_id = col_id
    method col_desc = col_desc
    method col_public = col_public
    method store = store

    val mutable selected_question : int option = None
    method selected_question = selected_question
    method set_selected_question s = selected_question <- s
  end
```

All those backends are grouped in a common data structure.

131b ⟨*clsf.ml* 129⟩+≡ ◁131a 132▷

```
type backends = {
    tags : tag_backend;
    question_tags : tag_backend;
    questions : question_backend;
  }
```

26.2 Views setup

In this section, we prepare the views of the different tree widgets.

26.2.1 Tag views

All the functions related to tag views are defined in section 24.1.

26.2.2 Question view

Function `fill_question_store` gets questions from server and adds for each one of them a new row in `backend`'s store if question's status is `tagging_only`. In case of error, an error message is display in status bar.

132 ⟨*clsf.ml* 129⟩+≡ ◁131b 133▷

```
let fill_question_store backend client cookie cache =
  let store : GTree.list_store = backend#store in
  store#clear ();
  let add_row id desc public =
    let iter = store#append () in
    store#set ~row:iter ~column:backend#col_id id;
    store#set ~row:iter ~column:backend#col_desc
      (Misc.add_line_splits 50 desc);
    store#set ~row:iter ~column:backend#col_public public in
  let add_question q =
    add_row q.q_id q.q_desc (q.q_info_status = public) in
  let f () =
    let ret = Demexp.V1.max_question_id client cookie in
    if ret.max_question_id_rc <> rt_ok then
      raise (Misc.Display_error
               (Printf.sprintf (f_ "unable to get max_question_id (%s)")
                  (Misc.string_of_return_code ret.max_question_id_rc)));
    let number = Rtypes.int_of_uint4 max_number_ids in
    let rec get_some_questions base max_id =
      if base <= max_id then (
        let ret = Cache.question_info cache client (cookie, base, number) in
        if ret.question_info_rc <> rt_ok then
          raise (Misc.Display_error
                   (Printf.sprintf
                      (f_ "unable to get info for question %d to %d (%s)")
                      base (base + number)
                      (Misc.string_of_return_code ret.question_info_rc)));
        Array.iter add_question ret.question_info;
        get_some_questions (base + number) max_id
       ) in
    get_some_questions 0 ret.max_question_id in
  MiscUI.handle_network_error f () ()
```

Callback `public_toggled` is called when the user clicks on a check box in the "public" column of question list. It gets the current state from the tree widget backend and sets the opposite state on the server.

133 ⟨*clsf.ml* 129⟩+≡ ◁132 134▷

```
let public_toggled client cookie cache backends user_msg
    ~(model : GTree.list_store) ~column path =
  let row = model#get_iter path in
  let is_public = model#get ~row ~column in
  match backends.questions#selected_question with
  | None -> ()
  | Some q_id ->
      let new_state, new_state_str =
        if is_public then
          (tagging_only, (s_ "tagging_only"))
        else
          (public, (s_ "public")) in
      let f () =
        let ret =
          Demexp.V1.set_question_status client (cookie, q_id, new_state) in
        if ret = rt_ok then (
          ignore(model#set ~row ~column (not is_public));
          Cache.invalidate cache (Cache.Question q_id)
        ) else
          user_msg (Printf.sprintf (f_ "cannot set client status to %s: %s")
                      new_state_str (Misc.string_of_return_code ret)) in
      MiscUI.handle_network_error f () ()
```

Function `connect_question_view` connects the `backend` to the `view` displaying the list of questions. It also adds the three graphical columns ("id", "descriptor" and "public") in the `view`.

134 ⟨*clsf.ml* 129⟩+≡ ◁133 135a▷

```
let connect_question_view client cookie cache
    backends user_msg ~view ~backend =
  (* public column *)
  let renderer = GTree.cell_renderer_toggle [] in
  ignore(renderer#connect#toggled
           ~callback:(public_toggled client cookie cache backends user_msg
                       ~model:backend#store
                       ~column:backend#col_public));
  let col = GTree.view_column ~title:(s_ "public")
      ~renderer:(renderer, ["active", backend#col_public]) () in
  col#set_sort_column_id backend#col_public.GTree.index;
  col#set_sort_indicator true;
  ignore(view#append_column col);
  (* id column *)
  let col = GTree.view_column ~title:(s_ "id")
      ~renderer:(GTree.cell_renderer_text [], ["text",
                                               backend#col_id]) () in
  col#set_sort_column_id backend#col_id.GTree.index;
  col#set_sort_indicator true;
  ignore(view#append_column col);
  (* descriptor column *)
  let col = GTree.view_column ~title:(s_ "descriptor")
      ~renderer:(GTree.cell_renderer_text [], ["text",
                                               backend#col_desc]) () in
  col#set_sort_column_id backend#col_desc.GTree.index;
  col#set_sort_indicator true;
  ignore(view#append_column col);

  view#set_rules_hint true;
  view#selection#set_mode `SINGLE;
  view#set_model (Some backend#store#coerce)
```

26.3 Window management

In all following code, `ui` denotes an object of class `Demexp_gladeui.classification`, `backends` contains the containers for the widget data and `user_msg` is a function that displays a user message on the window status bar.

Helper function `get_question_tags` get the list of identifiers and tag ids for question of identifier `q_id`. Inner function `tag_label` returns for a given `tag_id` its label.

Note: We get the label for each tag from the server and we are not relying of the list of tags as a hash table in the backend in case new tags have been added on the server.

135a ⟨*clsf.ml* 129⟩+≡ ◁134 135b▷

```
let get_question_tags user_msg client cookie cache q_id =
  let h = Hashtbl.create 3 in
  let f () =
    let tags =
      Array.to_list (Demexp.V1.get_question_tags client (cookie, q_id)) in
    let tag_label tag_id =
      let info = Cache.tag_info cache client (cookie, tag_id, 1) in
      if info.tag_info_rc <> rt_ok then (
        user_msg (Printf.sprintf (f_ "Cannot get label of tag %d: %s")
                    tag_id
                    (Misc.string_of_return_code info.tag_info_rc));
        "??"
       ) else
        info.tag_info.(0).a_tag_label in
    List.iter (fun tag_id -> Hashtbl.add h tag_id (tag_label tag_id)) tags;
    h in
  MiscUI.handle_network_error f () h
```

Callback `question_clicked` is called each time a row in question list is selected or unselected. When a new question is selected, we get its list of tags and update the corresponding widget backend.

135b ⟨*clsf.ml* 129⟩+≡ ◁135a 136▷

```
let question_clicked backends user_msg client cookie cache
    path currently_selected =
  let model : #GTree.model = backends.questions#store in
  let row = model#get_iter path in
  if not currently_selected then (
    let q_id = model#get ~row ~column:backends.questions#col_id in
    let tags = get_question_tags user_msg client cookie cache q_id in
    backends.question_tags#clear ();
    backends.question_tags#set_tags_hash tags;
    Tags.fill_tag_store backends.question_tags tags;
    backends.questions#set_selected_question (Some q_id)
   ) else (
    backends.question_tags#clear ();
    backends.questions#set_selected_question None
   );
  true (* allow selection state to change *)
```

Callback `remove_callback` is called when the user clicks on the button "Remove this tag from question" button. It gets the selected tag and removes it on the server. It does nothing if no question is selected.

136 ⟨*clsf.ml* 129⟩+≡ ◁135b 137▷

```
let remove_callback ui backends user_msg client cookie () =
  (* get current selected row in question_tags list *)
  let paths = ui#treeview_question_tags22#selection#get_selected_rows in
  let model : #GTree.model = backends.question_tags#store in
  match paths with
  | [] -> ()
  | path :: _ ->
      let iter = model#get_iter path in
      let tag_id = model#get ~row:iter ~column:backends.question_tags#col_id in
      match backends.questions#selected_question with
      | None -> ()
      | Some q_id ->
          let f () =
            let ret = Demexp.V1.untag_question client (cookie, q_id, tag_id) in
            if ret <> rt_ok then
              user_msg
                (Printf.sprintf
                   (f_ "unable to remove tag:%d from question:%d : %s")
                   tag_id q_id (Misc.string_of_return_code ret))
            else (
              let store : GTree.tree_store = backends.question_tags#store in
              ignore(store#remove iter);
              backends.question_tags#remove_tag tag_id;
              user_msg "" (* we erase any error message *)
             ) in
          MiscUI.handle_network_error f () ()
```

Callback `add_callback` is called when the user clicks on the button "Add tag to question" button. It gets the selected tag in the tag list and calls the server to add it to the selected question. The callback does nothing in case no question is selected or if the tag is already associated to the selected question.

137 ⟨*clsf.ml* 129⟩+≡ ◁136 138a▷

```
let add_callback ui backends user_msg client cookie () =
  (* get current selected row in tags list *)
  let paths = ui#treeview_tags14#selection#get_selected_rows in
  let model : #GTree.model = backends.tags#store in
  match paths with
  | [] -> ()
  | path :: _ ->
      let iter = model#get_iter path in
      let tag_id = model#get ~row:iter ~column:backends.tags#col_id in
      let tag_label = Hashtbl.find backends.tags#tags_hash tag_id in
      match backends.questions#selected_question with
      | None -> ()
      | Some q_id ->
          let f () =
            if not (Hashtbl.mem backends.question_tags#tags_hash tag_id) then
              let ret = Demexp.V1.tag_question client (cookie, q_id, tag_id) in
              if ret <> rt_ok then
                user_msg
                  (Printf.sprintf
                     (f_ "unable to add tag:%d to question:%d : %s")
                     tag_id q_id (Misc.string_of_return_code ret))
              else (
                Tags.add_tag_row backends.question_tags tag_id tag_label;
                backends.question_tags#add_tag tag_id tag_label;
                user_msg "" (* we erase any error message *)
               ) in
          MiscUI.handle_network_error f () ()
```

At first, helper function `setup_window` gets the data (list of tags and questions) from the server, then it connects the tree widgets to their backends and sets needed callbacks.

138a ⟨*clsf.ml* 129⟩+≡ ◁137 138b▷

```
let setup_window ui backends user_msg client cookie cache
    reload_tags_and_questions =
  (* load data from server *)
  let tags_hash = Hashtbl.create 3 in
  Cache.update_tags_hash tags_hash client cookie cache;
  backends.tags#set_tags_hash tags_hash;
  Tags.fill_tag_store backends.tags tags_hash;
  fill_question_store backends.questions client cookie cache;
  (* list of all tags *)
  Tags.connect_tag_view ~view:ui#treeview_tags14
    ~backend:backends.tags ();
  (* tags specific to a question *)
  Tags.connect_tag_view ~view:ui#treeview_question_tags22
    ~backend:backends.question_tags ();
  (* list of tagging_only questions *)
  connect_question_view client cookie cache backends user_msg
    ~view:ui#treeview_questions15
    ~backend:backends.questions;
  ui#treeview_questions15#selection#set_select_function
    (question_clicked backends user_msg client cookie cache);
  (* button callbacks *)
  ignore(ui#button_remove40#connect#clicked
           ~callback:(remove_callback ui backends user_msg client cookie));
  ignore(ui#button_add41#connect#clicked
           ~callback:(add_callback ui backends user_msg client cookie));
  (* reload the browser when done *)
  ignore(ui#toplevel#event#connect#delete
           ~callback:(fun _ev ->
             reload_tags_and_questions ();
             false (*close window*)))
```

Function `ui` is called to display the user management window. Parameter `reload_tags_and_questions` is defined in code chunk 161.

138b ⟨*clsf.ml* 129⟩+≡ ◁138a

```
let ui_classification client cookie cache reload_tags_and_questions () =
  let ui = new Demexp_gladeui.classification () in
  let user_msg =
    MiscUI.create_user_msg_fun ui#statusbar3 "classification sb" in
  (* we show dialog immediately so we can tell what we are doing in
     the status bar *)
  ui#toplevel#show ();
  try
    let backends = { tags = new tag_backend ();
                     question_tags = new tag_backend ();
                     questions = new question_backend (); } in
    setup_window ui backends user_msg client cookie cache
      reload_tags_and_questions
  with Failure msg -> user_msg msg
```

Chapter 27

“Add reponse” window (Addrep)

Module Addrep displays and manages the “Add reponse” window.

139 ⟨*addrep.ml* 139⟩≡ 140 ▷

```
(* copyright 2004-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Messages_aux
open Messages_clnt
open DemexpGettext.Gettext
```

Callback `add_callback` is called when the user clicks on the "OK" button. It simply checks that the user still agree and then calls the relevant RPC call on the server. In case of error, a message is displayed in calling window with `user_msg` function.

Note: We are forced to tell that `resp_buf` is of type `GText.buffer` otherwise OCaml is unable to compile it (optional parameters of the method are not seen properly).

140 ⟨*addrep.ml* 139⟩+≡ ◁139 141▷

```
let add_callback ui client cookie user_msg q_id (resp_buf : GText.buffer)
    update_callback () =
  let response = Norm.normalize_response (resp_buf#get_text ())
  and link = Norm.normalize_link ui#entry_link18#text in
  resp_buf#set_text response;
  ui#entry_link18#set_text link;
  try
    Norm.check_response response;
    try
      Norm.check_link link;
      let message =
        Printf.sprintf (f_ "Are you sure you want to add response: \"%s\" [%s]? You won't be ab
          response link in
      let confirm_addition () =
        let md = GWindow.message_dialog
            ~message
            ~message_type:`QUESTION
            ~buttons:GWindow.Buttons.ok_cancel
            ~modal:true () in
        let res = md#run () = `OK in
        md#destroy () ;
        res in
      let f () =
        if confirm_addition () then (
          let ret =
            Demexp.V1.add_response client (cookie, q_id, response, link) in
          if ret <> rt_ok then (
            let msg =
              Printf.sprintf (f_ "Error while adding a new reponse: %s")
                (Misc.string_of_return_code ret) in
            user_msg msg
           ) else (
            user_msg "Response added.";
            update_callback ()
           );
          ui#toplevel#destroy ()
         ) in
      MiscUI.handle_network_error f () ()
    with Norm.Invalid_format ->
      MiscUI.display_error (s_ "Invalid link format")
  with Norm.Invalid_format ->
    MiscUI.display_error (s_ "Invalid response format")
```

Function `ui_add_response` is used to open the window "Add reponse" that allows the user to add a new reponse to question `q_id`, with descriptor `q_desc`. Function `user_msg` is used to display messages to the user in calling window. Function `update_callback` is called to refresh the browser window once the response is added (cf. code chunk 158b).

141 ⟨*addrep.ml* 139⟩+≡ ◁140

```
let ui_add_response client cookie user_msg q_id q_desc update_callback =
  let title = Printf.sprintf (f_ "Add response to \"%s\"") q_desc in
  let ui = new Demexp_gladeui.add_response () in
  ui#toplevel#set_title title;
  let resp_buf = GText.buffer () in
  ui#textview_response4#set_buffer resp_buf;
  ignore(ui#button_add8#connect#clicked
           ~callback:(add_callback ui client cookie user_msg q_id resp_buf
                        update_callback));
  ignore(ui#button_cancel9#connect#clicked
           ~callback:ui#toplevel#destroy);
  ui#toplevel#show ()
```

Chapter 28

"Vote" window (`Vote`)

Module `Vote` allows the user to enter its preferences and to express them as a vote on the demexp server.

142a ⟨*vote.ml* 142a⟩≡ 142b ▷
```
(* copyright 2004-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Messages_aux
open Messages_clnt
open DemexpGettext.Gettext
```

28.1 List widget backends

We define a `response_backend` that stores the data needed for the two widgets displaying the available and selected responses of the vote. The backend contains two columns: the `id`entifier of the choice and its `desc`riptor.

142b ⟨*vote.ml* 142a⟩+≡ ◁ 142a 142c ▷
```
class response_backend () =
  let columns = new GTree.column_list in
  let col_id = columns#add Gobject.Data.int in
  let col_desc = columns#add Gobject.Data.string in
  let store = GTree.list_store columns in

  object
    method col_id = col_id
    method col_desc = col_desc
    method store = store
  end
```

For the radio buttons at the bottom of the window, we define a datatype to store the selected vote type (user and/or delegate).

142c ⟨*vote.ml* 142a⟩+≡ ◁ 142b 143a ▷
```
type vote_type =
  | User
  | Delegate
  | User_and_delegate
```

We define a structure to store the backends.

143a ⟨*vote.ml* 142a⟩+≡ ◁142c 143b▷

```
type backends = {
    available : response_backend;
    myvote : response_backend;
    mutable vote_type : vote_type;
  }
```

Helper function `add_row` adds a new response row with `id` and `desc` into `backend`.

143b ⟨*vote.ml* 142a⟩+≡ ◁143a 143c▷

```
let add_row backend (id, desc) =
  let store : GTree.list_store = backend#store in
  let iter = store#append () in
  store#set ~row:iter ~column:backend#col_id id;
  store#set ~row:iter ~column:backend#col_desc desc
```

Function `fill_backend` fills up a `backend` with a list of responses (each one being a couple (id, descriptor)).

Note: we do the line split here and not in `add_row` because `add_row` is used multiple times when user moves responses from/to his vote and responses would be splitted at each call of `add_row`.

143c ⟨*vote.ml* 142a⟩+≡ ◁143b 143d▷

```
let fill_backend backend responses =
  let add_splitted_row backend (id, desc) =
    let multiline_desc = Misc.add_line_splits 40 desc in
    add_row backend (id, multiline_desc) in
  List.iter (add_splitted_row backend) responses
```

28.2 List widget views

Function `connect_view` connect a widget list `view` to its `backend`. In this `view`, only one row can be selected at a time.

143d ⟨*vote.ml* 142a⟩+≡ ◁143c 144a▷

```
let connect_view ~view ~backend =
  (* id column *)
  let col = GTree.view_column ~title:(s_ "id")
      ~renderer:(GTree.cell_renderer_text [], ["text",
                                               backend#col_id]) () in
  ignore(view#append_column col);
  (* descriptor column *)
  let col = GTree.view_column ~title:(s_ "descriptor")
      ~renderer:(GTree.cell_renderer_text [], ["text",
                                               backend#col_desc]) () in
  ignore(view#append_column col);

  view#selection#set_mode `SINGLE;
  view#set_model (Some backend#store#coerce)
```

28.3 Window management

Function `transfer_row` transfer a row designated by `path` from the `src` backend to the `dst` backend.

144a ⟨*vote.ml* 142a⟩+≡ ◁143d 144b▷

```
let transfer_row ~src ~dst ~path =
  let src_model : #GTree.model = src#store in
  let row = src_model#get_iter path in
  let r_id = src_model#get ~row ~column:src#col_id in
  let r_desc =
    src_model#get ~row ~column:src#col_desc in
  ignore(src#store#remove row);
  add_row dst (r_id, r_desc)
```

Callback `add_callback` is called when the user clicks on "Add to my vote" button. It finds the selected response in the "Available responses" list and adds it to the "My vote" list.

144b ⟨*vote.ml* 142a⟩+≡ ◁144a 144c▷

```
let add_callback ui backends () =
  let sel = ui#treeview_available_responses6#selection#get_selected_rows in
  match sel with
  | [] -> ()
  | path :: _ -> (* only one row can be selected simultaneously *)
      transfer_row ~src:backends.available ~dst:backends.myvote ~path
```

Callback `remove_callback` is called when the user clicks on "Remove from my vote" button. It finds the selected response in the "My vote" responses list and adds it to the "Available responses" list.

144c ⟨*vote.ml* 142a⟩+≡ ◁144b 144d▷

```
let remove_callback ui backends () =
  let sel = ui#treeview_my_vote7#selection#get_selected_rows in
  match sel with
  | [] -> ()
  | path :: _ -> (* only one row can be selected simultaneously *)
      transfer_row ~src:backends.myvote ~dst:backends.available ~path
```

Callback `preferred_callback` is called when the user clicks on the "Preferred" button. It puts the selected response in the "My vote" responses list one row up.

144d ⟨*vote.ml* 142a⟩+≡ ◁144c 145a▷

```
let preferred_callback ui backends () =
  let sel = ui#treeview_my_vote7#selection#get_selected_rows in
  match sel with
  | [] -> ()
  | path :: _ -> (* only one row can be selected simultaneously *)
      let model : #GTree.model = backends.myvote#store in
      let row1 = model#get_iter path in
      ignore(GTree.Path.prev path);
      let row2 = model#get_iter path in
      ignore(backends.myvote#store#swap row1 row2)
```

Callback `disliked_callback` is called when the user clicks on the "Disliked" button. It puts the selected response in the "My vote" responses list one row down.

145a ⟨*vote.ml* 142a⟩+≡ ◁144d 145b▷

```
let disliked_callback ui backends () =
  let sel = ui#treeview_my_vote7#selection#get_selected_rows in
  match sel with
  | [] -> ()
  | path :: _ -> (* only one row can be selected simultaneously *)
      let model : #GTree.model = backends.myvote#store in
      let row1 = model#get_iter path in
      GTree.Path.next path;
      try
        let row2 = model#get_iter path in
        ignore(backends.myvote#store#swap row1 row2)
      with Failure "GtkTree.TreeModel.get_iter" -> () (* bottom of the list *)
```

Helper function `result_msg` returns, depending on whether the `user_ret` and `delegate_ret` return codes are a success or not, the appropriate user message.

145b ⟨*vote.ml* 142a⟩+≡ ◁145a 145c▷

```
let result_msg user_ret delegate_ret type_str =
  match user_ret, delegate_ret with
  | a, b when a = rt_ok && b = rt_ok ->
      Printf.sprintf (f_ "Vote as %s succeeded.") type_str
  | a, b when a = rt_ok && b <> rt_ok ->
      Printf.sprintf (f_ "Vote as %s failed partially. Delegate error: %s.")
        type_str
        (Misc.string_of_return_code delegate_ret)
  | a, b when a <> rt_ok && b = rt_ok ->
      Printf.sprintf (f_ "Vote as %s failed partially. User error: %s.")
        type_str
        (Misc.string_of_return_code user_ret)
  | _ ->
      Printf.sprintf
        (f_ "Vote as %s failed. User error: %s. Delegate error: %s.")
        type_str
        (Misc.string_of_return_code user_ret)
        (Misc.string_of_return_code delegate_ret)
```

Helper function `vote_as_delegate` allows to `client` to vote as a delegate defined in preferences `pref` on question of identifier `q_id`.

We do not check the return code of login because, in case of inability to login as a delegate, the `vote()` RPC will return an error code.

145c ⟨*vote.ml* 142a⟩+≡ ◁145b 145d▷

```
let vote_as_delegate client q_id pref vote =
  let login_ret =
    Demexp.V1.login client (Rtypes.int_of_uint4 protocol_version,
                            pref#delegate_login, pref#delegate_password) in
  let ret =
    Demexp.V1.vote client (login_ret.login_cookie, q_id, Array.of_list vote) in
  Demexp.V1.goodbye client login_ret.login_cookie;
  ret
```

Helper function `record_voted_question` stores in `clerk` that we have voted on question of identifier `q_id` if no error appeared when voting.

145d ⟨*vote.ml* 142a⟩+≡ ◁145c 146a▷

```
let record_voted_question clerk q_id user_ret delegate_ret =
  if user_ret = rt_ok && delegate_ret = rt_ok then
    Clerk.mark_as_voted ~clerk ~q_id
```

Callback `vote_callback` is called when the user clicks on the "Vote" button. It gets the vote choice from the "My vote" list widget and calls the needed RPCs on the server. In any case, the "Vote" window is destroyed and a message is displayed in the browser window.

146a ⟨*vote.ml* 142a⟩+≡ ◁145d 146b▷

```
let vote_callback ui backends client cookie user_msg q_id pref clerk
    update_callback () =
  let f () =
    let model : #GTree.model = backends.myvote#store in
    let vote = ref [] in
    let add_choice _ iter =
      let r_id = model#get ~row:iter ~column:backends.myvote#col_id in
      vote := r_id :: !vote;
      false (* do not stop, continue walking the store *) in
    backends.myvote#store#foreach add_choice;
    vote := List.rev !vote;
    let display_result_msg user_ret delegate_ret type_str =
      let msg = result_msg user_ret delegate_ret type_str in
      user_msg msg in
    (match backends.vote_type with
    | User ->
        let user_ret =
          Demexp.V1.vote client (cookie, q_id, Array.of_list !vote) in
        display_result_msg user_ret rt_ok (s_ "user");
        record_voted_question clerk q_id user_ret rt_ok
    | Delegate ->
        let delegate_ret = vote_as_delegate client q_id pref !vote in
        display_result_msg rt_ok delegate_ret (s_ "delegate");
        record_voted_question clerk q_id rt_ok delegate_ret
    | User_and_delegate ->
        let delegate_ret = vote_as_delegate client q_id pref !vote in
        let user_ret =
          Demexp.V1.vote client (cookie, q_id, Array.of_list !vote) in
        display_result_msg user_ret delegate_ret (s_ "user & delegate");
        record_voted_question clerk q_id user_ret delegate_ret);
    update_callback (); (* refresh browser display *)
    ui#toplevel#destroy () in
  MiscUI.handle_network_error f () ()
```

Function `get_question_details` returns the description and responses of question of identifier `q_id`.

146b ⟨*vote.ml* 142a⟩+≡ ◁146a 147▷

```
let get_question_details client cookie cache q_id =
  let ret = Cache.question_info cache client (cookie, q_id, 1) in
  if ret.question_info_rc <> rt_ok then
    raise (Misc.Display_error
             (Printf.sprintf
                (f_ "Unable to load information on question:%d : %s")
                q_id (Misc.string_of_return_code ret.question_info_rc)));
  if Array.length ret.question_info <> 1 then
    raise (Misc.Display_error
             (Printf.sprintf
                (f_ "Invalid array length for question_info:%d : %d")
                q_id (Array.length ret.question_info)));
  let q_desc = ret.question_info.(0).q_desc in
  let q_responses =
    Array.to_list (Array.mapi (fun i r -> (i, r.r_info_desc))
                     ret.question_info.(0).q_info_responses) in
  (q_desc, q_responses)
```

Function `setup_vote_backends` returns the content of the two backends corresponding to (i) my vote and (ii) the remaining choices. It simply splits the list `q_responses` into two lists, depending on whether each item is available in my vote or note.

147 ⟨*vote.ml* 142a⟩+≡ ◁146b 148▷

```
let setup_vote_backends client cookie user_msg pref q_id q_responses =
  let ret = Demexp.V1.get_vote client (cookie, q_id, pref#user_login) in
  if ret.get_vote_rc <> rt_ok then (
    user_msg (Printf.sprintf (f_ "Cannot get my own vote. Error: %s")
                (Misc.string_of_return_code ret.get_vote_rc));
    ([], q_responses)
   ) else (
    let my_vote = Array.to_list ret.get_vote in
    Misc.split_responses my_vote q_responses
   )
```

Function `ui_vote` is used to display the "Vote" window for question of identifier `q_id` and descriptor `q_desc`. The list of available responses is given in `q_responses`. The function `update_callback` is called to refresh the main browser window when the vote is registered. Function `user_msg` is used to display a message in main browser's status bar. Object `pref` contains user preferences.

148 ⟨*vote.ml* 142a⟩+≡ ◁147

```
let ui_vote client cookie user_msg q_id update_callback pref cache clerk =
  let q_desc, q_responses = get_question_details client cookie cache q_id in
  let title = Printf.sprintf (f_ "Vote on \"%s\" question") q_desc in
  let ui = new Demexp_gladeui.vote () in
  (* display question descriptor *)
  ui#toplevel#set_title title;
  let q_desc_buf = GText.buffer () in
  q_desc_buf#set_text q_desc;
  ui#textview_question1#set_buffer q_desc_buf;
  (* prepare backends and connect them to views *)
  let backends = { available = new response_backend ();
                   myvote = new response_backend ();
                   vote_type = User; } in
  let my_vote, other_responses =
    setup_vote_backends client cookie user_msg pref q_id q_responses in
  fill_backend backends.available other_responses;
  fill_backend backends.myvote my_vote;
  connect_view ~view:ui#treeview_available_responses6
    ~backend:backends.available;
  connect_view ~view:ui#treeview_my_vote7 ~backend:backends.myvote;
  (* setup callbacks *)
  ignore(ui#button_cancel17#connect#clicked
           ~callback:ui#toplevel#destroy);
  ignore(ui#button_add14#connect#clicked
           ~callback:(add_callback ui backends));
  ignore(ui#button_remove15#connect#clicked
           ~callback:(remove_callback ui backends));
  ignore(ui#button_preferred36#connect#clicked
           ~callback:(preferred_callback ui backends));
  ignore(ui#button_disliked37#connect#clicked
           ~callback:(disliked_callback ui backends));
  ignore(ui#button_vote16#connect#clicked
           ~callback:(vote_callback ui backends client cookie user_msg
                        q_id pref clerk update_callback));
  ignore(ui#radiobutton_as_user1#connect#clicked
           ~callback:(fun () -> backends.vote_type <- User));
  ignore(ui#radiobutton_as_delegate2#connect#clicked
           ~callback:(fun () -> backends.vote_type <- Delegate));
  ignore(ui#radiobutton_as_both3#connect#clicked
           ~callback:(fun () -> backends.vote_type <- User_and_delegate));
  ui#toplevel#show ()
```

Chapter 29

"demexp" window (`Browser`)

Module `Browser` displays the main demexp client window that allows to navigate within the demexp base.

149a ⟨*browser.ml* 149a⟩≡ 149b ▷
```
(* copyright 2004-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Messages_aux
open Messages_clnt
open DemexpGettext.Gettext
```

29.1 About dialog

Callback `about_dialog` display the "About demexp" dialog.

149b ⟨*browser.ml* 149a⟩+≡ ◁ 149a 149c ▷
```
let about_dialog () =
  let msg =
    Printf.sprintf
      (f_ ("Demexp: a client for the democratic experience server.\n  Version: %s\n  Protocol v
      Config.client_version
      (string_of_int (Rtypes.int_of_uint4 protocol_version)) in
  MiscUI.display_message msg
```

29.2 Set of integers

We define a module `IntSet` to manipulate set of integers.

149c ⟨*browser.ml* 149a⟩+≡ ◁ 149b 150a ▷
```
module OrderedInt =
  struct
    type t = int
    let compare = compare
  end

module IntSet = Set.Make(OrderedInt)
```

29.3 Window backends

We define in this section the two backends that stores the data needed to display the two tree widgets of the demexp window. For this, we define two kinds of backends: `tag_backend` for the two tree widgets showing tags, and `question_backend` for the tree widget displaying the set of available questions.

A `tag_backend` contains two columns: the id and the label of each tag. It also contains a hash table `tag_hash` of all stored tags. Associated methods `add_tag` and `remove_tag` allows to manipulate this hash table. Method `clear` removes all entries in the backend. `tag_backend` also contains a set of currently selected tag identifiers.

150a ⟨*browser.ml* 149a⟩+≡ ◁149c 150b▷

```
class tag_backend () =
  let columns = new GTree.column_list in
  let col_id = columns#add Gobject.Data.int in
  let col_new = columns#add Gobject.Data.boolean in
  let col_voted = columns#add Gobject.Data.boolean in
  let col_label = columns#add Gobject.Data.string in
  let store = GTree.tree_store columns in

  let _ = store#set_sort_column_id col_label.GTree.index 'ASCENDING in

  object
    method col_id = col_id
    method col_new = col_new
    method col_voted = col_voted
    method col_label = col_label
    method store = store

    val mutable tag_hash : (int, string) Hashtbl.t = Hashtbl.create 3
    method tag_hash = tag_hash
    method set_tag_hash h = tag_hash <- h
    method add_tag id label = Hashtbl.add tag_hash id label
    method remove_tag id = Hashtbl.remove tag_hash id

    method clear () =
      Hashtbl.clear tag_hash;
      store#clear ()

    val mutable selected_tags = IntSet.empty
    method selected_tags = IntSet.elements selected_tags
    method add_selected_tag tag_id =
      selected_tags <- IntSet.add tag_id selected_tags
    method remove_selected_tag tag_id =
      selected_tags <- IntSet.remove tag_id selected_tags
  end
```

Helper functions `is_question_(new|voted)` return true if the corresponding question of identifier `q_id` is considered as new or voted by the user.

- New: the question has never been seen or has been updated on the server;
- Voted: the user has voted on the question.

150b ⟨*browser.ml* 149a⟩+≡ ◁150a 151a▷

```
let is_question_new clerk q_id = not (Clerk.seen ~clerk ~q_id)

let is_question_voted clerk q_id = Clerk.voted ~clerk ~q_id
```

A `question_backend` contains two columns: the identifier and the descriptor of each question.

151a ⟨*browser.ml* 149a⟩+≡ ◁150b 151b▷

```
class question_backend () =
  let columns = new GTree.column_list in
  let col_id = columns#add Gobject.Data.int in
  let col_new = columns#add Gobject.Data.boolean in
  let col_voted = columns#add Gobject.Data.boolean in
  let col_desc = columns#add Gobject.Data.string in
  let store = GTree.list_store columns in

  let _ = store#set_sort_column_id col_id.GTree.index 'ASCENDING in

  object
    method col_id = col_id
    method col_new = col_new
    method col_voted = col_voted
    method col_desc = col_desc
    method store = store

    method update_row_state ~clerk ~row ~q_id =
      store#set ~row ~column:col_new (is_question_new clerk q_id);
      store#set ~row ~column:col_voted (is_question_voted clerk q_id)
  end
```

29.4 Browser context

We define a `context` of a browser that contains all needed state for a browser connected to a given server.

151b ⟨*browser.ml* 149a⟩+≡ ◁151a 152a▷

```
type context = {
    client : Rpc_client.t;
    cookie : Messages_aux.cookie_t;
    pref : Pref.preferences;
    tag_backend : tag_backend;
    question_backend : question_backend;
    cache : Cache.t;
    clerk : Clerk.t;
    ui : Demexp_gladeui.demexp;
  }
```

This context is given to all following functions through the variable `ctx`.

29.5 Display of relevant questions in question list

When the user selects a set of tags, only the questions having those tags are displayed. This is done by maintaining a hash table indexed by tag identifiers and associating, to each tag id, the set of all questions containing this tag. When a tag is selected or unselected, the set of valid question to display is recomputed.

Function `update_questions` updates the `h` hash table with the set of all public questions of the server.

152a ⟨*browser.ml* 149a⟩+≡ ◁151b 152b▷

```
let update_questions h ctx user_msg =
  (* get maximum question number *)
  let ret = Demexp.V1.max_question_id ctx.client ctx.cookie in
  if ret.max_question_id_rc <> rt_ok then
    raise (Misc.Display_error
             (Printf.sprintf "unable to get max_question_id (%s)"
                (Misc.string_of_return_code ret.max_question_id_rc)));
  (* open a progress bar and needed callback *)
  let pb = MiscUI.open_progress_bar ~max:ret.max_question_id
      ~title:(s_ "Update question classification") () in
  let update_cb base = MiscUI.update_progress_bar ~pb ~v:base in
  (* do actual update (with possible network access) *)
  let f () =
    Cache.update_questions_hash ~update_cb h ret.max_question_id
      ctx.client ctx.cookie ctx.cache in
  MiscUI.handle_network_error f () ();
  MiscUI.close_progress_bar ~pb
```

Function `update_question_tag_indexed` computes, from a `question_hash` containing for each question id the question descriptor and the list of its tags, a hash indexed by tag ids and associating to each tag the set of question ids having it. The result is put into the hash table `h` given as parameter.

152b ⟨*browser.ml* 149a⟩+≡ ◁152a 153a▷

```
let update_question_tag_indexed h question_hash =
  Hashtbl.clear h;
  let add_q_to_tag q_id tag_id =
    if Hashtbl.mem h tag_id then (
      let q_ids = Hashtbl.find h tag_id in
      Hashtbl.replace h tag_id (IntSet.add q_id q_ids)
     ) else
      Hashtbl.add h tag_id (IntSet.singleton q_id) in
  let add_question q_id (_, tag_id_list) =
    List.iter (add_q_to_tag q_id) tag_id_list in
  Hashtbl.iter add_question question_hash
```

Function `compute_selected_questions` computes the list of question identifiers that is common to all tag ids in `tag_id_list` (i.e. the intersection of all question ids associated to each one of the selected tags).

153a ⟨*browser.ml* 149a⟩+≡ ◁152b 153b▷

```
let compute_selected_questions question_tag_indexed tag_id_list =
  try
    let rec compute_remaining tag_id_list q_set =
      match tag_id_list with
      | [] -> IntSet.elements q_set
      | tag_id :: tail ->
          let tag_id_q_set = Hashtbl.find question_tag_indexed tag_id in
          let new_q_set = IntSet.inter q_set tag_id_q_set in
          compute_remaining tail new_q_set in
    match tag_id_list with
    | [] -> []
    | tag_id :: tail ->
        compute_remaining tail (Hashtbl.find question_tag_indexed tag_id)
  with Not_found -> [] (* no question associated to a tag_id, so resulting
                          list is empty (intersection with empty set) *)
```

29.6 Views setup

In this section, we prepare the views of the two tree widgets: tags and questions.

29.6.1 Tag views

All the functions related to tag views are defined in section 24.1.

29.6.2 Question view

Helper function `add_question_row` add a new row which contains `id` and `desc` to given question `backend`.

153b ⟨*browser.ml* 149a⟩+≡ ◁153a 153c▷

```
let add_question_row ctx (q_id, desc) =
  let backend = ctx.question_backend in
  let store : GTree.list_store = backend#store in
  let iter = store#append () in
  store#set ~row:iter ~column:backend#col_id q_id;
  store#set ~row:iter ~column:backend#col_desc (Misc.add_line_splits 50 desc);
  backend#update_row_state ~clerk:ctx.clerk ~row:iter ~q_id
```

Function `fill_question_store` fill up the backend of question list widget with the list of question identifiers `q_id_list`. To do this operation, the `questions` hash table containing for each question id the descriptor and the set of tags is also given as parameter.

153c ⟨*browser.ml* 149a⟩+≡ ◁153b 154a▷

```
let fill_question_store ctx questions q_id_list =
  (* get a descriptor for each question *)
  let id_to_id_desc id =
    let desc, _ = Hashtbl.find questions id in
    (id, desc) in
  let q_id_desc_list = List.map id_to_id_desc q_id_list in
  (* fill the store *)
  ctx.question_backend#store#clear ();
  List.iter (add_question_row ctx) q_id_desc_list
```

Function `connect_question_view` connects the view widget to the corresponding backend.

154a ⟨*browser.ml* 149a⟩+≡ ◁153c 154b▷

```
let connect_question_view ~view ~backend =
  let add_text_column ~title ~backend_col =
    let col = GTree.view_column ~title
        ~renderer:(GTree.cell_renderer_text [], ["text",
                                                 backend_col]) () in
    col#set_sort_column_id backend_col.GTree.index;
    col#set_sort_indicator true;
    ignore(view#append_column col) in
  let add_toggle_column ~title ~backend_col =
    let renderer = GTree.cell_renderer_toggle [] in
    let col = GTree.view_column ~title
        ~renderer:(renderer, ["active", backend_col]) () in
    col#set_sort_column_id backend_col.GTree.index;
    col#set_sort_indicator true;
    ignore(view#append_column col) in
  (* id column *)
  add_text_column ~title:(s_ "id") ~backend_col:backend#col_id;
  (* new column *)
  add_toggle_column ~title:(s_ "N") ~backend_col:backend#col_new;
  (* voted column *)
  add_toggle_column ~title:(s_ "V") ~backend_col:backend#col_voted;
  (* descriptor column *)
  add_text_column ~title:(s_ "question") ~backend_col:backend#col_desc;

  view#selection#set_mode `SINGLE;
  view#set_model (Some backend#store#coerce)
```

29.7 Window management

In the following code three hash tables, `tags`, `questions` and `question_tag_indexed` are used to store respectively the set of tags (indexed by their ids), the set of questions (indexed by their ids) and the set of all questions corresponding to a given tag id. Those hashes are given as parameter to following functions when needed. The use of hashes allows to reload them from the server (by calling `reload_tags_and_questions`, see code chunk 161) when the classification is modified.

Callback `quit_callback` is called when the application is closed. It is connected to demexp main window delete signal and the Quit menu item (cf. code chunk 159a).

154b ⟨*browser.ml* 149a⟩+≡ ◁154a 155▷

```
let quit_callback ev =
  GMain.Main.quit ();
  false
```

Function `update_question_display` updates on the screen the information on question of identifier `q_id` by fetching this information from the server.

155 ⟨*browser.ml* 149a⟩+≡ ◁154b 156a▷

```
let update_question_display ctx user_msg tags questions q_id =
  let f () =
    let timer = Perf.timer_start () in
    let ret = Cache.question_info ctx.cache ctx.client (ctx.cookie, q_id, 1) in
    if ret.question_info_rc <> rt_ok then
      user_msg (Printf.sprintf
                  (f_ "Unable to load information on question:%d : %s")
                  q_id (Misc.string_of_return_code ret.question_info_rc))
    else if Array.length ret.question_info <> 1 then
      user_msg (Printf.sprintf
                  (f_ "Invalid array length for question_info:%d : %d")
                  q_id (Array.length ret.question_info))
    else (
      (* question descriptor *)
      let q_desc_buf = GText.buffer () in
      q_desc_buf#set_text ret.question_info.(0).q_desc;
      ctx.ui#textview_q_desc4#set_buffer q_desc_buf;
      (* limit date *)
      let limit_date =
        match ret.question_info.(0).q_info_limit_date with
        | x when x = Int64.zero -> (s_ "no limit date")
        | x -> (* transform limit date in local time *)
            let offset = Int64.to_float x in
            Time.time_as_localtime_iso_string offset in
      ctx.ui#label_q_limitdate84#set_text limit_date;
      (* tags *)
      let _, q_tags = Hashtbl.find questions q_id in
      let q_label_tags = List.map (fun id -> Hashtbl.find tags id) q_tags in
      let str = List.fold_left (fun str e -> str ^ e ^ "\n") "" q_label_tags in
      let q_tags_buf = GText.buffer () in
      q_tags_buf#set_text str;
      ctx.ui#textview_q_tags5#set_buffer q_tags_buf;
      (* responses *)
      let string_of_response i r =
        let link =
          if r.r_info_link <> "" then ("[" ^ r.r_info_link ^ "]")
          else "" in
        Printf.sprintf "%d. %s %s\n" i r.r_info_desc link in
      let str_responses =
        Array.mapi string_of_response ret.question_info.(0).q_info_responses in
      let str = Array.fold_left (fun str e -> str ^ e) "" str_responses in
      let q_resp_buf = GText.buffer () in
      q_resp_buf#set_text str;
      ctx.ui#textview_responses4#set_buffer q_resp_buf;
      (* number of votes *)
      ctx.ui#label_q_num_votes105#set_text
        (string_of_int ret.question_info.(0).q_info_num_votes);
      (* winning response(s) *)
      let response_desc r_id =
        let desc = ret.question_info.(0).q_info_responses.(r_id).r_info_desc in
        Printf.sprintf "%d. %s" r_id desc in
      let str =
        Array.fold_left (fun str r_id -> str ^ (response_desc r_id) ^ ". ") ""
          ret.question_info.(0).q_info_elected_responses in
      let q_winning_buf = GText.buffer () in
```

```
        q_winning_buf#set_text str;
        ctx.ui#textview_winning_response4#set_buffer q_winning_buf;
        Perf.timer_stop_and_record "Browser.update_question_display" timer
       ) in
    MiscUI.handle_network_error f () ()
```

Function `erase_question_display` erases all the right side panel.

156a ⟨*browser.ml* 149a⟩+≡ ◁155 156b▷

```
  let erase_question_display ui =
    let empty_buf = GText.buffer () in
    empty_buf#set_text "";
    ui#textview_q_desc4#set_buffer empty_buf;
    ui#label_q_limitdate84#set_text "";
    ui#textview_q_tags5#set_buffer empty_buf;
    ui#textview_responses4#set_buffer empty_buf;
    ui#textview_winning_response4#set_buffer empty_buf
```

Helper function `get_selected_question_model_row` returns the `model` and `row` pointing to the selected question row, or `None` otherwise.

156b ⟨*browser.ml* 149a⟩+≡ ◁156a 156c▷

```
  let get_selected_question_model_row ctx =
      let selected_rows =
      ctx.ui#treeview_questions19#selection#get_selected_rows in
    match selected_rows with
    | [] -> None
    | [ path ] ->
        let model = ctx.question_backend#store in
        let row = model#get_iter path in
        Some (model, row)
    | _ -> (* cannot happen, only one row selected at once *)
        failwith "Browser.get_selected_question_model_row: should never happen"
```

Function `get_selected_question` returns the question identifier and description of a selected row, otherwise `None`.

156c ⟨*browser.ml* 149a⟩+≡ ◁156b 156d▷

```
  let get_selected_question ctx =
    match get_selected_question_model_row ctx with
    | None -> None
    | Some (model, row) ->
        let q_id = model#get ~row ~column:ctx.question_backend#col_id in
        let q_desc = model#get ~row ~column:ctx.question_backend#col_desc in
        Some (q_id, q_desc)
```

Function `update_selected_question_row` updates the display of the selected question row.

156d ⟨*browser.ml* 149a⟩+≡ ◁156c 157a▷

```
  let update_selected_question_row ctx =
    match get_selected_question_model_row ctx with
    | None -> ()
    | Some (model, row) ->
        let q_id = model#get ~row ~column:ctx.question_backend#col_id in
        ctx.question_backend#update_row_state ~clerk:ctx.clerk ~row ~q_id
```

Callback `modify_question_selection` is called each time a question is selected or unselected in the list of questions.

fixme: This callback seems to be called twice each time a new row is selected. I don't know why. FIXME

157a ⟨*browser.ml* 149a⟩+≡ ◁156d 157b▷

```
let modify_question_selection ctx user_msg tags questions
    path currently_selected =
  let model : #GTree.model = ctx.question_backend#store in
  let row = model#get_iter path in
  let q_id = model#get ~row ~column:ctx.question_backend#col_id in
  if not currently_selected then (
    update_question_display ctx user_msg tags questions q_id;
    Clerk.mark_as_seen ~clerk:ctx.clerk ~q_id;
    update_selected_question_row ctx
   ) else (
    erase_question_display ctx.ui
   );
  true (* allow selection state to change *)
```

Function `get_selected_tags_model_rows` return the list of `rows` selected in the tag view.

157b ⟨*browser.ml* 149a⟩+≡ ◁157a 157c▷

```
let get_selected_tags_rows ctx =
  let selected_rows =
    ctx.ui#treeview_tags18#selection#get_selected_rows in
  let model = ctx.tag_backend#store in
  List.map model#get_iter selected_rows
```

Function `update_tag_row` updates the boxes New, Voted and Updated from the status of questions associated to tag `row`.

157c ⟨*browser.ml* 149a⟩+≡ ◁157b 157d▷

```
let update_tag_row ctx question_tag_indexed row =
  let model = ctx.tag_backend#store in
  let tag_id = model#get ~row ~column:ctx.tag_backend#col_id in
  let q_ids =
    try
      IntSet.elements (Hashtbl.find question_tag_indexed tag_id)
    with Not_found -> [] in
  (* Seen box *)
  let tag_new = List.exists (is_question_new ctx.clerk) q_ids in
  model#set ~row ~column:ctx.tag_backend#col_new tag_new;
  (* Voted box *)
  let tag_voted = List.exists (is_question_voted ctx.clerk) q_ids in
  model#set ~row ~column:ctx.tag_backend#col_voted tag_voted
```

Function `update_selected_tag_rows` updates the boxes New, Voted and Updated from the status of associated questions to each tag.

157d ⟨*browser.ml* 149a⟩+≡ ◁157c 158a▷

```
let update_selected_tag_rows ctx question_tag_indexed =
  let rows = get_selected_tags_rows ctx in
  List.iter (update_tag_row ctx question_tag_indexed) rows
```

Callback `modify_tag_selection` is called each time a tag is selected or unselected in the list of tags. It updates the set of selected tags. It then computes the set of selected questions and display it in question list.

158a ⟨*browser.ml* 149a⟩+≡ ◁157d 158b▷

```
let modify_tag_selection ctx questions question_tag_indexed
    path currently_selected =
  let model : #GTree.model = ctx.tag_backend#store in
  let row = model#get_iter path in
  let tag_id = model#get ~row ~column:ctx.tag_backend#col_id in
  if currently_selected then
    ctx.tag_backend#remove_selected_tag tag_id
  else
    ctx.tag_backend#add_selected_tag tag_id;
  update_selected_tag_rows ctx question_tag_indexed;
  let tag_id_list = ctx.tag_backend#selected_tags in
  let selected_questions =
    compute_selected_questions question_tag_indexed tag_id_list in
  fill_question_store ctx questions selected_questions;
  erase_question_display ctx.ui;
  true (* allow selection state to change *)
```

Callback `add_reponse_callback` is called when the user clicks on the "Add response" button. It simply gets the selected question identifier and descriptor and calls the relevant window.

158b ⟨*browser.ml* 149a⟩+≡ ◁158a 158c▷

```
let add_reponse_callback ctx user_msg tags questions () =
  match get_selected_question ctx with
  | None -> ()
  | Some (q_id, q_desc) ->
      let update_callback () =
        Cache.invalidate ctx.cache (Cache.Question q_id);
        update_question_display ctx user_msg tags questions q_id;
        update_selected_question_row ctx in
      Addrep.ui_add_response ctx.client ctx.cookie user_msg
        q_id q_desc update_callback
```

Callback `vote_callback` is called when the user clicks on the "Vote" button. It simply gets the selected question identifier and descriptor and calls the relevant window.

158c ⟨*browser.ml* 149a⟩+≡ ◁158b 159a▷

```
let vote_callback ctx user_msg tags questions () =
  match get_selected_question ctx with
  | None -> ()
  | Some (q_id, q_desc) ->
      let update_callback () =
        Cache.invalidate ctx.cache (Cache.Question q_id);
        update_question_display ctx user_msg tags questions q_id;
        update_selected_question_row ctx in
      Vote.ui_vote ctx.client ctx.cookie user_msg
        q_id update_callback ctx.pref ctx.cache ctx.clerk
```

In function `setup_first_half_window`, we firstly open the main window and attach to it the minimal callbacks to be able to display messages in the status box and to quit properly the application.

The `delete` signal of `ui#toplevel` window is sent when the user closes a window.

fixme: We should initially disable menu entries without call backs. FIXME

159a ⟨*browser.ml* 149a⟩+≡ ◁158c 159b▷

```
let setup_window_1_over_3 ui =
  (* to stop application when closing demexp window or choosing menu option *)
  ignore(ui#toplevel#event#connect#delete ~callback:quit_callback);
  ignore(ui#quit1#connect#activate
           ~callback:(fun () -> ignore (quit_callback ())) );
  ignore(ui#about1#connect#activate ~callback:about_dialog);
  (* show main window, so we can display things on its status bar *)
  ui#toplevel#show ()
```

We then connect the "Preferences..." menu item.

159b ⟨*browser.ml* 149a⟩+≡ ◁159a 159c▷

```
let setup_window_2_over_3 ui pref connection_param =
  ignore(ui#preferences1#connect#activate
           ~callback:(Pref.ui_preferences pref));
  let title =
    match connection_param with
    | Some _ -> (* successful connection (hopefully login is correct, as
                   we cannot detect the case when we are connected as
                   Anonymous) *)
        Printf.sprintf (f_ "demexp (%s, %s) - %s:%d")
          pref#user_login
          pref#delegate_login
          pref#server_name
          pref#server_port
    | None -> (* connection failed *)
        Printf.sprintf (f_ "Connection failed to %s:%d")
          pref#server_name
          pref#server_port in
  ui#toplevel#set_title title
```

Helper function `select_question` search over all questions in the GtkTreeView of questions the one which has the same identifier as `q_id`.

159c ⟨*browser.ml* 149a⟩+≡ ◁159b 160a▷

```
let rec select_question ctx selection q_id iter =
  let current_q_id =
    ctx.question_backend#store#get ~row:iter
      ~column:ctx.question_backend#col_id in
  if current_q_id = q_id then (
    (* we have found our question *)
    selection#select_iter iter;
   ) else (
    if ctx.question_backend#store#iter_next iter then
      select_question ctx selection q_id iter
   )
```

Helper function `select_tags` search over all tags in the GtkTreeView of tags those which are included in `q_tags` and select them.

160a ⟨*browser.ml* 149a⟩+≡ ◁159c 160b▷

```
let rec select_tags ctx selection q_tags iter =
  match q_tags with
  | [] -> ()
  | _ ->
      let tag_id =
        ctx.tag_backend#store#get ~row:iter ~column:ctx.tag_backend#col_id in
      if List.exists (fun x -> x = tag_id) q_tags then (
        (* we have found a tag in our list *)
        selection#select_iter iter;
        let new_q_tags = List.filter (fun x -> x <> tag_id) q_tags in
        if ctx.tag_backend#store#iter_next iter then
          select_tags ctx selection new_q_tags iter
       ) else (
        if ctx.tag_backend#store#iter_next iter then
          select_tags ctx selection q_tags iter
       )
```

Helper function `open_on_question` is used to open the browser on a selected question, of identifier `q_id`.

160b ⟨*browser.ml* 149a⟩+≡ ◁160a 160c▷

```
let open_on_question ctx user_msg tags questions q_id =
  try
    (* update tags panel *)
    let _, q_tags = Hashtbl.find questions q_id in
    let tag_model : #GTree.model = ctx.tag_backend#store in
    ctx.ui#treeview_tags18#selection#unselect_all (); (* erase previous
                                                         selection *)
    let tag_iter = tag_model#get_iter_first in
    (match tag_iter with
    | None ->
        failwith "(open_on_question)no first tag iter should never happen"
    | Some iter ->
        select_tags ctx ctx.ui#treeview_tags18#selection q_tags iter);
    (* update question list panel *)
    let question_model : #GTree.model = ctx.question_backend#store in
    ctx.ui#treeview_questions19#selection#unselect_all (); (* erase previous
                                                              selection *)
    let question_iter = question_model#get_iter_first in
    (match question_iter with
    | None ->
        failwith "(open_on_question)no first question iter should never happen"
    | Some iter ->
        select_question ctx ctx.ui#treeview_questions19#selection q_id iter);
    (* update question detail panel *)
    update_question_display ctx user_msg tags questions q_id
  with Not_found ->
    user_msg (Printf.sprintf (f_ "question %d not found") q_id)
```

We define a type `browser_action` that describes the different kind of action we would like to do when opening a new browser.

160c ⟨*browser.ml* 149a⟩+≡ ◁160b 161▷

```
type action =
  | Nothing
  | Browse of int (* question_id *)
  | Vote of int (* question_id *)
```

Once the connection to the server is made, the callbacks to other menus, tree widget selections and buttons are added.

fixme: As we attache cache and clerk save functions on the callback of close event of each browser, other browsers do not save their state when we quit the application with several browsers opened. FIXME

161 ⟨*browser.ml* 149a⟩+≡ ◁160c 162▷

```
let setup_window_3_over_3 ctx user_msg action =
  let ui = ctx.ui
  and cookie = ctx.cookie
  and client = ctx.client in
  (* prepare a function to get data from server *)
  let questions = Hashtbl.create 3 in
  let tags = Hashtbl.create 3 in
  ctx.tag_backend#set_tag_hash tags;
  let question_tag_indexed = Hashtbl.create 3 in
  let reload_tags_and_questions () =
    user_msg (s_ "Get list of questions...");
    update_questions questions ctx user_msg;
    user_msg (s_ "Compute question indexed by tags...");
    update_question_tag_indexed question_tag_indexed questions;
    (* get all tags, update corresponding backend and connect it to view *)
    Cache.update_tags_hash tags ctx.client ctx.cookie ctx.cache;
    ui#treeview_tags18#selection#unselect_all ();
    let row_callback row = update_tag_row ctx question_tag_indexed row in
    Tags.fill_tag_store ~skip_question_specific_tags:true ~row_callback
      ctx.tag_backend tags;
    user_msg (s_ "List of questions and tags reloaded.") in
  reload_tags_and_questions ();
  (* hooking of menu callbacks needing connection parameters *)
  ignore(ui#manage_user1#connect#activate
           ~callback:(Users.ui_manage_users client cookie ctx.cache));
  ignore(ui#manage_tags1#connect#activate
           ~callback:(Tags.ui_manage_tags client cookie ctx.cache));
  ignore(ui#new_question1#connect#activate
           ~callback:(Newquestion.ui_new_question client cookie));
  ignore(ui#classification1#connect#activate
           ~callback:(Clsf.ui_classification client cookie ctx.cache
                        reload_tags_and_questions));
  ignore(ui#reload1#connect#activate ~callback:reload_tags_and_questions);
  (* connect to tag view *)
  Tags.connect_tag_view
    ~for_browser:true ~mode:`MULTIPLE ~view:ui#treeview_tags18
    ~backend:ctx.tag_backend ();
  ui#treeview_tags18#selection#set_select_function
    (modify_tag_selection ctx questions question_tag_indexed);
  (* connect question view *)
  connect_question_view ~view:ui#treeview_questions19
    ~backend:ctx.question_backend;
  ui#treeview_questions19#selection#set_select_function
    (modify_question_selection ctx user_msg tags questions);
  (* "add response" button *)
  ignore(ui#button_add_response44#connect#clicked
           ~callback:(add_reponse_callback ctx user_msg tags questions));
  (* "vote" button *)
  ignore(ui#button_vote45#connect#clicked
           ~callback:(vote_callback ctx user_msg tags questions));
  (* disable unused menu items and buttons *)
  ui#button_delegate46#misc#set_sensitive false;
```

```
ui#imagemenuitem_cut6#misc#set_sensitive false;
ui#imagemenuitem_copy7#misc#set_sensitive false;
ui#imagemenuitem_paste8#misc#set_sensitive false;
ui#imagemenuitem_delete9#misc#set_sensitive false;
ui#new1#misc#set_sensitive false;
(* as we have a cache, dump cache to disk when quitting *)
let cache_cb _ev = Cache.save ctx.cache; false in
ignore(ui#toplevel#event#connect#delete ~callback:cache_cb);
ignore(ui#quit1#connect#activate
         ~callback:(fun () -> ignore (cache_cb ())) );
(* and dump to disk on exit for the clerk *)
let clerk_cb _ev = Clerk.save ctx.clerk; false in
ignore(ui#toplevel#event#connect#delete ~callback:clerk_cb);
ignore(ui#quit1#connect#activate
         ~callback:(fun () -> ignore (clerk_cb ())) );
(* open on a given question if requested *)
match action with
| Nothing -> ()
| Browse q_id ->
    open_on_question ctx user_msg tags questions q_id
| Vote q_id ->
    open_on_question ctx user_msg tags questions q_id;
    vote_callback ctx user_msg tags questions ()
```

The function `ui_demexp` setup and starts the main demexp browser window. It then connects to the server and if no error it setup all remaining callbacks. It then execute requested `action` (like to browse or vote on a specific question). It raises `Connection_error` in case the connection to server cannot be established.

162 ⟨*browser.ml* 149a⟩+≡ ◁161 163▷

```
let ui_demexp pref connection_param action =
  (* show browser window *)
  let ui = new Demexp_gladeui.demexp () in
  let user_msg = MiscUI.create_user_msg_fun ui#statusbar1 "demexp sb" in
  setup_window_1_over_3 ui;
  setup_window_2_over_3 ui pref connection_param;
  match connection_param with
  | None -> () (* connection failed, do nothing *)
  | Some (client, cookie) ->
      (* setup remaining part of demexp window *)
      let cache_filename =
        Printf.sprintf "%s/cache-%s-%d"
          pref#preference_dir_name pref#server_name pref#server_port in
      let cache = Cache.create cache_filename client cookie in
      let clerk = Clerk.create cache pref pref#server_name pref#server_port in
      Clerk.update_seen_questions_from_cache ~clerk;
      if !Clntflags.flag_update_voted_state then
        Clerk.determine_voted_state ~clerk ~client ~cookie;
      let ctx = { client = client;
                  cookie = cookie;
                  pref = pref;
                  tag_backend = new tag_backend ();
                  question_backend = new question_backend ();
                  cache = cache;
                  clerk = clerk;
                  ui = ui; } in
      setup_window_3_over_3 ctx user_msg action
```

29.8 Autotests

163 ⟨*browser.ml* 149a⟩+≡ ◁162

```
  let _ =
    if Config.do_autotests then begin
      Printf.printf "  browser autotests...";
      let q_hash = Hashtbl.create 3 in
      Hashtbl.add q_hash 0 ("id0", [0; 3]);
      Hashtbl.add q_hash 1 ("id1", [2; 3]);
      let q_tag_indexed = Hashtbl.create 3 in
      update_question_tag_indexed q_tag_indexed q_hash;
      assert(IntSet.elements (Hashtbl.find q_tag_indexed 0) = [0]);
      assert(IntSet.elements (Hashtbl.find q_tag_indexed 2) = [1]);
      assert(IntSet.elements (Hashtbl.find q_tag_indexed 3) = [0; 1]);
      assert(compute_selected_questions q_tag_indexed [0] = [0]);
      assert(compute_selected_questions q_tag_indexed [2; 3] = [1]);
      assert(compute_selected_questions q_tag_indexed [0; 2] = []);
      Printf.printf "done\n"
    end
```

Chapter 30

URL handling

Module `Url` provides routines to parse demexp URLs. demexp URLs follow the pattern: `demexp://server[:port][`
where the `port` and the `action` are optional.

The optional `action` can be:

- `browse/question/45`: open browser on question 45;
- `vote/question/45`: vote on question 45;
- `stop_server`: put server on halt;
- `server_timers`: returns the value of timers measured on server as a string.

164a ⟨*url.ml* 164a⟩≡ 164b ▷
```
(* copyright 2005 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Str
```

We define statically the regexp to parse the server part of the url.

164b ⟨*url.ml* 164a⟩+≡ ◁164a 164c ▷
```
let server_regexp =
  regexp "demexp://\\([a-zA-Z0-9.-]+\\)\\(:[0-9]+\\)?\\(.*\\)"
```

The type `action` defines the different set of action that can be found in a demexp URL.

164c ⟨*url.ml* 164a⟩+≡ ◁164b 164d ▷
```
type action =
  | No_action
  | Browse_action of int (* question_id *)
  | Vote_action of int (* question_id *)
  | Stop_server_action
  | Server_timers
```

After analysis of an URL, we return an `url` data structure.

164d ⟨*url.ml* 164a⟩+≡ ◁164c 164e ▷
```
type url = {
    server : string;
    port : int option;
    action : action;
  }
```

In case of error in the analysis, we raise exception `Bad_url`, with a cause as string argument.

164e ⟨*url.ml* 164a⟩+≡ ◁164d 165a ▷
```
exception Bad_url of string (* cause *)
```

We define the regexps to match the different recognized actions.

165a ⟨*url.ml* 164a⟩+≡ ◁164e 165b▷

```
let browse_regexp = regexp "/browse/question/\\([0-9]+\\)"
let vote_regexp = regexp "/vote/question/\\([0-9]+\\)"
```

Function `parse_action_part` parses the right side of an URL, after the server part. It returns the found action as an `action` sum type.

A failure on `int_of_string` can never happen because the regexp matches only digits for this part.

165b ⟨*url.ml* 164a⟩+≡ ◁165a 165c▷

```
let parse_action_part action_str =
  if action_str = "/" || action_str = "" then
    No_action
  else if action_str = "/stop_server" then
    Stop_server_action
  else if action_str = "/server_timers" then
    Server_timers
  else if string_match browse_regexp action_str 0 then
    Browse_action (int_of_string (matched_group 1 action_str))
  else if string_match vote_regexp action_str 0 then
    Vote_action (int_of_string (matched_group 1 action_str))
  else
    raise (Bad_url ("unknown action: '" ^ action_str ^ "'"))
```

Function `parse` parses a string URL `str_url` and returns an `url` data structure. It can raise `Bad_url` exception in case of failure.

A failure on `int_of_string` can never happen because the regexp matches only digits for this part.

165c ⟨*url.ml* 164a⟩+≡ ◁165b 166▷

```
let parse str_url =
  if string_match server_regexp str_url 0 then (
    let server = matched_group 1 str_url in
    let port =
      try
        (* avoid ":" *)
        let port_str = string_after (matched_group 2 str_url) 1 in
        Some (int_of_string port_str)
      with Not_found -> None in
    let action =
      try
        let action_str = matched_group 3 str_url in
        parse_action_part action_str
      with Not_found -> No_action in
    { server = server; port = port; action = action }
   ) else
    raise (Bad_url "bad format")
```

30.1 Automatic tests

166 ⟨*url.ml* 164a⟩+≡ ◁165c

```
  let _ =
    if Config.do_autotests then begin
  (*     let print_url url = *)
  (*       Printf.printf "%s" url.server; *)
  (*       (match url.port with *)
  (*       | None -> () *)
  (*       | Some n -> Printf.printf ":%d" n); *)
  (*       match url.action with *)
  (*       | No_action -> () *)
  (*       | Browse_action q_id -> Printf.printf "/browse/action/%d\n" q_id *)
  (*       | Vote_action q_id -> Printf.printf "/vote/action/%d\n" q_id *)
  (*       | Stop_server_action -> Printf.printf "/stop_server" *)
  (*       | Server_timers -> Printf.printf "/server_timers" in *)

      Printf.printf "  url autotests...";
      (try
        ignore(parse "toto");
        assert(false)
      with Bad_url "bad format" -> ());
      assert(parse "demexp://server.org" = { server = "server.org";
                                             port = None;
                                             action = No_action; });
      assert(parse "demexp://server.org/" = { server = "server.org";
                                              port = None;
                                              action = No_action; });
      assert(parse "demexp://server.org:1234" = { server = "server.org";
                                                  port = Some 1234;
                                                  action = No_action; });
      assert(parse "demexp://server.org:1234/" = { server = "server.org";
                                                   port = Some 1234;
                                                   action = No_action; });
      assert(parse "demexp://server.org:1234" = { server = "server.org";
                                                  port = Some 1234;
                                                  action = No_action; });
      assert(parse "demexp://127.0.0.1" = { server = "127.0.0.1";
                                            port = None;
                                            action = No_action; });
      assert(parse "demexp://127.0.0.1:1234" = { server = "127.0.0.1";
                                                 port = Some 1234;
                                                 action = No_action; });
      assert(parse "demexp://server.org:1234/browse/question/56"
               = { server = "server.org";
                   port = Some 1234;
                   action = Browse_action 56; });
      assert(parse "demexp://server.org:1234/vote/question/0"
               = { server = "server.org";
                   port = Some 1234;
                   action = Vote_action 0; });
      assert(parse "demexp://server.org/vote/question/0"
               = { server = "server.org";
                   port = None;
                   action = Vote_action 0; });
      assert(parse "demexp://server.org/stop_server"
               = { server = "server.org";
                   port = None;
```

```
              action = Stop_server_action; });
  assert(parse "demexp://server.org/server_timers"
           = { server = "server.org";
               port = None;
               action = Server_timers; });
  (try
    ignore(parse "demexp://server.org:1234/toto/titi");
    assert(false)
  with Bad_url "unknown action: '/toto/titi'" -> ());
  (try
    ignore(parse "demexp:///toto/titi");
    assert(false)
  with Bad_url "bad format" -> ());
  (try
    ignore(parse "demexp://server.org:1234Z/vote/question/0");
    assert(false)
  with Bad_url "unknown action: 'Z/vote/question/0'" -> ());
  Printf.printf "done\n"
end
```

Chapter 31

Client main module

This module defines the starting point of the LablGTK2 client.

168a ⟨*demexp-gtk2-client.ml* 168a⟩≡ 168b ▷
```
(* copyright 2004-2006 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Messages_aux
open Messages_clnt
open Clntflags
open Url
open DemexpGettext.Gettext
```

31.1 Command line parsing

`usage_msg` contains the usage information for demexp client printed with option `--help` or when an option is not recognized.

168b ⟨*demexp-gtk2-client.ml* 168a⟩+≡ ◁ 168a 168c ▷
```
let usage_msg = s_ "demexp-gtk2-client [options]\noptions are:"
```

We initialize `ocaml-gettext`.

168c ⟨*demexp-gtk2-client.ml* 168a⟩+≡ ◁ 168b 168d ▷
```
let gettext_args, _ = DemexpGettext.Gettext.init
```

168d ⟨*demexp-gtk2-client.ml* 168a⟩+≡ ◁ 168c 169a ▷
```
let cmdline_options =
  Arg.align ([
   ("--preference-dir", Arg.Set_string flag_pref_dir,
    s_ "dir use given DIR as preference directory");
   ("-l", Arg.Set flag_log, s_ " enable logging on stdout");
   ("--log", Arg.Set flag_log, s_ " ditto");
   ("--autotests", Arg.Set flag_autotests, s_ " do auto-tests");
   ("--dexceptions", Arg.Set flag_dexceptions,
    s_ " (debug) don't catch exceptions");
   ("--dall-dialogs", Arg.Set flag_dall_dialogs,
    s_ " (debug) open all dialogs");
   ("--update-voted-state", Arg.Set flag_update_voted_state,
    s_ " force update of voted state from server");
  ] @ gettext_args)
```

Function `parse_cmdline` parses command line options. It sets global flags defined in `Clntflags` (see code chunk 91a).

169a ⟨*demexp-gtk2-client.ml* 168a⟩+≡ ◁168d 169b▷

```
let parse_cmdline () =
  let anon_fun url =
    flag_url_list := url :: !flag_url_list in
  Arg.parse cmdline_options anon_fun usage_msg
```

31.2 Connection to server

Helper function `string_of_rpc_error` returns the RPC `error` as a meaningful string.

169b ⟨*demexp-gtk2-client.ml* 168a⟩+≡ ◁169a 169c▷

```
let string_of_rpc_error error =
  match error with
  | Rpc.Unavailable_program -> "Unavailable_program"
  | Rpc.Unavailable_version (i1, i2) ->
      Printf.sprintf "Unavailable_version (%s, %s)"
        (Rtypes.uint4_as_string i1) (Rtypes.uint4_as_string i2)
  | Rpc.Unavailable_procedure -> "Unavailable_procedure"
  | Rpc.Garbage -> "Garbage"
  | Rpc.System_err -> "System_err"
  | Rpc.Rpc_mismatch (i1,i2) ->
      Printf.sprintf "Rpc_mismatch (%s, %s)"
        (Rtypes.uint4_as_string i1) (Rtypes.uint4_as_string i2)
  | Rpc.Auth_bad_cred -> "Auth_bad_cred"
  | Rpc.Auth_rejected_cred -> "Auth_rejected_cred"
  | Rpc.Auth_bad_verf -> "Auth_bad_verf"
  | Rpc.Auth_rejected_verf -> "Auth_rejected_verf"
  | Rpc.Auth_too_weak -> "Auth_too_weak"
  | Rpc.Auth_invalid_resp -> "Auth_invalid_resp"
  | Rpc.Auth_failed -> "Auth_failed"
```

Exception `Connection_error` is raised when the client is unable to connect to the server.

169c ⟨*demexp-gtk2-client.ml* 168a⟩+≡ ◁169b 170▷

```
exception Connection_error
```

Function `connect_to_server` opens a new TCP connection to the server and logs in as the user defined in `pref` (of type `Pref.preferences` class). This function returns the `client` identifier and the login `cookie`. The function can raise a `Connection_error` exception.

170 ⟨*demexp-gtk2-client.ml* 168a⟩+≡ ◁169c 171▷

```
let connect_to_server pref user_msg =
  let timer = Perf.timer_start () in
  user_msg (Printf.sprintf (f_ "Connecting to server %s:%d...")
              pref#server_name pref#server_port);
  try
    let client = Demexp.V1.create_client
        (Rpc_client.Inet (pref#server_name, pref#server_port)) Rpc.Tcp in
    let login = pref#user_login in
    let pass = pref#user_password in
    user_msg (Printf.sprintf
                (f_ "Connected to server. Try to login with \"%s\"...")
                login);
    let r = Demexp.V1.login client (Rtypes.int_of_uint4 protocol_version,
                                    login, pass) in
    if r.login_return_code <> rt_ok then (
      MiscUI.display_error
        (Printf.sprintf
           (f_ "Unable to login (%s).\nContinue as Anonymous (just browse).")
           (Misc.string_of_return_code r.login_return_code))
     ) else
      user_msg (s_ "Logged in.");
    if r.server_protocol_version <> Rtypes.int_of_uint4 protocol_version then (
      MiscUI.display_error
        (Printf.sprintf
           (f_ "Unmatched protocol version (server:%d <> client:%d), please upgrade your demexp
           r.server_protocol_version
           (Rtypes.int_of_uint4 protocol_version));
      raise Connection_error;
     );
    Perf.timer_stop_and_record "Browser.connect_to_server" timer;
    (client, r.login_cookie)
  with
  | Unix.Unix_error (err, _, _) ->
      MiscUI.display_error
        (Printf.sprintf (f_ "Unable to connect to server: %s")
           (Glib.Convert.locale_to_utf8 (Unix.error_message err)));
      raise Connection_error
  | Failure str ->
      MiscUI.display_error
        (Printf.sprintf (f_ "Unable to connect to server: %s")
           (Glib.Convert.locale_to_utf8 str));
      raise Connection_error
  | Rpc.Rpc_server err ->
      MiscUI.display_error
        (Printf.sprintf
           (f_ "RPC error returned by server (%s). You should probably upgrade your demexp clie
           (string_of_rpc_error err));
      raise Connection_error
```

Function `disconnect_of_server` is called to close a connection to a server. It sends an RPC `goodbye()` to the server and then closes TCP connection.

171 ⟨*demexp-gtk2-client.ml* 168a⟩+≡ ◁170 172▷

```
let disconnect_of_server connection_parameters =
  match connection_parameters with
  | None -> ()
  | Some (client, cookie) ->
      Demexp.V1.goodbye client cookie;
      Rpc_client.shut_down client
```

Function `open_a_browser` open a browser on server described by `str_url`. It returns the optional connection parameters `(client_identifier, cookie)` or `None`.

172 ⟨*demexp-gtk2-client.ml* 168a⟩+≡ ◁171 173▷

```
let open_a_browser user_msg str_url =
  (* open preferences *)
  let relative =
    Clntflags.default_config_dirname = !Clntflags.flag_pref_dir in
  let pref =
    new Pref.preferences user_msg ~relative !Clntflags.flag_pref_dir () in
  pref#load;
  (* parse url *)
  let url = parse str_url in
  let server = url.server in
  let port =
    match url.port with
    | None -> Config.default_server_port
    | Some n -> n in
  (* configure preferences for this server *)
  pref#set_server_name server;
  pref#set_server_port port;
  try
    (* connect to server *)
    let client, cookie = connect_to_server pref user_msg in
    (* open appropriate browser *)
    (match url.action with
    | Stop_server_action ->
        let err = Demexp.V1.stop_server client cookie in
        if err = Messages_aux.rt_ok then
          Printf.printf (f_ "Server will stop.\n")
        else
          (Printf.eprintf (f_ "Unable to stop server, an error occured (%s)\n")
             (Misc.string_of_return_code err));
        exit 0
    | Server_timers ->
        let str = Demexp.V1.server_timers client cookie in
        Printf.printf "\n== Server timers of %s:%d ==\n%s" server port str;
        Browser.ui_demexp pref (Some (client, cookie)) Browser.Nothing
    | Browse_action q_id ->
        Browser.ui_demexp pref (Some (client, cookie)) (Browser.Browse q_id);
    | Vote_action q_id ->
        Browser.ui_demexp pref (Some (client, cookie)) (Browser.Vote q_id);
    | No_action ->
        Browser.ui_demexp pref (Some (client, cookie)) Browser.Nothing);
    Some (client, cookie)
  with
  | Bad_url cause ->
      user_msg (Printf.sprintf (f_ "ERROR: unable to parse url '%s' (%s)")
                  str_url cause);
      None
  | Connection_error ->
      (* we still start the UI, to allow the user to set its preferences *)
      Browser.ui_demexp pref None Browser.Nothing;
      None
```

31.3 Main window

We finally define the main function of this client. This function initializes GTK, setups a preference object and then opens needed browsing windows, depending on URLs given on the command line. At the end, GTK main loop is called to receive and handle events. When this loop finishes, the connection to servers are closed.

fixme: The `user_msg` function should display in a graphical window and not just on stdout. FIXME

173 ⟨*demexp-gtk2-client.ml* 168a⟩+≡ ◁172

```
let _ =
  parse_cmdline ();
  if !flag_dall_dialogs then (* for debug or pen & paper user test *)
    Demexp_gladeui.check_all ~show:true ()
  else if not !flag_autotests then (
    flush_all ();
    ignore (GMain.Main.init ());
    (* fixme: use a display-less user_msg for now *)
    let user_msg msg = Printf.printf "%s\n" msg in
    let connection_parameters =
      if !flag_url_list <> [] then
        List.map (open_a_browser user_msg) !flag_url_list
      else
        [open_a_browser user_msg
           (Printf.sprintf "demexp://%s:%d" Config.default_server_name
              Config.default_server_port)] in
    GMain.Main.main ();
    List.iter disconnect_of_server connection_parameters;
    Printf.printf "\n== Client timers ==\n";
    Perf.print_timers ()
   )
```

Part V

Server (`srv/`)

Chapter 32

Definition of command line flags

Module `Srvflags` defines command line option that are set when the server is launched.

175a ⟨*srvflags.ml* 175a⟩≡ 175b ▷
```
(* copyright 2003-2005 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Config
```

32.1 Global flags

Flags used throughout the server.

175b ⟨*srvflags.ml* 175a⟩+≡ ◁175a 175c ▷
```
let default_log_filename = ""

let flag_debug = ref false                          (* -d --debug *)
and flag_autotests = ref false                      (* --autotests *)
and flag_address = ref default_server_address       (* --listen host *)
and flag_port = ref default_server_port             (* --listen :port *)
and flag_bases_name = ref default_bases_name        (* --bases *)
and flag_log_filename = ref default_log_filename    (* --logfile *)
and flag_daemon = ref false                         (* --daemon *)
```

32.2 Log functions

By default, standard log are sent to standard output and debug log are sent to standard error.

175c ⟨*srvflags.ml* 175a⟩+≡ ◁175b 176a ▷
```
let log_formatter = ref Format.std_formatter

let debug_formatter = ref Format.err_formatter
```

However, `redirect_logs_to_file` can be called when command line options are parsed to redirect normal and debug logs to a file named `filename`. In case `filename` is "`-`", the logs are redirected to standard output.

176a ⟨*srvflags.ml* 175a⟩+≡ ◁175c 176b▷

```
let redirect_logs_to_file () =
  if !flag_log_filename = "" then
    failwith "internal error: Srvflags.redirect_logs_to_file";
  if !flag_log_filename = "-" then (
    log_formatter := Format.std_formatter;
    debug_formatter := Format.std_formatter
   ) else (
    try
      let oc = open_out !flag_log_filename in
      let fmt = Format.formatter_of_out_channel oc in
      log_formatter := fmt;
      debug_formatter := fmt
    with Sys_error str ->
      Format.eprintf
        "ERROR: cannot open log file '%s': %s\n" !flag_log_filename str
  )
```

Helper function `current_time_as_string` returns current time in international ISO-8601 string format.

176b ⟨*srvflags.ml* 175a⟩+≡ ◁176a 176c▷

```
let current_time_as_iso_string () =
  Time.time_as_localtime_iso_string (Unix.time ())
```

We define the helper function `dbg` which prints on output its arguments only if `flag_debug` is `true`. `dbg` can be used in the same way as `printf`.

The trick here is to call `kprintf` as last expression in the function: http://caml.inria.fr/archives/200405/msg00355.html.

176c ⟨*srvflags.ml* 175a⟩+≡ ◁176b 176d▷

```
let dbg fmt =
  let print_if_necessary str =
    if !flag_debug then (
      let id = Thread.id (Thread.self ()) in
      Format.fprintf !debug_formatter
        "<%d>%s@[ %s@]@." id (current_time_as_iso_string ()) str;
     ) in
  Format.kprintf print_if_necessary fmt
```

Helper function `log` is similar to `dbg`, but print its output to standard output.

176d ⟨*srvflags.ml* 175a⟩+≡ ◁176c

```
let log fmt =
  let print str =
    let id = Thread.id (Thread.self ()) in
    Format.fprintf !log_formatter
      "<%d>%s@[ %s@]@." id (current_time_as_iso_string ()) str in
  Format.kprintf print fmt
```

Chapter 33

Dynamic Bit Vector

The module `Dbitv` implements Dynamically resizable Bit Vectors. It has basically the same API (Application Programming Interface) as `Bitv`[1] module of Jean-Christophe Filliâtre but the vectors given in argument can be of different size. In such a case, the smallest vector is extended to be of the same size as the biggest one.

In case of error, the same exceptions as in the `Bitv` module are raised.

33.1 Data structure

177a ⟨*dbitv.ml* 177a⟩≡ 177b ▷
```
(* copyright 2003 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Printf
```

Our Dynamic Bit Vector is a `Bitv` bit vector `vec`, of length `length`.

177b ⟨*dbitv.ml* 177a⟩+≡ ◁177a 177c ▷
```
type t = { mutable vec : Bitv.t;
           mutable length : int;
         }
```

33.2 Creation, access and assignement

`create` creates a fresh bit vector of length `len`, with all elements set to `init` value.

177c ⟨*dbitv.ml* 177a⟩+≡ ◁177b 177d ▷
```
let create ~len ~init =
  { vec = Bitv.create len init; length = len }
```

`init` initializes a fresh bit vector of length `len` where all bits are set by calling function `f`.

177d ⟨*dbitv.ml* 177a⟩+≡ ◁177c 177e ▷
```
let init ~len ~f =
  { vec = Bitv.init len f; length = len }
```

`set` sets the bit `n` of vector `vec` to boolean value `b`. Bits are numbered from 0 to `length vec - 1`.

177e ⟨*dbitv.ml* 177a⟩+≡ ◁177d 178a ▷
```
let set ~vec ~n ~b =
  Bitv.set vec.vec n b
```

[1] http://www.lri.fr/~filliatr/software.en.html

`get` returns the boolean value of bit `n` of vector `vec`.

178a ⟨*dbitv.ml* 177a⟩+≡ ◁177e 178b▷
```
let get ~vec ~n =
  Bitv.get vec.vec n
```

`length` returns length of vector `vec`.

178b ⟨*dbitv.ml* 177a⟩+≡ ◁178a 178c▷
```
let length ~vec =
  vec.length
```

`max_length` returns the system dependant maximum size of a `Dbitv` vector.

178c ⟨*dbitv.ml* 177a⟩+≡ ◁178b 178d▷
```
let max_length =
  Bitv.max_length
```

33.3 Resizing

Function `resize` resizes the vector `vec` to length `len`, setting added elements to default value `init`. In case the requested length `len` is smaller that the actual length of `vec`, exception `Invalid_argument "Dbitv.resize len"` is raised.

178d ⟨*dbitv.ml* 177a⟩+≡ ◁178c 178e▷
```
let resize ~vec ~len ~init =
  if len < vec.length then raise (Invalid_argument "Dbitv.resize len");
  let v = Bitv.append vec.vec (Bitv.create (len - vec.length) init) in
  vec.vec <- v;
  vec.length <- len
```

Function `resize_if_needed` takes to vectors `a` and `b` and resizes the smallest one to the length of the biggest one.

178e ⟨*dbitv.ml* 177a⟩+≡ ◁178d 178f▷
```
let resize_if_needed ~a ~b =
  if a.length < b.length then resize ~vec:a ~len:b.length ~init:false
  else if b.length < a.length then resize ~vec:b ~len:a.length ~init:false
```

Function `resize_set` sets the bit `n` of vector `vec` to boolean value `b`. In case the vector `vec` is not big enough, it is extended to the required size.

178f ⟨*dbitv.ml* 177a⟩+≡ ◁178e 178g▷
```
let resize_set ~vec ~n ~b =
  if n > vec.length - 1 then resize ~vec ~len:(n+1) ~init:false;
  set ~vec ~n ~b
```

33.4 Copies and concatenation

Once again, no change with `Bitv` module.

178g ⟨*dbitv.ml* 177a⟩+≡ ◁178f 179a▷
```
let copy ~vec =
  { vec = Bitv.copy vec.vec; length = vec.length }

let append ~a ~b =
  let v = Bitv.append a.vec b.vec in
  { vec = v; length = Bitv.length v }

let concat l =
  let v = Bitv.concat (List.map (fun v -> v.vec) l) in
  { vec = v; length = Bitv.length v }
```

33.5 Sub-vectors and filling

Do you really think there is a change with `Bitv` module?

179a ⟨*dbitv.ml* 177a⟩+≡ ◁178g 179b▷

```
let sub ~vec ~start ~len =
  { vec = Bitv.sub vec.vec start len; length = len }

let fill ~vec ~ofs ~len ~init =
  Bitv.fill vec.vec ofs len init

let blit ~v1 ~o1 ~v2 ~o2 ~len =
  Bitv.blit v1.vec o1 v2.vec o2 len
```

33.6 Iterators

We define the same operators as in `Bitv` module.

179b ⟨*dbitv.ml* 177a⟩+≡ ◁179a 180a▷

```
let iter ~f ~vec =
  Bitv.iter f vec.vec

let map ~f ~vec =
  { vec = Bitv.map f vec.vec; length = vec.length }

let iteri ~f ~vec =
  Bitv.iteri f vec.vec

let mapi ~f ~vec =
  { vec = Bitv.mapi f vec.vec; length = vec.length }

let fold_left ~f ~init ~vec =
  Bitv.fold_left f init vec.vec

let fold_right ~f ~vec ~init =
  Bitv.fold_right f vec.vec init
```

33.7 Bitwise operations

We redefine the bitwise operations as found in `Bitv` module. The only added step in each operation makes the two vectors `a` and `b` of equal length before applying the operation.

180a ⟨*dbitv.ml* 177a⟩+≡ ◁179b 180b▷

```
let bw_and ~a ~b =
  resize_if_needed ~a ~b;
  { vec = Bitv.bw_and a.vec b.vec; length = a.length }

let bw_or ~a ~b =
  resize_if_needed ~a ~b;
  { vec = Bitv.bw_or a.vec b.vec; length = a.length }

let bw_xor ~a ~b =
  resize_if_needed ~a ~b;
  { vec = Bitv.bw_xor a.vec b.vec; length = a.length }

let bw_not ~vec =
  { vec = Bitv.bw_not vec.vec; length = vec.length }

let shiftl ~vec ~shift =
  { vec = Bitv.shiftl vec.vec shift; length = vec.length }

let shiftr ~vec ~shift =
  { vec = Bitv.shiftr vec.vec shift; length = vec.length }

let all_zeros ~vec =
  Bitv.all_zeros vec.vec

let all_ones ~vec =
  Bitv.all_ones vec.vec
```

33.8 Conversions to and from string

Once again, same API as in `Bitv`.

180b ⟨*dbitv.ml* 177a⟩+≡ ◁180a 181a▷

```
let to_string ~vec =
  Bitv.to_string vec.vec

let from_string ~init =
  let v = Bitv.from_string init in
  { vec = v; length = Bitv.length v }

let print ~formatter ~vec =
  Bitv.print formatter vec.vec
```

33.9 Conversions to and from lists of integers

181a ⟨*dbitv.ml* 177a⟩+≡ ◁180b 181b▷

```
let to_list ~vec =
  Bitv.to_list vec.vec

let from_list ~init =
  let v = Bitv.from_list init in
  { vec = v; length = Bitv.length v }

let from_list_with_length ~init ~len =
  let v = Bitv.from_list_with_length init len in
  { vec = v; length = len }
```

33.10 Automatic tests

We do some simple auto-tests to check size handling. Otherwise, we rely on correctness of Jean-Christophe's module.

181b ⟨*dbitv.ml* 177a⟩+≡ ◁181a

```
let _ =
  if Config.do_autotests then begin
    printf "  dbitv autotests...";
    let a = create ~len:3 ~init:false in
    let b = create ~len:4 ~init:true in
(*     printf "a:%s\n" (to_string a); *)
(*     printf "b:%s\n" (to_string b); *)
    assert(bw_or a b = b);
    assert(all_zeros(bw_and a b) = true);
    assert(length a = 4);
    assert(get ~vec:a ~n:3 = false);
    set ~vec:a ~n:3 ~b:true;
    let c = init ~len:6 ~f:(fun i -> i mod 2 = 1) in
    let d = copy a in
    assert(get a 1 = false);
    set d 1 true;
    assert(get a 1 = false);
    let e = bw_or c d in
    assert(length e = 6);
    assert(length c = 6);
    assert(length d = 6);
    resize_set ~vec:d ~n:7 ~b:true;
    assert(get d 7 = true);
    assert(length d = 8);
    printf "done\n"
  end
```

Chapter 34

RSS feed

Module `Rss` allows to output new questions and responses as an RSS 2.0 feed.

For Apache, you can put following command in a `.htaccess` file if you want that the RSS feed to be given proper HTTP type.

```
#  .htaccess for RSS feed
AddType application/xml .rss
```

182a ⟨*rss.ml* 182a⟩≡ 182b ▷

```
(* copyright 2005 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Format
open Srvflags
```

34.1 Generic feed

In the feed, we store new questions. The feed is stored as `queue` of `maximum_size` elements.

182b ⟨*rss.ml* 182a⟩+≡ ◁182a 182c ▷

```
type t = {
    queue: (int * string * (string list) * ((string * string) list)) Queue.t;
        (* q_id * q_desc *    tag list   *  (r_desc * r_link) list *)
    maximum_size: int;
  }
```

The `create` function creates a new feed of `maximum_size`.

182c ⟨*rss.ml* 182a⟩+≡ ◁182b 182d ▷

```
let create maximum_size =
  { queue = Queue.create (); maximum_size = maximum_size; }
```

Function `add_in_feed` adds a new `element` in a `feed`. If the feed is too big, it is truncated.

182d ⟨*rss.ml* 182a⟩+≡ ◁182c 183a ▷

```
let add_in_feed element feed =
  Queue.add element feed.queue;
  if Queue.length feed.queue > feed.maximum_size then
    ignore(Queue.pop feed.queue)
```

34.2 Output of feed

Function `write_rss_header` writes the leading part of an RSS feed to formater `f`. We use `Format` module facilities to nicely format the output.

183a ⟨*rss.ml* 182a⟩+≡ ◁182d 183b▷

```
let write_rss_header f =
  fprintf f "<?xml version=\"1.0\"?>@\n";
  fprintf f
    "<rss version=\"2.0\" xmlns:dc=\"http://purl.org/dc/elements/1.1/\">@\n";
  fprintf f "  <channel>@\n    @[";
  fprintf f "<title>demexp %s:%d</title>@\n" !flag_address !flag_port;
  fprintf f "<link>demexp://%s:%d</link>@\n" !flag_address !flag_port;
  fprintf f "<description>New events in demexp server %s:%d</description>@\n"
    !flag_address !flag_port
```

Function `write_rss_footer` writes the trailing part of an RSS feed to formater `f`.

183b ⟨*rss.ml* 182a⟩+≡ ◁183a 183c▷

```
let write_rss_footer f =
  fprintf f "@]@\n  </channel>@\n</rss>@."
```

Helper function `escape_xml_characters` rewrites `str` to produce a string that can be used inside an XML document. Basically, characters `<`, `>`, `&`, and `"` are rewrote into `<`, `>`, `&` and `&quot;`.

183c ⟨*rss.ml* 182a⟩+≡ ◁183b 184a▷

```
let escape_xml_characters str =
  let lt_regex = Str.regexp "<"
  and gt_regex = Str.regexp ">"
  and amp_regex = Str.regexp "&"
  and quot_regex = Str.regexp "\"" in
  (* we start with '&' first otherwise we would substitute following '&' :) *)
  let a = Str.global_replace amp_regex "&amp;" str in
  let b = Str.global_replace gt_regex "&gt;" a in
  let c = Str.global_replace lt_regex "&lt;" b in
  Str.global_replace quot_regex "&quot;" c
```

Function `write_rss_element` writes the content of feed `element` to the formatter `f`. The `<guid>` tag (or a `<link>` tag) that uniquely identifies the item is mandatory otherwise Firefox does not display the feed correctly. As we don't have usable URL, we must set attribute `isPermaLink="false"`.

184a ⟨*rss.ml* 182a⟩+≡ ◁183c 184b▷

```
let write_rss_element f element =
  let write ~q_id ~title ~description =
    let xml_title = escape_xml_characters title in
    let xml_description = escape_xml_characters description in
    fprintf f "<item>@\n  @[";
    fprintf f "<title>%s</title>@\n" xml_title;
    fprintf f "<description>\n%s@\n</description>@\n" xml_description;
    fprintf f "<link>demexp://%s:%d/browse/question/%d</link>@\n"
      !flag_address !flag_port q_id;
    fprintf f "@]</item>@\n@\n" in
  let q_id, q_desc, tag_list, resp_list = element in
  let title = sprintf "%d. %s" q_id q_desc in
  let tag_str =
    List.fold_left (fun s t -> s ^ (sprintf "Tag: %s\n" t)) "" tag_list in
  let _, resp_str =
    List.fold_left
      (fun (i, s) (r_desc, r_link) ->
        (i + 1, s ^ (sprintf "%d. %s [%s]\n" i r_desc r_link)))
      (0, "") resp_list in
  let description = tag_str ^ "\nResponses:\n" ^ resp_str in
  write ~q_id ~title ~description
```

Function `write_rss` outputs RSS `feed` as a file with the same name as bases but with suffix "`.rss`" (or "`rss`" if the bases have not extension). The feed elements are printed from the earliest to the oldest.

184b ⟨*rss.ml* 182a⟩+≡ ◁184a 185a▷

```
let write_rss feed =
  let elements = Queue.fold (fun accu e -> e :: accu) [] feed.queue in
  let filename =
    try
      (Filename.chop_extension !flag_bases_name) ^ ".rss"
    with Invalid_argument("Filename.chop_extension") ->
      (Filename.dirname !flag_bases_name) ^ "/rss" in
  try
    let out = open_out filename in
    let f = formatter_of_out_channel out in
    write_rss_header f;
    List.iter (fun e -> write_rss_element f e) elements;
    write_rss_footer f;
    close_out out
  with Sys_error str ->
    log "ERROR: could not write file %s. Reason: %s" filename str
```

34.3 Server wide RSS feed

We define an RSS feed that is global to this server, with corresponding addition functions that perform effective writting of RSS feed.

185a ⟨*rss.ml* 182a⟩+≡ ◁184b 185b▷

```
let rss_feed = create Config.maximum_size_of_rss_feed

let add ~q_id ~q_desc ~tag_list ~resp_list =
  add_in_feed (q_id, q_desc, tag_list, resp_list) rss_feed;
  write_rss rss_feed
```

A function `clear_feed` cleanup a feed, to be used after auto-tests.

185b ⟨*rss.ml* 182a⟩+≡ ◁185a 185c▷

```
let clear_feed () = Queue.clear rss_feed.queue
```

34.4 Automatic tests

185c ⟨*rss.ml* 182a⟩+≡ ◁185b

```
let _ =
  if Config.do_autotests then begin
    Printf.printf "  rss autotests...";
    let f = create 1 in
    add_in_feed (0, "a", [], []) f;
    assert(Queue.top f.queue = (0, "a", [], []));
    add_in_feed (1, "b", [], []) f;
    assert(Queue.top f.queue = (1, "b", [], []));
    assert(escape_xml_characters "<>&\"" = "&lt;&gt;&amp;&quot;");
    Printf.printf "done\n"
  end
```

Chapter 35

Identifiers management

The `Id` module defines a generic module `IdTable` to manage a unique mapping (aka bijection) between an identifier and an information (typically a string). This mapping is called an *identifier table*.

This `IdTable` module is used to define identifiers management modules for Tags, Questions and Participants of classification, as well as to store their timestamps.

We use the module system of OCaml (with a functor) to create a generic module that defines specific ones, so we factorize code (so less code and less tests).

186a ⟨*id.ml* 186a⟩≡ 186b ▷
```
(* copyright 2004 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Printf
```

35.1 Generic module

We first define the signature that will allow to parameterize the generic module, where `info` is the type of information associated with an identifier (typically a string).

186b ⟨*id.ml* 186a⟩+≡ ◁186a 187a ▷
```
module type IDTABLE_PARAM = sig type info end
```

We then define the module generic module `IdTable` itself. It is guaranteed that the first identifier created in a table is 0 once converted into an integer.

To each identifier is associated a timestamp which is updated each time the information of an identifier is updated.

187a ⟨*id.ml* 186a⟩+≡ ◁186b 187b▷

```
module IdTable :
functor (SP : IDTABLE_PARAM) ->
  sig
    type t
    type info = SP.info
    exception Already_exists
    val create : unit -> t
    val of_list : (int * Timestamp.t * info) list -> t
    val force : t -> int -> Timestamp.t -> info -> unit
    val add : t -> info -> int
    val find : t -> int -> info
    val rev_find : t -> info -> int
    val replace : t -> int -> info -> unit
    val fold : (int -> Timestamp.t * info -> 'a -> 'a) -> t -> 'a -> 'a
    val iter : (int -> Timestamp.t * info -> unit) -> t -> unit
    val mem : t -> int -> bool
    val rev_mem : t -> info -> bool
    val remove : t -> int -> unit
    val rev_remove : t -> info -> unit
    val update_timestamp : t -> int -> unit
    val rev_update_timestamp : t -> info -> unit
    val get_timestamp : t -> int -> Timestamp.t
    val length : t -> int
    val timestamp_list : t -> Timestamp.block
  end
```

We now define the implementation of the above signature of `IdTable`. An identifier table is implemented using a integer counter of identifier (`next_id`) and two hash tables, one for the `direct` translation (from identifiers to information), and one for the `reverse` translation. In the `direct` table, we also associate a timestamp to each entry

When a couple (identifier, information) is added to an identifier table, it is simultaneously added to both tables.

187b ⟨*id.ml* 186a⟩+≡ ◁187a 188a▷

```
    =
  functor (SP : IDTABLE_PARAM) ->
  struct
    type t = { mutable next_id: int;
               direct: (int, Timestamp.t * SP.info) Hashtbl.t;
               reverse: (SP.info, int) Hashtbl.t; }
    type info = SP.info
    exception Already_exists
    let create () = { next_id = 0;
                      direct = Hashtbl.create 3;
                      reverse = Hashtbl.create 3; }
```

Operation `force` adds a new information with a given `id`. It assumes that an id is a natural integer, the `id` is not already allocated, and that the information `info` is not already in the table, otherwise an exception is raised.

188a ⟨*id.ml* 186a⟩+≡ ◁187b 188b▷

```
let force t id timestamp info =
  if id < 0 then
    failwith "IdTable.force: assumption not followed, negative id";
  if Hashtbl.mem t.direct id  then
    failwith "IdTable.force: assumption not followed, id already seen";
  if Hashtbl.mem t.reverse info then
    failwith "IdTable.force: assumption not followed, info already seen";
  Hashtbl.add t.direct id (timestamp, info);
  Hashtbl.add t.reverse info id;
  if id >= t.next_id then t.next_id <- id + 1
```

Operation `of_list` creates a new table from a list of triples `(id, timestamp, info)`. The assumptions are the same as in `force`.

188b ⟨*id.ml* 186a⟩+≡ ◁188a 188c▷

```
let of_list l =
  let t = { next_id = 0;
            direct = Hashtbl.create 3;
            reverse = Hashtbl.create 3; } in
  let add_entry (id, timestamp, info) = force t id timestamp info in
  List.iter add_entry l;
  t
```

Operation `add` adds a new `info` to the table, while checking it hasn't already been stored in it (an raises `Already_exists` in such a case).

188c ⟨*id.ml* 186a⟩+≡ ◁188b 188d▷

```
let add t info =
  let id = t.next_id in
  if id = max_int then
    failwith "internal error: maximum tag indentifier reached";
  if Hashtbl.mem t.reverse info then raise Already_exists;
  t.next_id <- t.next_id + 1;
  Hashtbl.add t.direct id (Timestamp.current (), info);
  Hashtbl.add t.reverse info id;
  id
```

Operations `find` and `rev_find` can both raised `Not_found` exception if the searched identifier or information do not exist in the table.

188d ⟨*id.ml* 186a⟩+≡ ◁188c 188e▷

```
let find t id =
  let _, info = Hashtbl.find t.direct id in
  info

let rev_find t info = Hashtbl.find t.reverse info
```

Operation `replace` change the currently stored information with new `info`. In case the searched `id` is not found, it raises `Not_found`.

188e ⟨*id.ml* 186a⟩+≡ ◁188d 189a▷

```
let replace t id info =
  let _, old_info = Hashtbl.find t.direct id in
  Hashtbl.replace t.direct id (Timestamp.current (), info);
  Hashtbl.remove t.reverse old_info;
  Hashtbl.add t.reverse info id
```

Operation `fold` has same behaviour as `Hashtbl.fold`.

189a ⟨*id.ml* 186a⟩+≡ ◁188e 189b▷
```
let fold f t init = Hashtbl.fold f t.direct init
```

Operation `iter` has same behaviour as `Hashtbl.iter`.

189b ⟨*id.ml* 186a⟩+≡ ◁189a 189c▷
```
let iter f t = Hashtbl.iter f t.direct
```

Operations `mem` and `rev_mem` have similar behavior as `Hashtbl.mem`: it returns `true` if the searched element exists in the table.

189c ⟨*id.ml* 186a⟩+≡ ◁189b 189d▷
```
let mem t id = Hashtbl.mem t.direct id

let rev_mem t info = Hashtbl.mem t.reverse info
```

Operation `remove` removes the entry `id` from the table. `rev_remove` is similar, but takes the `info` field as parameter. Both operations do nothing if the requested identifier is not in the table.

189d ⟨*id.ml* 186a⟩+≡ ◁189c 189e▷
```
let remove t id =
  try
    let _, info = Hashtbl.find t.direct id in
    Hashtbl.remove t.direct id;
    Hashtbl.remove t.reverse info
  with Not_found -> ()

let rev_remove t info =
  try
    let id = Hashtbl.find t.reverse info in
    let _, info = Hashtbl.find t.direct id in
    Hashtbl.remove t.direct id;
    Hashtbl.remove t.reverse info
  with Not_found -> ()
```

Operation `update_timestamp` sets the timestamp of `id` to the current one. Operation `rev_update_timestamp` is similar but uses `info` as lookup key.

189e ⟨*id.ml* 186a⟩+≡ ◁189d 189f▷
```
let update_timestamp t id =
  let _, info = Hashtbl.find t.direct id in
  Hashtbl.replace t.direct id (Timestamp.current (), info)

let rev_update_timestamp t info =
  let id = Hashtbl.find t.reverse info in
  Hashtbl.replace t.direct id (Timestamp.current (), info)
```

Operation `get_timestamp` returns the timestamp of identifier `id`.

189f ⟨*id.ml* 186a⟩+≡ ◁189e 189g▷
```
let get_timestamp t id =
  let timestamp, _ = Hashtbl.find t.direct id in
  timestamp
```

Operation `length` returns the number of elements in the table. For an empty table, it is 0. For a table with one element, it is 1 (the only element has identifier 0).

189g ⟨*id.ml* 186a⟩+≡ ◁189f 190a▷
```
let length t = t.next_id
```

Operation `timestamp_list` returns the complete set of timestamps in `block`.

190a ⟨*id.ml* 186a⟩+≡ ◁189g 190b▷

```
let timestamp_list t =
  let number = length t in
  let block = Timestamp.create number in
  fold (fun id (ts, _) _ -> block.{id} <- ts) t ();
  block
```

We have now finished the definition of the generic identifier table.

190b ⟨*id.ml* 186a⟩+≡ ◁190a 190c▷

```
end
```

35.2 Definition of identifier tables

We now define the module type that will allow to define identifier tables to manage Tags and Questions.

190c ⟨*id.ml* 186a⟩+≡ ◁190b 191▷

```
module Tag = IdTable(struct type info = string end)

module Question = IdTable(struct type info = string end)

module Participant = IdTable(struct type info = string end)
```

35.3 Automatic tests

191 ⟨*id.ml* 186a⟩+≡ ◁190c

```
let _ =
  if Config.do_autotests then begin
    printf "  id autotests...";
    let q = Question.create () in
    assert(Question.length q = 0);
    let s1 = "string1" in
    let id1 = Question.add q s1 in
    assert(id1 = 0);
    assert(Question.find q id1 = s1);
    assert(Question.length q = 1);
    let s2 = "string2" in
    let id2 = Question.add q s2 in
    assert(Question.find q id2 = s2);
    assert(id2 = 1);
    assert(Question.find q id1 = s1);
    assert(Question.rev_find q s1 = id1);
    assert(Question.find q id2 = s2);
    assert(Question.rev_find q s2 = id2);

    assert(Question.mem q id2 = true);
    assert(Question.rev_mem q s2 = true);

    assert(List.sort compare
             (Question.fold (fun a (_, b) c -> (a, b) :: c) q [])
             = [(id1, s1); (id2, s2)]);

    (* check replace *)
    let new_s2 = "new string2" in
    Question.replace q id2 new_s2;
    assert(Question.find q id2 = new_s2);

    (* error cases *)
    (try
      ignore(Question.rev_find q "toto");
      assert(false)
    with Not_found -> assert(true));
    (try
      ignore(Question.find q 42);
      assert(false);
    with Not_found -> assert(true));
    (try
      ignore(Question.add q s1);
      assert(false)
    with Question.Already_exists -> assert(true));
    (try
      Question.replace q 42 "toto";
      assert(false);
    with Not_found -> assert(true));

    (* check remove *)
    Question.remove q id2;
    assert(Question.mem q id2 = false);
    assert(Question.rev_mem q s2 = false);
    Question.remove q 0;
    (try
```

```
    ignore(Question.find q 0);
    assert(false);
  with Not_found -> assert(true));
  let id3 = Question.add q s1 in
  assert(Question.mem q id3 = true);
  Question.rev_remove q s1;
  assert(Question.mem q id3 = false);

  ignore(Question.timestamp_list q);

  (* check of typing *)
  let t = Tag.create () in
  let id1t = Tag.add t s1 in
  (* we cannot enter following expression and this is a good thing because
     we cannot mix different types of ids:

     let _ = id1 = id1t in
     => This expression has type Tag.id but is here used with type
        Question.id
   *)

  (* check iter *)
  let l = ref [] in
  let f id (_, info) = l := (id, info) :: !l in
  Tag.iter f t;
  assert(!l = [ (id1t, s1) ]);

  (* check of_list *)
  let ts1 = Int32.of_int 2
  and ts2 = Int32.of_int 4 in
  let t = Tag.of_list [(1, ts1, "a"); (2, ts2, "b")] in
  ignore(Tag.add t "c");
  assert(Tag.find t 3 = "c");
  (try
    let _ = Tag.of_list [(-1, ts1, "a"); (2, ts2, "b")] in
    assert(false)
  with
    Failure "IdTable.force: assumption not followed, negative id" -> ());
  (try
    let _ = Tag.of_list [(1, ts1, "a"); (1, ts2, "b")] in
    assert(false)
  with
    Failure "IdTable.force: assumption not followed, id already seen" -> ());
  (try
    let _ = Tag.of_list [(1, ts1, "a"); (2, ts2, "a")] in
    assert(false)
  with
    Failure "IdTable.force: assumption not followed, info already seen" -> ());

  printf "done\n"
end
```

Chapter 36

Participants

This module defines code related to authentication and management of Participants.

193a ⟨*participants.ml* 193a⟩≡ 193b ▷
```
(* copyright 2003-2005 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)
```

We define some renaming of Digest standard module to make following code more clear.

193b ⟨*participants.ml* 193a⟩+≡ ◁193a 193c ▷
```
let md5_hash str = Digest.to_hex (Digest.string str)
```

The `delegate_string_prefix` defines the string that starts each delegate name.

193c ⟨*participants.ml* 193a⟩+≡ ◁193b 193d ▷
```
let delegate_string_prefix = "delegate_"
```

36.1 Participant identity

After login, an `authenticated_participant` is either `Authenticated_individual` or `Authenticated_delega` with a corresponding login, otherwise he remains `Anonymous`.

193d ⟨*participants.ml* 193a⟩+≡ ◁193c 193e ▷
```
type authenticated_participant =
    Anonymous
  | Authenticated_individual of string (* login *)
  | Authenticated_delegate of string (* login *)
```

36.2 Database of participants

A participant of the database is either an `Individual` or a `Delegate`.

193e ⟨*participants.ml* 193a⟩+≡ ◁193d 194a ▷
```
type participant_kind =
    Individual
  | Delegate
```

A participant is identified by its `login` within the system. When connecting to the demexp server, he authenticates himself by giving his `password`, which is stored as an MD5 cryptographic hash. A participant can pertain to zero or more `groups` of participants.

194a ⟨*participants.ml* 193a⟩+≡ ◁193e 194b▷
```
type participant_base_entry = {
    kind : participant_kind;
    mutable password : string;
    mutable groups : string list;
  }
```

We define the global base of participants, `the_participant_base` which contains:

`ids`: the table of participants identifiers;

`attr`: the attributes of each participant, stored as a hash table containing several `participant_base_entry` and indexed by the participant logins.

194b ⟨*participants.ml* 193a⟩+≡ ◁194a 194c▷
```
type participant_base = {
    ids: Id.Participant.t;
    attr: (string, participant_base_entry) Hashtbl.t;
  }
```

Function `create_participant_base` returns a reference on an empty base of participants.

194c ⟨*participants.ml* 193a⟩+≡ ◁194b 194d▷
```
let create_participant_base () =
  { ids = Id.Participant.create ();
    attr = Hashtbl.create 3; }
```

194d ⟨*participants.ml* 193a⟩+≡ ◁194c 194e▷
```
let the_participant_base : participant_base ref =
  ref (create_participant_base ())
```

We define the textual string used to represent the anonymous login.

194e ⟨*participants.ml* 193a⟩+≡ ◁194d 194f▷
```
let anonymous_as_string = "anonymous"
```

36.3 Checking of invariants

We define a set of functions that check that a Participant base is always in correct state. Those functions are modeled following the same principle: they return `true` if all invariants are correctly checked, otherwise they return `false`. In that case, they print a warning on standard output for each invalidated invariant through the call to function `print_warning`.

Function `print_warning` prints `str` if `correct` is `false`. It is called by `check_invariants` (see code chunk 196a).

194f ⟨*participants.ml* 193a⟩+≡ ◁194e 194g▷
```
let print_warning correct str =
  if not correct then Printf.printf "Participant base WARNING:\n  %s\n" str
```

Helper function `kind_as_string` returns the participant `kind` as a character string.

194g ⟨*participants.ml* 193a⟩+≡ ◁194f 195a▷
```
let kind_as_string kind =
  match kind with
  | Delegate -> "delegate"
  | Individual -> "individual"
```

Helper function `delegate_naming` returns `true` if the given `name` string corresponds to a delegate name.

195a ⟨*participants.ml* 193a⟩+≡ ◁194g 195b▷
```
let delegate_naming name =
  (String.length name >= 9)
    && (String.sub name 0 (String.length delegate_string_prefix)
          = delegate_string_prefix)
```

Function `login_naming_invariant` checks that a `login` starting with `"delegate_"` is classified as `Delegate` in `entry`, otherwise as `Individual`. `previously_correct` is the boolean value resulting from previous tests.

195b ⟨*participants.ml* 193a⟩+≡ ◁195a 195c▷
```
let login_naming_invariant print_warning login entry previously_correct =
  let correct =
    ((delegate_naming login) && entry.kind = Delegate)
    || (entry.kind = Individual) in
  print_warning correct
    (Printf.sprintf "naming invariant not verified for login:%s kind:%s"
       login (kind_as_string entry.kind));
  previously_correct && correct
```

Function `password_invariant` checks that the `password` of `entry` is not empty. It print a warning message if `print_warning` is true. `previously_correct` is the boolean value resulting from previous tests.

195c ⟨*participants.ml* 193a⟩+≡ ◁195b 195d▷
```
let password_invariant print_warning login entry previously_correct =
  let correct = entry.password != "" in
  print_warning correct
    (Printf.sprintf "non empty password invariant not verified for login:%s"
       login);
  previously_correct && correct
```

Function `login_id_invariant` checks that each login has an identifier.

195d ⟨*participants.ml* 193a⟩+≡ ◁195c 195e▷
```
let login_id_invariant print_warning login _entry previously_correct =
  let correct = Id.Participant.rev_mem !the_participant_base.ids login in
  print_warning correct (Printf.sprintf "login:%s without id" login);
  previously_correct && correct
```

Function `check_invariants_of_base` check all invariants for `base` and print warnings if `print_warning` is `true`.

195e ⟨*participants.ml* 193a⟩+≡ ◁195d 196a▷
```
let check_invariants_of_base print_warning base =
  let to_check login entry previously_correct =
    (login_naming_invariant print_warning login entry previously_correct)
      && (login_id_invariant print_warning login entry previously_correct)
      && (password_invariant print_warning login entry previously_correct) in
  Hashtbl.fold to_check base.attr true
```

Function `check_invariants` check all invariants on default Participant base `the_participant_base`. In case of invariant violation, the warnings are printed on standard output.

If we are not executing autotests, we do not take further action in case an invariant is invalid because we hope to recover from a crash, knowing which invariant is invalid and by manually modifying the saved databases.

196a ⟨*participants.ml* 193a⟩+≡ ◁195e 196b▷

```
let check_invariants () =
  if Config.check_invariants then
    if Config.do_autotests then
      assert(check_invariants_of_base print_warning !the_participant_base)
    else
      ignore(check_invariants_of_base print_warning !the_participant_base)
```

36.4 Participant base management

Helper function `remove_empty_groups` removes from `group_list` the groups having an empty name (i.e. empty string).

196b ⟨*participants.ml* 193a⟩+≡ ◁196a 196c▷

```
let remove_empty_groups group_list =
  List.filter (fun g -> g <> "") group_list
```

Function `max_id` returns the maximum identifier of participants in the base.

196c ⟨*participants.ml* 193a⟩+≡ ◁196b 196d▷

```
let max_id () = Id.Participant.length !the_participant_base.ids - 1
```

Function `details_of_id` returns kind, login, password and groups for participant of given `id`. It raises `Not_found` exception if such participant does not exist.

196d ⟨*participants.ml* 193a⟩+≡ ◁196c 197a▷

```
let details_of_id id =
  let login = Id.Participant.find !the_participant_base.ids id in
  let attr = Hashtbl.find !the_participant_base.attr login in
  (login, attr.kind, attr.password, attr.groups)
```

Function `add_participant` is used to add a new participant to `the_participant_base` with given `login`, `password` and `groups`. In case the participant is already in the database, an exception `Already_in_base` is raised. In case the `login` is badly formatted, exception `Norm.Invalid` is raised.

If the login of the participant starts with "`delegate_`", then the participant is considered a delegate, otherwise he is an individual.

197a ⟨*participants.ml* 193a⟩+≡ ◁196d 197b▷

```
exception Already_in_base

let add_participant raw_login password groups =
  let login = Norm.normalize_login raw_login in
  Norm.check_login login;
  let kind =
    if delegate_naming login then Delegate
    else Individual in
  if Id.Participant.rev_mem !the_participant_base.ids login then
    raise Already_in_base;
  let p = { kind = kind;
            password = md5_hash password;
            groups = remove_empty_groups groups; } in
  Hashtbl.replace !the_participant_base.attr login p;
  let id = Id.Participant.add !the_participant_base.ids login in
  check_invariants ();
  id
```

Function `update_participant` is used to update a participant record in `the_participant_base` with given `login`, `password` and `groups`. In case the participant is not in the database, exception `Not_found` is raised.

If the given password is different from the stored MD5 hash of the previous password, then the new password is stored as an MD5 hash. Otherwise, the old password is kept without doing an MD5 hash operation on it. The rationale behind this is to let the client read and write the password as a MD5 hash and only update it (i.e. redo a MD5 hash) if it is a new clear text password.

197b ⟨*participants.ml* 193a⟩+≡ ◁197a 198a▷

```
let update_participant login password groups =
  let kind =
    if delegate_naming login then Delegate
    else Individual in
  let id = Id.Participant.rev_find !the_participant_base.ids login in
  let p = Hashtbl.find !the_participant_base.attr login in
  if p.password <> password then (
    (* new password *)
    let p = { kind = kind;
              password = md5_hash password;
              groups = remove_empty_groups groups; } in
    Hashtbl.replace !the_participant_base.attr login p;
  ) else (
    (* same password *)
    let p = { kind = kind;
              password = password;
              groups = remove_empty_groups groups; } in
    Hashtbl.replace !the_participant_base.attr login p
   );
  Id.Participant.update_timestamp !the_participant_base.ids id;
  check_invariants ()
```

Function `remove_participant` deletes a participant of `the_participant_base`. Does nothing if the given `login` is not in the base.

198a ⟨*participants.ml* 193a⟩+≡ ◁197b 198b▷
```
let remove_participant login =
  Hashtbl.remove !the_participant_base.attr login;
  Id.Participant.rev_remove !the_participant_base.ids login;
  check_invariants ()
```

Function `change_password` replaces the password of participant identified by its `login` with the `new_password`. Its raises `Not_found` if the `login` is not found in the participant base.

198b ⟨*participants.ml* 193a⟩+≡ ◁198a 198c▷
```
let change_password login new_password =
  let p = Hashtbl.find !the_participant_base.attr login in
  p.password <- md5_hash new_password;
  Id.Participant.rev_update_timestamp !the_participant_base.ids login;
  check_invariants ()
```

Function `get_groups` returns the groups that the participant identified by `login` is in. It raises `Not_found` in case the participant is not found in the participant base.

198c ⟨*participants.ml* 193a⟩+≡ ◁198b 198d▷
```
let get_groups login =
  try
    let p = Hashtbl.find !the_participant_base.attr login in
    check_invariants ();
    p.groups
  with Not_found -> []
```

Function `set_groups` sets the `groups` that the participant identified by `login` is in. It raises `Not_found` in case the participant is not found in the participant base.

198d ⟨*participants.ml* 193a⟩+≡ ◁198c 198e▷
```
let set_groups login groups =
  let p = Hashtbl.find !the_participant_base.attr login in
  p.groups <- remove_empty_groups groups;
  Id.Participant.rev_update_timestamp !the_participant_base.ids login;
  check_invariants ()
```

Function `authenticate_participant` is used to identified a participant within the dem-exp server. If a participant with given `login` and `password` is found, then it is `Authenticated`, otherwise it remains `Anonymous`.

198e ⟨*participants.ml* 193a⟩+≡ ◁198d 198f▷
```
let authenticate_participant login password =
  if login = anonymous_as_string then Anonymous
  else try
    let p = Hashtbl.find !the_participant_base.attr login in
    if p.password = md5_hash password then
      match p.kind with
      | Individual -> Authenticated_individual login
      | Delegate -> Authenticated_delegate login
    else Anonymous
  with Not_found -> Anonymous
```

Function `authenticated_to_string` returns the authenticated participant `p` as a string.

198f ⟨*participants.ml* 193a⟩+≡ ◁198e 199a▷
```
let authenticated_to_string p =
  match p with
  | Anonymous -> anonymous_as_string
  | Authenticated_individual name -> name
  | Authenticated_delegate name -> name
```

Function `unsafe_authenticated_of_string` transforms a participant string login into its respective authenticated data structure. It is unsafe as the password is not checked.

199a ⟨*participants.ml* 193a⟩+≡ ◁198f 199b▷
```
let unsafe_authenticated_of_string s =
  match s with
  | name when name = anonymous_as_string -> Anonymous
  | name when delegate_naming name -> Authenticated_delegate name
  | name -> Authenticated_individual name
```

Auxiliary function `is_delegate` returns `true` if the given participant `name` is a delegate in the Participants base, or `false` if it is an individual or is not found.

199b ⟨*participants.ml* 193a⟩+≡ ◁199a 199c▷
```
let is_delegate name =
  try
    let p = Hashtbl.find !the_participant_base.attr name in
    match p.kind with
    | Delegate -> true
    | Individual -> false
  with
  | Not_found -> false
```

Function `initialize_default_participant_base` initializes the base of participants with an empty base containing only one participant: an administrator of name "`root`", of login "`demexp`" and belonging to the group `administration_group`.

199c ⟨*participants.ml* 193a⟩+≡ ◁199b 199d▷
```
let administration_group = "admin"

let classification_group = "classifier"

let make_empty_participant_base_with_root () =
  the_participant_base := create_participant_base ();
  ignore(add_participant "root" "demexp" [administration_group;
                                          classification_group]);
  check_invariants ()
```

Auxiliary function `is_administrator` returns `true` if the given authenticated participant `auth` has `administration_group` in its group.

199d ⟨*participants.ml* 193a⟩+≡ ◁199c 199e▷
```
let is_administrator auth =
  let check_admin_group name =
    let g = get_groups name in
    List.exists (fun a_group -> a_group = administration_group) g in
  match auth with
  | Anonymous -> false
  | Authenticated_delegate _ -> false
  | Authenticated_individual name -> check_admin_group name
```

Auxiliary function `is_classifier` returns `true` if the given authenticated participant `auth` has `classification_group` in its group.

199e ⟨*participants.ml* 193a⟩+≡ ◁199d 200a▷
```
let is_classifier auth =
  let check_classification_group name =
    let g = get_groups name in
    List.exists (fun a_group -> a_group = classification_group) g in
  match auth with
  | Anonymous -> false
  | Authenticated_delegate _ -> false
  | Authenticated_individual name -> check_classification_group name
```

36.5 XML support

Function `to_cduce_xml` produces a version of the Participant base compatible with needed CDuce data structure for XML export (see code chunk 258a).

200a ⟨*participants.ml* 193a⟩+≡ ◁199e 200b▷

```
let to_cduce_xml () =
  let participants =
    !(the_participant_base).ids in
  let to_list id (timestamp, login) accu =
    let _, kind, password, groups = details_of_id id in
    let xml_kind = match kind with
    | Delegate -> Xml.Delegate
    | Individual -> Xml.Individual in
    let p = { Xml.p_timestamp = Timestamp.to_string timestamp;
              Xml.kind = xml_kind;
              Xml.login = login;
              Xml.password = password;
              Xml.groups = groups } in
    (id, p) ::  accu in
  Id.Participant.fold to_list participants []
```

Function `of_cduce_xml` creates an new in memory Participant base from given `participant_list`.

200b ⟨*participants.ml* 193a⟩+≡ ◁200a 200c▷

```
let of_cduce_xml participant_list =
  let new_base = create_participant_base () in
  let convert (id, participant) =
    let ts = Timestamp.of_string participant.Xml.p_timestamp in
    Id.Participant.force new_base.ids id ts participant.Xml.login;
    let kind =
      match participant.Xml.kind with
      | Xml.Individual -> Individual
      | Xml.Delegate -> Delegate in
    let p_entry = { kind = kind;
                    password = participant.Xml.password;
                    groups = participant.Xml.groups; } in
    Hashtbl.add new_base.attr participant.Xml.login p_entry in
  List.iter convert participant_list;
  the_participant_base := new_base
```

Helper function `comparable_base` returns the Participant base as a data structure than can be easily compared with the equal operator. Useful for tests.

200c ⟨*participants.ml* 193a⟩+≡ ◁200b 201a▷

```
let comparable_base () =
  let base_to_lists base =
    let l1 =
      Id.Participant.fold (fun id login accu -> (id, login) :: accu)
        base.ids [] in
    let l2 =
      Hashtbl.fold (fun id e accu -> (id, e) :: accu) base.attr [] in
    (l1, l2) in
  base_to_lists !the_participant_base
```

36.6 Timestamps

Function `timestamp_list` returns the complete set of participant's timestamps in `block` as well as the `number` of them.

201a ⟨*participants.ml* 193a⟩+≡ ◁200c 201b▷

```
let timestamp_list () = Id.Participant.timestamp_list !the_participant_base.ids
```

Function `get_timestamp` returns the timestamp of participant `id`.

201b ⟨*participants.ml* 193a⟩+≡ ◁201a 202▷

```
let get_timestamp id =
  Id.Participant.get_timestamp !the_participant_base.ids id
```

36.7 Automatic tests

202 ⟨*participants.ml* 193a⟩+≡ ◁201b

```
let _ =
  if Config.do_autotests then begin
    Printf.printf "  participants autotests...";
    let login1 = "the_marvelous.login"
    and passwd1 = "and nonetheless marvelous password"
    and passwd2 = "another password" in
    (try
      update_participant login1 passwd1 ["a group"];
      assert(false)
    with Not_found -> assert(true));
    assert(max_id () = -1);
    ignore(add_participant login1 passwd1 []);
    (try
      ignore(add_participant login1 passwd1 []);
      assert(false);
    with Already_in_base -> assert(true));
    assert(max_id () = 0);
    assert(get_groups login1 = []);
    update_participant login1 passwd1 ["a group"];
    assert(max_id () = 0);
    assert(get_groups login1 = ["a group"]);
    set_groups login1 ["g";"";""];
    assert(get_groups login1 = ["g"]);
    assert(authenticate_participant login1 passwd1
            = Authenticated_individual login1);
    assert(authenticate_participant login1 passwd1
            = unsafe_authenticated_of_string login1);
    assert(authenticate_participant login1 "wrong password" = Anonymous);
    assert(authenticate_participant login1 passwd2
            = unsafe_authenticated_of_string anonymous_as_string);
    change_password login1 passwd2;
    assert(authenticate_participant login1 passwd1 = Anonymous);
    assert(authenticate_participant login1 passwd2
            = Authenticated_individual login1);
    remove_participant login1;
    assert(authenticate_participant login1 passwd2 = Anonymous);
    ignore(add_participant "delegate_a" "a" []);
    assert(authenticate_participant "delegate_a" "a"
            = unsafe_authenticated_of_string "delegate_a");
    assert(max_id () = 1);
    assert(is_delegate "delegate_a" = true);
    assert(is_delegate login1 = false);

    (* check import/export *)
    let old_base = comparable_base () in
    let saved_base = to_cduce_xml () in
    of_cduce_xml saved_base;
    assert(old_base = comparable_base ());

    (* check default base *)
    make_empty_participant_base_with_root ();
    assert(max_id () = 0);
    assert(authenticate_participant "root" "demexp"
            = Authenticated_individual "root");
    assert(authenticated_to_string (Authenticated_individual "root") = "root");
```

```
  assert(is_administrator (Authenticated_individual "root") = true);
  assert(is_classifier (Authenticated_individual "root") = true);
  assert(is_administrator (Authenticated_individual "toto") = false);
  assert(is_classifier (Authenticated_individual "toto") = false);
  assert(is_administrator Anonymous = false);
  Printf.printf "done\n"
end
```

Chapter 37

Classification

37.1 Classification overview

The aim of the *Classification* module is to provide needed support to organize information in the Position Base (see §41) and in the Delegation Base (see §39).

To each question in the Position base is attached a set of descriptive strings called *tags*. Each tag describes a concept, a certain category of the classification. Those concepts can be geographic (e.g. France, Rennes), can be a domain (e.g. Ecology), etc.

The Classification module contains a base of those tags. Each tag is identified by a unique natural number. The string of a tag is called its *label*.

Note: The `Id.Tag` module provides timestamps for each tag. For tag list associated to each question, we use the same timestamp as the one associated with question identifier in `Posbase` module.

todo: We do not check that a question id exists when tagging a question. The problem is that we have dependency posbase → vote → delegation → classification pref → classification. The needed question identifiers are defined in posbase module. We have a circular dependency (classification → posbase). One solution would be to move question identifiers outside posbase module (inside classification module?). TO DO

37.2 Definition of the Classification base

204a ⟨*classification.ml* 204a⟩≡ 204b ▷

```
(* copyright 2004 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Printf
```

`the_classification_base` contains:

- the base of `tags`, as defined in `Id` module;
- the set of tags attached to each question, `tagged_questions`, stored as a hash table from question descriptors to lists of tag identifiers.

204b ⟨*classification.ml* 204a⟩+≡ ◁ 204a 205a ▷

```
type classification_base = {
    tags : Id.Tag.t;
    tagged_questions : (int, int list) Hashtbl.t;
       (* question_descriptor |-> tag_id list *)
  }
```

We define the initial state of the Classification base.

205a ⟨*classification.ml* 204a⟩+≡ ◁204b 205b▷
```
let get_initial_classification_base () =
  { tags = Id.Tag.create ();
    tagged_questions = Hashtbl.create 3; }
```

Finally, we define the system wide classification base.

205b ⟨*classification.ml* 204a⟩+≡ ◁205a 205c▷
```
let the_classification_base : classification_base ref =
  ref (get_initial_classification_base ())
```

37.3 Operations on classification tags

Function `max_id` returns the maximum identifier of participants in the base.

205c ⟨*classification.ml* 204a⟩+≡ ◁205b 205d▷
```
let max_id () = Id.Tag.length !the_classification_base.tags - 1
```

Operation `create_tag` adds a new tag into the classification base and returns its identifier. It raises `Id.Tag.Already_exists` in case the `tag_label` has already been defined.

205d ⟨*classification.ml* 204a⟩+≡ ◁205c 205e▷
```
let create_tag ~tag_label =
  Id.Tag.add !the_classification_base.tags tag_label
```

Operation `check_tag_id` raises `Not_found` exception if the `tag_id` is not found in the base, otherwise do nothing.

205e ⟨*classification.ml* 204a⟩+≡ ◁205d 205f▷
```
let check_tag_id ~tag_id =
  ignore(Id.Tag.find !the_classification_base.tags tag_id)
```

Operation `get_tag_label` returns the tag label for `tag_id`. It raises `Not_found` if `tag_id` is not found.

205f ⟨*classification.ml* 204a⟩+≡ ◁205e 205g▷
```
let get_tag_label ~tag_id =
  check_tag_id ~tag_id;
  Id.Tag.find !the_classification_base.tags tag_id
```

Operation `get_tag_label` returns the tag id for `tag_label`. It raises `Not_found` if `tag_label` is not found.

205g ⟨*classification.ml* 204a⟩+≡ ◁205f 205h▷
```
let get_tag_id ~tag_label =
  Id.Tag.rev_find !the_classification_base.tags tag_label
```

Operation `change_tag_label` changes the label of a tag from its `tag_id`. It raises `Not_found` exception in case the `tag_id` is not found.

205h ⟨*classification.ml* 204a⟩+≡ ◁205g 205i▷
```
let change_tag_label ~tag_id ~tag_label =
  Id.Tag.replace !the_classification_base.tags tag_id tag_label;
  Id.Tag.update_timestamp !the_classification_base.tags tag_id
```

Operation `get_tag_id_list` returns a list of all tag identifiers available in the base.

205i ⟨*classification.ml* 204a⟩+≡ ◁205h 206a▷
```
let get_tag_list () =
  let collect_tag_id id (ts, lbl) l =
    (id, (ts, lbl)) :: l in
  Id.Tag.fold collect_tag_id !the_classification_base.tags []
```

37.4 Operations related to question tagging

Operation `tag_question` adds a new `tag_id` to question identifier `q_id`. It raises `Not_found` if the `tag_id` does not exists. It does nothing if the tag is already associated to the question.

206a ⟨*classification.ml* 204a⟩+≡ ◁205i 206b▷

```
let tag_question ~q_id ~tag_id =
  check_tag_id ~tag_id;
  try
    let l = Hashtbl.find !the_classification_base.tagged_questions q_id in
    if not (List.mem tag_id l) then
      Hashtbl.replace !the_classification_base.tagged_questions q_id (tag_id :: l)
  with Not_found ->
    Hashtbl.add !the_classification_base.tagged_questions q_id [tag_id]
```

Operation `untag_question` removes `tag_id` from question identifier `q_id`. It raises `Not_found` if the tag id is an invalid tag. If the question descriptor does not exist or if the `tag_id` was not put on question descriptor, it does nothing.

206b ⟨*classification.ml* 204a⟩+≡ ◁206a 206c▷

```
let untag_question ~q_id ~tag_id =
  check_tag_id ~tag_id;
  try
    let l = Hashtbl.find !the_classification_base.tagged_questions q_id in
    let new_l, _ = List.partition (fun id -> id <> tag_id) l in
    if new_l = [] then
      Hashtbl.remove !the_classification_base.tagged_questions q_id
    else
      Hashtbl.replace !the_classification_base.tagged_questions q_id new_l
  with
    Not_found -> ()
```

Operation `get_question_tags` returns the list of tag ids put on question identifier `q_id`. In case it does not exist in the base, the returned list is empty.

206c ⟨*classification.ml* 204a⟩+≡ ◁206b 206d▷

```
let get_question_tags ~q_id =
  try
    Hashtbl.find !the_classification_base.tagged_questions q_id
  with Not_found -> []
```

Operation `get_question_tags` returns the list of tag labels put on question identifier `q_id`. In case it does not exist in the base, the returned list is empty.

206d ⟨*classification.ml* 204a⟩+≡ ◁206c 206e▷

```
let get_question_tags_as_label ~q_id =
  try
    let tags = Hashtbl.find !the_classification_base.tagged_questions q_id in
    List.map (fun tag_id -> get_tag_label ~tag_id) tags
  with Not_found -> []
```

Operation `get_questions_matching_tags` returns the list of question descriptors that are tagged with all tags in `tags` set.

206e ⟨*classification.ml* 204a⟩+≡ ◁206d 207a▷

```
let get_questions_matching_tags tags =
  let matches_tags tag_set =
    List.for_all (fun t -> List.mem t tag_set) tags in
  let add_if_match q_id tag_set l =
    if matches_tags tag_set then q_id :: l else l in
  Hashtbl.fold add_if_match !the_classification_base.tagged_questions []
```

37.5 Classification base reset

Operation `initialize` resets the Classification base to its initial state.

207a ⟨*classification.ml* 204a⟩+≡ ◁206e 207b▷
```
let initialize () =
  the_classification_base := get_initial_classification_base ()
```

37.6 XML support

Function `tags_to_cduce_xml` and `tagged_questions_to_cduce_xml` produces a version of the tags in Classification base compatible with needed CDuce data structure for XML export (see code chunk 258a).

207b ⟨*classification.ml* 204a⟩+≡ ◁207a 207c▷
```
let tags_to_cduce_xml () =
  let to_list id (ts, tag) accu =
    (id, Timestamp.to_string ts, tag) :: accu in
  Id.Tag.fold to_list !the_classification_base.tags []
```

207c ⟨*classification.ml* 204a⟩+≡ ◁207b 207d▷
```
let tagged_questions_to_cduce_xml () =
  let to_list id id_list accu = (id, id_list) :: accu in
  Hashtbl.fold to_list !the_classification_base.tagged_questions []
```

Function `classification_of_cduce_xml` builds the in memory Classification base from two parameters: `tag_list`, a list of tags with their identifiers, and `tagged_questions`, a list of question identifiers with the set of tag identifiers assign to them.

207d ⟨*classification.ml* 204a⟩+≡ ◁207c 207e▷
```
let of_cduce_xml tag_list tagged_questions =
  let tag_list_int32 =
    let map_fun (a, str_ts, b) = (a, Timestamp.of_string str_ts, b) in
    List.map map_fun tag_list in
  let tags = Id.Tag.of_list tag_list_int32 in
  let tagged_q = Hashtbl.create 3 in
  List.iter (fun (id, taglist) -> Hashtbl.add tagged_q id taglist)
    tagged_questions;
  the_classification_base := { tags = tags;
                               tagged_questions = tagged_q; }
```

Helper function `comparable_base` returns the Classification base as a data structure than can be easily compared with the equality operator. Used for tests.

207e ⟨*classification.ml* 204a⟩+≡ ◁207d 208a▷
```
let comparable_base () =
  let base_to_lists base =
    let l1 =
      let fold_fun id (ts, tag) accu = (id, ts, tag) :: accu in
      Id.Tag.fold fold_fun base.tags [] in
    let l2 =
      Hashtbl.fold (fun id l accu -> (id, l) :: accu)
        base.tagged_questions [] in
    (l1, l2) in
  base_to_lists !the_classification_base
```

37.7 Timestamps

Function `timestamp_list` returns the complete set of tag's timestamps in `block` as well as the `number` of them.

208a ⟨*classification.ml* 204a⟩+≡ ◁207e 208b▷

```
let tag_timestamp_list () = Id.Tag.timestamp_list !the_classification_base.tags
```

Function `get_tag_timestamp` returns the timestamp of tag `id`.

208b ⟨*classification.ml* 204a⟩+≡ ◁208a 209▷

```
let get_tag_timestamp ~tag_id =
  Id.Tag.get_timestamp !the_classification_base.tags tag_id
```

37.8 Automatic tests

209 ⟨*classification.ml* 204a⟩+≡ ◁208b

```
  let _ =
    if Config.do_autotests then begin
      printf "  classification autotests...";
      (* tests on tags *)
      assert(create_tag "A" = 0);

      let id0 = 0 in
      change_tag_label id0 "B";
      assert(get_tag_label id0 = "B");
      assert(get_tag_id "B" = id0);

      (* tests on questions & tags *)
      let qid_base = Id.Question.create () in
      let q_desc = "Question 1" in
      let q_id = Id.Question.add qid_base q_desc in
      let tag1 = create_tag "Tag 1" in
      let tag2 = create_tag "Tag 2" in
      assert(get_question_tags ~q_id = []);
      tag_question ~q_id ~tag_id:tag1;
      assert(get_question_tags ~q_id = [tag1]);
      assert(get_question_tags_as_label ~q_id = ["Tag 1"]);
      tag_question ~q_id ~tag_id:tag2;
      assert(get_question_tags ~q_id = [tag2; tag1]);
      assert(get_question_tags_as_label ~q_id = ["Tag 2"; "Tag 1"]);
      untag_question ~q_id ~tag_id:tag1;
      assert(get_question_tags ~q_id = [tag2]);
      assert(get_question_tags_as_label ~q_id = ["Tag 2"]);
      untag_question ~q_id ~tag_id:tag2;
      assert(get_question_tags ~q_id = []);
      assert(not (Hashtbl.mem !the_classification_base.tagged_questions q_id));

    (* following test is not doable, due to question id checking *)
    (*     untag_question ~q_id:"unavailable question" ~tag_id:tag1; *)

      (* tag id list *)
      assert(List.sort compare
               (List.map (fun (a, (_, b)) -> (a, b)) (get_tag_list ()))
               = [(id0, "B"); (tag1, "Tag 1"); (tag2, "Tag 2")]);

      (* get questions from tags *)
      let qa = "Question A"
      and qb = "Question B" in
      let q_ida = Id.Question.add qid_base qa in
      let q_idb = Id.Question.add qid_base qb in
      tag_question q_ida tag1;
      tag_question q_ida tag1; (* does nothing, tag already added *)
      tag_question q_idb tag1;
      tag_question q_idb tag2;
      assert(List.sort compare (get_questions_matching_tags [tag1])
               = [q_ida; q_idb]);
      assert(List.sort compare (get_questions_matching_tags [tag2])
               = [q_idb]);
      assert(List.sort compare (get_questions_matching_tags [tag1; tag2])
               = [q_idb]);
      assert(List.sort compare (get_questions_matching_tags [id0])
```

```
          = []);

  (* check import/export functions *)
  let old_base = comparable_base () in
  let saved_tags = tags_to_cduce_xml ()
  and saved_tagged_questions = tagged_questions_to_cduce_xml () in
  of_cduce_xml saved_tags saved_tagged_questions;
  assert(old_base = comparable_base ());

  (* reset base and check the previous tag is correctly lost *)
  initialize ();
  (try
    ignore(Id.Tag.find !the_classification_base.tags 0);
    assert(false)
  with Not_found -> assert(true));
  printf "done\n"
end
```

Chapter 38

Classification preferences

Module `Cpref` defines the handling of classification individual preferences for each participant.

38.1 Introduction

The classification allows to add to each question a set of *tags*. Within the Delegation sub-system, it is possible to let all the questions marked with a given tag to be handled by a delegate.

Now, suppose we have a question q marked with two tags t_1 and t_2. A participant delegates all questions having the tag t_1 to delegate D_1 and all questions having tag t_2 to delegate D_2. To whose delegate is delegated question q? The participant has to disambiguate this case by saying which tag, and therefore which delegate, takes precedence over the other one (for example, t_1 over t_2). By generalizing such a hierarchy between tags, we create, for each participant, a personnal hierarchy between tags, i.e. a *classification preference*.

The purpose of the `Cpref` (classification preference) module is to manage the tag hierarchy of each participant. This module arranges tags into a set of trees, the tags being upper into a tree taking precedence over tags being lower. For example, in figure 38.1, tag A takes precedence over tags C and D but there is no ordering between tags A and B.

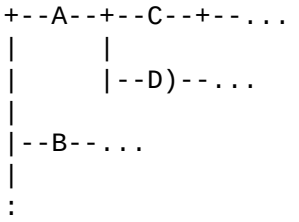

Figure 38.1: A tag hierarchy

38.2 Data structure

The main purpose of the tree is to check the relationship between two given tags. So the check that a tag is available in the tree and its position should be as fast as possible. We therefore use a hash table to store the tree. The key of the table is a tag. The associated information to the key is the list of sub-tags (i.e. sub-nodes) as well as the parent tag.

For example, the tree of figure 38.1 is represented as in figure 38.2. Getting the position of the tree is a matter of climbing up parents and getting the relative position of each parent. This a

$\log(n)$ operation. We currently use a list to store the sons of a tag, under the assumption this list is rather short.

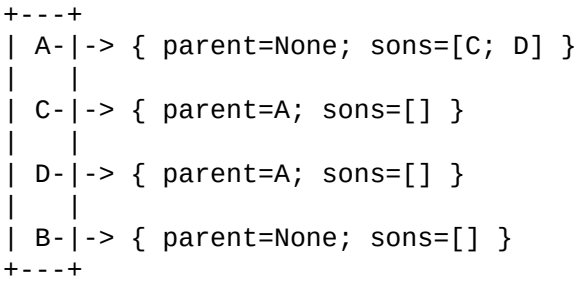

Figure 38.2: Storing of a tree as a hash table

212a ⟨*cpref.ml* 212a⟩≡ 212b ▷
```
(* copyright 2004 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Printf
```

An entry in the preference is associated to its optional `parent` and the list of `sons`.

212b ⟨*cpref.ml* 212a⟩+≡ ◁212a 212c ▷
```
type entry = {
    parent : int option;
    sons : int list;
  }
```

A classification preference is a hash table associating tag-ids to entries.

212c ⟨*cpref.ml* 212a⟩+≡ ◁212b 212d ▷
```
type t = (int, entry) Hashtbl.t
```

38.3 Tag manipulation in classification preference

Exception `Tag_id_not_found` is raised when a given tag-id does not exists. Exception `Tag_id_already_exists` is raised on the contrary.

212d ⟨*cpref.ml* 212a⟩+≡ ◁212c 212e ▷
```
exception Tag_id_not_found of int
exception Tag_id_already_exists of int
```

In the following functions, we denote `cpref` a classification preference and `tag_id` a tag-id. Function `create` returns a fresh classification preference.

212e ⟨*cpref.ml* 212a⟩+≡ ◁212d 212f ▷
```
let create () = Hashtbl.create 3
```

Helper function `add_to_parent` adds `tag_id` in the list of sons of `parent`.

212f ⟨*cpref.ml* 212a⟩+≡ ◁212e 213a ▷
```
let add_to_parent cpref parent tag_id =
  let pe = Hashtbl.find cpref parent in
  let new_pe = { parent = pe.parent; sons = tag_id :: pe.sons } in
  Hashtbl.replace cpref parent new_pe
```

Function `add` allows to add a `tag_id` under another optional tag of id `under`. If no parent tag is given, `tag_id` is added without parent into the classification preference. If given, `under` should be in the classification preference otherwise it raises `Tag_id_not_found`. `tag_id` should not be in the preference, otherwise it raises `Tag_id_already_exists`.

If a `parent` to add under is given, this parent is look for and the `tag_id` is added in the list of sons of the parent.

213a ⟨*cpref.ml* 212a⟩+≡ ◁212f 213b▷

```
let add ~cpref ?under ~tag_id () =
  if Hashtbl.mem cpref tag_id then
    raise (Tag_id_already_exists(tag_id));
  match under with
  | Some parent ->
      if not(Hashtbl.mem cpref parent) then
        raise (Tag_id_not_found(parent));
      let e = { parent = Some parent; sons = [] } in
      add_to_parent cpref parent tag_id;
      Hashtbl.add cpref tag_id e
  | None ->
      let e = { parent = None; sons = [] } in
      Hashtbl.add cpref tag_id e
```

Helper function `remove_from_parent` remove the `tag_id` in the list of sons of the given `parent`.

213b ⟨*cpref.ml* 212a⟩+≡ ◁213a 213c▷

```
let remove_from_parent cpref parent tag_id =
  let pe = Hashtbl.find cpref parent in
  let sons_wo_e = List.filter (fun t -> t <> tag_id) pe.sons in
  let new_pe = { parent = pe.parent; sons = sons_wo_e; } in
  Hashtbl.replace cpref parent new_pe
```

Function `remove` removes the `tag_id` as well as all of its sons. It raises `Tag_id_not_found` if `tag_id` does not exists.

If the `tag_id` has a `parent`, it is removed from its parent's list of sons.

213c ⟨*cpref.ml* 212a⟩+≡ ◁213b 214a▷

```
let remove ~cpref ~tag_id =
  let rec remove_sons sons =
    let remove_a_sun s =
      let se = Hashtbl.find cpref s in
      Hashtbl.remove cpref s;
      remove_sons se.sons in
    List.iter remove_a_sun sons in
  if not(Hashtbl.mem cpref tag_id) then
    raise (Tag_id_not_found(tag_id));
  let e = Hashtbl.find cpref tag_id in
  match e.parent with
  | Some parent ->
      remove_from_parent cpref parent tag_id;
      remove_sons [tag_id]
  | None ->
      remove_sons [tag_id]
```

Function `move` moves the `tag_id` and its sons from its current parent (if it exists) to the new parent (`under`).

Note: `move` does not work if one tag id is already, directly or indirectly, the son of the other. With this restriction, it is impossible to change the already set preferences.

214a ⟨*cpref.ml* 212a⟩+≡ ◁213c 214b▷

```
let move ~cpref ~tag_id ~under =
  let change_parent entry tag_id new_p =
    let new_entry = { parent = Some new_p; sons = entry.sons } in
    Hashtbl.replace cpref tag_id new_entry in
  if not(Hashtbl.mem cpref tag_id) then
    raise (Tag_id_not_found(tag_id));
  if not(Hashtbl.mem cpref under) then
    raise (Tag_id_not_found(under));
  let e = Hashtbl.find cpref tag_id in
  match e.parent with
  | Some parent ->
      remove_from_parent cpref parent tag_id;
      add_to_parent cpref under tag_id;
      change_parent e tag_id under
  | None ->
      add_to_parent cpref under tag_id;
      change_parent e tag_id under
```

38.4 Comparisons between tags

Function `as_parent_list` returns the list of parents of `tag_id`, from most upper parent downto `tag_id`. It returns the empty list if `tag_id` is not in `cpref`.

For example, in figure 38.1, we have following parent lists:

D: `[A; D]`

A: `[A]`

B: `[B]`

E: `[]` (E in not in the tree)

214b ⟨*cpref.ml* 212a⟩+≡ ◁214a 214c▷

```
let as_parent_list ~cpref ~tag_id =
  let rec as_pl_aux path tag_id =
    if not(Hashtbl.mem cpref tag_id) then
      path
    else
      let e = Hashtbl.find cpref tag_id in
      match e.parent with
      | None -> tag_id :: path
      | Some parent -> as_pl_aux (tag_id :: path) parent in
  as_pl_aux [] tag_id
```

Helper function `is_fst_prefix_of_snd` returns true if the list `fst` argument is a prefix of `snd` (e.g. `[0; 3]` is a prefix of `[0; 3; 1]`).

214c ⟨*cpref.ml* 212a⟩+≡ ◁214b 215a▷

```
let rec is_fst_prefix_of_snd fst snd =
  match fst, snd with
  | [], _ -> true
  | _, [] -> false
  | a :: fst_tail, b :: snd_tail ->
      if a = b then is_fst_prefix_of_snd fst_tail snd_tail
      else false
```

We define the type `preferred` that allows us to decide if a tag is preferred over another one with following meaning:

1. `[No_relation]` A and B are present but there is no relationship between them;
2. `[A_preferred]` A is preferred over B;
3. `[B_preferred]` B is preferred over A.

215a ⟨*cpref.ml* 212a⟩+≡ ◁214c 215b▷

```
type preferred =
  | No_relation
  | A_preferred
  | B_preferred
```

Function `which_preferred` implements above comparison.

215b ⟨*cpref.ml* 212a⟩+≡ ◁215a 215c▷

```
let which_preferred ~cpref ~a ~b =
  let pl_a = as_parent_list ~cpref ~tag_id:a
  and pl_b = as_parent_list ~cpref ~tag_id:b in
  if pl_a = [] or pl_b = [] then No_relation
  else
    if is_fst_prefix_of_snd pl_a pl_b then A_preferred
    else
      if is_fst_prefix_of_snd pl_b pl_a then B_preferred
      else No_relation
```

Function `prefer` stores in `cpref` that tag id `tag_id` is preferred to tag id `preferred_to`. In case one of the tag id does not exists in `cpref`, it is first added to it.

215c ⟨*cpref.ml* 212a⟩+≡ ◁215b 216▷

```
let prefer ~cpref ~tag_id ~preferred_to =
  if not(Hashtbl.mem cpref tag_id) then (
    add ~cpref ~tag_id ();
    move ~cpref ~tag_id:preferred_to ~under:tag_id
   )
  else
    if not(Hashtbl.mem cpref preferred_to) then
      add ~cpref ~tag_id:preferred_to ~under:tag_id ()
    else
      move ~cpref ~tag_id:preferred_to ~under:tag_id
```

38.5 Automatic tests

216 ⟨*cpref.ml* 212a⟩+≡ ◁215c

```
let _ =
  if Config.do_autotests then begin
    printf "  cpref autotests...";
    (* handling of classification preference *)
    let cpref = create () in
    let a = Classification.create_tag "a" in
    let b1 = Classification.create_tag "b1" in
    let b2 = Classification.create_tag "b2" in
    let c1 = Classification.create_tag "c1" in
    let c2 = Classification.create_tag "c2" in
    let d = Classification.create_tag "d" in

    add ~cpref ~tag_id:a ();
    assert(Hashtbl.find cpref a = { parent=None; sons=[] });
    assert(as_parent_list ~cpref ~tag_id:a = [a]);
    assert(as_parent_list ~cpref ~tag_id:b1 = []);

    add ~cpref ~tag_id:d ();
    assert(Hashtbl.find cpref d = { parent=None; sons=[] });

    add ~cpref ~tag_id:b1 ~under:a ();
    assert(Hashtbl.find cpref a = { parent=None; sons=[b1] });
    assert(Hashtbl.find cpref b1 = { parent=Some a; sons=[] });
    assert(as_parent_list ~cpref ~tag_id:a = [a]);
    assert(as_parent_list ~cpref ~tag_id:b1 = [a; b1]);
    assert(which_preferred ~cpref ~a:a ~b:b1 = A_preferred);
    assert(which_preferred ~cpref ~a:b1 ~b:a = B_preferred);
    assert(which_preferred ~cpref ~a:b1 ~b:d = No_relation);

    add ~cpref ~tag_id:c1 ~under:b1 ();
    add ~cpref ~tag_id:c2 ~under:b1 ();
    assert(Hashtbl.find cpref b1 = { parent=Some a; sons=[c2; c1] });
    add ~cpref ~tag_id:b2 ~under:a ();
    assert(Hashtbl.find cpref a = { parent=None; sons=[b2; b1] });

    remove ~cpref ~tag_id:b1;
    assert(Hashtbl.find cpref a = { parent=None; sons=[b2] });
    assert(Hashtbl.find cpref b2 = { parent=Some a; sons=[] });
    assert(Hashtbl.find cpref d = { parent=None; sons=[] });
    assert(Hashtbl.mem cpref b1 = false);
    assert(Hashtbl.mem cpref c1 = false);
    assert(Hashtbl.mem cpref c2 = false);
    assert(which_preferred ~cpref ~a:a ~b:b2 = A_preferred);
    assert(which_preferred ~cpref ~a:a ~b:d = No_relation);
    assert(which_preferred ~cpref ~a:b2 ~b:d = No_relation);

    remove ~cpref ~tag_id:d;
    assert(Hashtbl.find cpref a = { parent=None; sons=[b2] });
    assert(Hashtbl.find cpref b2 = { parent=Some a; sons=[] });
    assert(Hashtbl.mem cpref d = false);

    (* test error cases *)
    (try
      remove ~cpref ~tag_id:c1;
      assert(false)
```

```
  with Tag_id_not_found t -> assert(t = c1));
  (try
    add ~cpref ~tag_id:a ();
    assert(false);
  with Tag_id_already_exists t -> assert(t = a));

  (* test move *)
  add ~cpref ~tag_id:c1 ~under:b2 ();
  add ~cpref ~tag_id:d ();
  assert(Hashtbl.find cpref a = { parent=None; sons=[b2] });
  assert(Hashtbl.find cpref b2 = { parent=Some a; sons=[c1] });
  assert(Hashtbl.find cpref c1 = { parent=Some b2; sons=[] });
  assert(Hashtbl.find cpref d = { parent=None; sons=[] });
  move ~cpref ~tag_id:b2 ~under:d;
  assert(Hashtbl.find cpref a = { parent=None; sons=[] });
  assert(Hashtbl.find cpref b2 = { parent=Some d; sons=[c1] });
  assert(Hashtbl.find cpref c1 = { parent=Some b2; sons=[] });
  assert(Hashtbl.find cpref d = { parent=None; sons=[b2] });

  (* test preference setting *)
  (* 1. both are in the cpref *)
  prefer ~cpref ~tag_id:a ~preferred_to:d;
  assert(Hashtbl.find cpref a = { parent=None; sons=[d] });
  assert(Hashtbl.find cpref b2 = { parent=Some d; sons=[c1] });
  assert(Hashtbl.find cpref c1 = { parent=Some b2; sons=[] });
  assert(Hashtbl.find cpref d = { parent=Some a; sons=[b2] });

  (* 2. preferred_to isn't in cpref *)
  remove ~cpref ~tag_id:d;
  prefer ~cpref ~tag_id:a ~preferred_to:d;
  assert(Hashtbl.find cpref a = { parent=None; sons=[d] });
  assert(Hashtbl.find cpref d = { parent=Some a; sons=[] });

  (* 3. tag_id isn't in cpref *)
  remove ~cpref ~tag_id:d;
  prefer ~cpref ~tag_id:d ~preferred_to:a;
  assert(Hashtbl.find cpref a = { parent=Some d; sons=[] });
  assert(Hashtbl.find cpref d = { parent=None; sons=[a] });

  (* reset everything *)
  Classification.initialize ();
  printf "done\n"
end
```

Chapter 39

Delegation

39.1 Delegation overview

39.2 Definition of Delegation base

218a ⟨*delegation.ml* 218a⟩≡ 218b ▷

```
(* copyright 2003 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)
```

218b ⟨*delegation.ml* 218a⟩+≡ ◁218a 218c ▷

```
open Printf
open Participants
open Classification
```

A `delegation` is made of a `delegator` pointing to a `delegate` for a given `domain`.
A `delegation_base` stores a set of delegations. It uses two hash tables:

- the `to_link` hash tables that stores a correspondance between a delegator towards its list of (domain of delegation, delegation) couples;
- the `from_link` hash table that stores a correspondance between a delegate towards its list of (domain of delegation, delegation) couples.

In other words, the `to_link` table stores forward delegation links and the `from_link` table stores backward delegation links.

39.3 Automatic tests

218c ⟨*delegation.ml* 218a⟩+≡ ◁218b

```
let _ =
  if Config.do_autotests then begin
    printf "  delegation autotests...";
    printf "done\n"
  end
```

Chapter 40

Voting

40.1 General overview of voting algorithm

40.2 Voting code

219a ⟨*voting.ml* 219a⟩≡ 219b ▷

```
(* copyright 2003 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)
```

Modules needed in later code:

219b ⟨*voting.ml* 219a⟩+≡ ◁219a 219c ▷

```
open Printf
```

40.3 Pairwise matrices

We can make an empty matrix (i.e. full of 0) of `size`.

219c ⟨*voting.ml* 219a⟩+≡ ◁219b 220a ▷

```
let empty_pairwise_matrix ~size =
  assert(size >= 0);
  Array.make_matrix size size 0
```

`vote_to_pairwise_matrix` transforms a `vote` expressed as a list of integers indicating each response chosen into a pairwise matrix of size `size`.

> **Assumption**: Each integer v in the `vote` list is in the valid range of responses, i.e. $0 \leq v <$`size`.

The used algorithm is as following:

- we create an empty matrix `m` of `size` rows $\times$ `size` columns
- we consider in turn each element `v` of the `vote` list and the list `above_rank` of responses already considered. For each element `v`:
  - for all possible responses, if the considered response `j` is in not in the list `above_rank` then the vote `v` is ranked above this reponse `j` and a 1 is added into the matrix `m`

220a ⟨*voting.ml* 219a⟩+≡ ◁219c 220b▷

```
let vote_to_pairwise_matrix ~size ~vote =
  assert(size >= 0);
  let m = empty_pairwise_matrix size in
  let put_vote m v above_rank =
    for j = 0 to size - 1 do
      if j <> v then
        if not (List.mem j above_rank) then m.(v).(j) <- 1;
    done in
  let rec rank_choices m l above_rank =
    match l with
      [] -> m
    | v :: other_votes ->
        put_vote m v above_rank;
        rank_choices m other_votes (v :: above_rank) in
  rank_choices m vote []
```

220b ⟨*voting.ml* 219a⟩+≡ ◁220a 220c▷

```
let print_matrix ~mat =
  let len = Array.length mat in
  printf "      |";
  for i = 0 to len - 1 do
    printf "%6d" i
  done;
  printf "\n       ";
  for i = 0 to len - 1 do printf "______" done;
  printf "\n";
  for i = 0 to len - 1 do
    for j = 0 to len - 1 do
      if j = 0 then printf "%6d|" i;
      printf "%6d" mat.(i).(j)
    done;
    printf "\n"
  done
```

Function `add_to_matrix` add the content of matrix `src` to matrix `dst`.

220c ⟨*voting.ml* 219a⟩+≡ ◁220b 221a▷

```
let add_to_matrix ~dst ~src =
  assert(Array.length dst = Array.length src
           && Array.length dst.(0) = Array.length dst.(0));
  for i = 0 to Array.length dst - 1 do
    for j = 0 to Array.length dst.(0) - 1 do
      dst.(i).(j) <- dst.(i).(j) + src.(i).(j)
    done
  done
```

Function `multiply_matrix` multiply each value in matrix `dst` by integer `by`.

221a ⟨*voting.ml* 219a⟩+≡ ◁220c 221b▷

```
let multiply_matrix ~dst ~by =
  for i = 0 to Array.length dst - 1 do
    for j = 0 to Array.length dst.(0) - 1 do
      dst.(i).(j) <- dst.(i).(j) * by
    done
  done
```

40.4 Winner determination

The simplest case to consider is when a response wins over all responses in a one-to-one match. The `unambiguous_winner` code returns, for a given pairwise matrix `mat`, the number of this winner if it exists, with a `majority`, or raise `Not_found` otherwise.

Assumption: The matrix `mat` given as argument is square. `majority`≥ 1

The used algorithm is as follow:

- We consider in turn each response (i.e. row in the matrix) `i` through `search_winner`. For each response:
  - we check, using `winner_over_all`, that response `i` wins over all other responses `j` with `majority`.

221b ⟨*voting.ml* 219a⟩+≡ ◁221a 222a▷

```
let unambiguous_winner ?(majority = 1) mat =
  assert(Array.length mat = 0
       || Array.length mat = Array.length mat.(0));
  assert(majority >= 1);
  let num_resp = Array.length mat in
  let rec winner_over_all i j =
    if j < num_resp then
      begin
        if i <> j then
          mat.(i).(j) > majority * mat.(j).(i) && winner_over_all i (j + 1)
        else winner_over_all i (j + 1)
      end
    else true in
  let rec search_winner i =
    if i < num_resp then
      begin
        if winner_over_all i 0 then i
        else search_winner (i + 1)
      end
    else raise Not_found in
  search_winner 0
```

To solve ambiguities in an ambiguous pairwise matrix, the available algorithms are using an ordered list of defeats. The following code produces such a list of triple (winner, loser, magnitude of defeat).

Assumption: The matrix `mat` given as argument is square.

Use algorithm is quite simple: we parse half of the pairwise matrix (avoiding the diagonal) and for each match between responses `i` and `j` we store the loser and the winner in a list `l`, with the score of the winner. Before returning this list `l`, we sort it.

222a ⟨*voting.ml* 219a⟩+≡ ◁221b 222b▷

```
let pairwise_matrix_to_ordered_defeats ~mat =
  assert(Array.length mat = 0
       || Array.length mat = Array.length mat.(0));
  let len = Array.length mat in
  let l = ref [] in
  for i = 1 to len - 1 do
    for j = 0 to i do
      if i <> j then
        begin
          if mat.(i).(j) > mat.(j).(i) then
            l := (i, j, mat.(i).(j)) :: !l
          else if mat.(i).(j) < mat.(j).(i) then
            l := (j, i, mat.(j).(i)) :: !l
          else
            () (* in case of tie, we don't include them *)
        end
    done
  done;
  List.sort (fun (_, _, s1) (_, _, s2) -> compare s1 s2) !l
```

The function `make_all_candidates` is used to build the list of all candidates, which is the list of integers from 0 to `size`-1.

222b ⟨*voting.ml* 219a⟩+≡ ◁222a 222c▷

```
let make_all_candidates ~size =
  assert(size >= 0);
  let rec make_aux n l =
    if n >= 0 then make_aux (n - 1) (n :: l)
    else l in
  make_aux (size - 1) []
```

The `get_unbeaten` function gives the list of candidates (of number `size`) that are never beaten in a list of `defeats`.

It works be recursively traversing the list of `defeats` and removing in the set of potential winners `winner_set` the elements that have be found to be a `loser` in a match.

222c ⟨*voting.ml* 219a⟩+≡ ◁222b 223▷

```
let get_unbeaten ~size ~defeats =
  assert(size >= 0);
  let rec f defeats winner_set =
    match defeats with
    | [] -> winner_set
    | (_, loser, _) :: tail ->
        let updated_winner_set =
          List.filter (fun e -> e <> loser) winner_set in
        f tail updated_winner_set in
  f defeats (make_all_candidates ~size)
```

40.5 Condorcet ambiguity resolution

The first approach to find a winner in an ambiguous pairwise matrix is to drop the weakest (i.e. smallest magnitude) defeat until one of the candidate is unbeaten. The following code applies this algorithm, given a pairwise matrix `mat`.

The used algorithm is as follow:

- we build the ordered list of `defeats`
- in this list, if we can find a unique candidate (using `get_unbeaten`) which is not beaten by others, then we have our winner. Otherwise, we remove the head element on the `defeat` list (i.e. the weakest defeat) and continue recursively through `find_winner`.

`condorcet_winner` raises `Not_found` exception only if pairwise matrix `mat` is of zero size. In that case, no winner can be found.

223 ⟨*voting.ml* 219a⟩+≡ ◁222c 224▷

```
let condorcet_winner ~mat =
  let defeats = pairwise_matrix_to_ordered_defeats ~mat in
  let size = Array.length mat in
  assert(size >= 0);
  if size = 0 then raise Not_found;
  let rec find_winner defeats =
    match defeats with
    | [] -> assert(false) (* this case should never happen *)
    | _ :: defeats_tail ->
        match get_unbeaten ~size ~defeats with
        | [e] -> e
        | _ -> find_winner defeats_tail in
  find_winner defeats
```

40.6 Schwartz Sequential Dropping (SSD) ambiguity resolution

This second approach to find a winner in an ambiguous pairwise matrix is organized around the notion of *Schwartz set*. The Schwartz set is the innermost unbeaten set, or the smallest set of candidates such that none are beaten by any candidate outside the set.[1]

In the following code, we are using instructions given in the Debian Constitution, section A.6 (http://www.debian.org/devel/constitution).

First, from the list of ordered pairwise defeats, we generate a set of transitive defeats. A response A transitively defeats a response C if A defeats C or if there is some other response B where A defeats B AND B transitively defeats C.

The used algorithm is organized around two recursive sub-functions looking for a fixpoint:

- the first one, `traverse_defeats`, traverses the original list of `defeats` and for each couple `(winner, loser)` it tries to augment the transitive defeats `trans_defeats` by calling `augment_trans_defeats`. In case `trans_defeats` is augmented, we restart from scratch using the original list of `defeats`, just in case new transitivities would appear.
- the second one, `augment_trans_defeats`, augments the orginal list of transitive defeats with all transitive defeats triggered by a couple `(b, c)`. To this aim, it searches a couple `(a, b)`, adds couple `(a, c)` if it does not already exists into the transitive defeats and then restart from scratch. The search stops when all couple have been examined and no new couple can be added.

Note: This algorithm could probably be optimized, in order to reach the fixpoint more quickly.

224 ⟨*voting.ml* 219a⟩+≡ ◁223 225a▷

```
let transitive_defeats ~defeats =
  let trans_defeats_start = List.map (fun (w, l, _) -> (w, l)) defeats in
  let rec augment_trans_defeats (b, c) td_seen td_to_see =
    match td_to_see with
    | [] -> td_seen
    | (a, x) :: tail_to_see ->
        if x = b && not(List.mem (a, c) td_seen) && not(List.mem (a, c) td_to_see) then
          augment_trans_defeats (b, c) [] ((a, c) :: td_seen @ td_to_see)
        else
          augment_trans_defeats (b, c) ((a, x) :: td_seen) tail_to_see in
  let rec traverse_defeats cur_defeats trans_defeats =
    match cur_defeats with
    | [] -> trans_defeats
    | (winner, loser, _) :: tail_defeats ->
        let new_trans_defeats =
          augment_trans_defeats (winner, loser) [] trans_defeats in
        if List.length new_trans_defeats > List.length trans_defeats then
          traverse_defeats defeats new_trans_defeats
        else
          traverse_defeats tail_defeats new_trans_defeats in
  traverse_defeats defeats trans_defeats_start
```

[1] Quoting http://www.electionmethods.org/CondorcetEx.htm.

The following auxiliary functions `make_other_candidates` gives a list of all candidates except `x`, i.e. $0, ...,$ `x`-1,`x`$+1, ...,$`size`-1.

225a ⟨*voting.ml* 219a⟩+≡ ◁224 225b▷

```
let make_other_candidates x ~size =
  let rec make_aux n l =
    if n >= 0 then
      begin
        if n <> x then make_aux (n - 1) (n :: l)
        else make_aux (n - 1) l
      end
    else l in
  make_aux (size - 1) []
```

The `schwartz_set` function gives the schwartz set for of set of candidates of size `size` and with a list of transitive defeats `trans_defeats`.

We construct the Schwartz set from the set of transitive defeats. A response A is in the Schwartz set if for all responses B, either A transitively defeats B, or B does not transitively defeat A.

More specifically:

- we iterate over all candidates that could be inserted into the schwartz set. For each candidate `a`:
  - we test for all candidates `b` different of `a` that (i) `a` transitively defeats `b` (i.e. the couple `(a,b)` is in the list of transitive defeats) or (ii) `b` does not transitively defeat `a` (i.e. the couple `(b,a)` is not in the list of transitive defeats)

225b ⟨*voting.ml* 219a⟩+≡ ◁225a 226▷

```
let schwartz_set ~size ~trans_defeats =
  assert(size >= 0);
  let add_to_schartz_set set a =
    let add_to_set_p a =
      let other_candidates = make_other_candidates a ~size in
      List.for_all (fun b ->   (List.mem (a,b) trans_defeats)
                            || not(List.mem (b,a) trans_defeats))
        other_candidates in
    if add_to_set_p a then a :: set else set in
  List.fold_left add_to_schartz_set [] (make_all_candidates ~size)
```

With the above function, we can implement the SSD method. Given a pairwise matrix `mat`, the function gives a the set of winners, containing either a unique element or either multiple elements in case of a tie.

The algorithm is as followed:

- the sub-function `find_smallest_schwartz_set` tries to find the smallest schwartz set by successively dropping the weakest defeat in the list of current defeats `cur_defeats`. In the `aux`iliary function, the smallest of the current and new schwartz set is kept.
- when dropping the weakest defeat in `find_smallest_schwartz_set`, in case the scores `s1` and `s2` of the two lowest defeat are equal (i.e. there is a tie), then these defeats are dropped such that all defeats of equal score are dropped in block; and we do not search for a smallest schwartz set until a higher score is found.

226 ⟨*voting.ml* 219a⟩+≡ ◁225b 227a▷

```
let cloneproof_schwartz_sequential_dropping_winner ~mat =
  let defeats = pairwise_matrix_to_ordered_defeats ~mat in
  let size = Array.length mat in
  assert(size >= 0);
  let start_schwartz_set =
    schwartz_set size (transitive_defeats defeats) in
  let rec find_smallest_schwartz_set cur_set cur_defeats =
    let aux cur_set defeats_tail =
      let trans_defeats = transitive_defeats cur_defeats in
      let new_set = schwartz_set size trans_defeats in
      match cur_set, new_set with
      | _, [] -> find_smallest_schwartz_set cur_set defeats_tail
      | _, [winner] -> [winner]
      | c, n ->
          if List.length c > List.length n then
            find_smallest_schwartz_set new_set defeats_tail
          else
            find_smallest_schwartz_set cur_set defeats_tail in
    match cur_defeats with
    | [] -> List.sort compare cur_set
    | _ :: [] ->
        aux cur_set []
    | (_, _, s1) :: (_, _, s2) :: [] when s1 = s2 ->
        find_smallest_schwartz_set cur_set []
    | (_, _, s1) :: (w, l, s2) :: tail when s1 = s2 ->
        find_smallest_schwartz_set cur_set ((w, l, s2) :: tail)
    | _ :: tail ->
        aux cur_set tail in
  find_smallest_schwartz_set start_schwartz_set defeats
```

40.7 From votes to winner determination

The transformation of a set of votes into the list of winning responses is done in three step:

1. we create an empty summation pairwise matrix that will store the result of the votes, using function `empty_pairwise_matrix` (see code chunk 219c);
2. for each vote, we store it in the summation pairwise matrix;
3. once all votes have been transformed and summed in their pairwise matrix counterpart, we call `winners_from_summed_votes` to compute the list of winners.

Function `sum_vote_to_pairwise_matrix` adds a `vote` (given as a list of chosen options) with its `weight` to the current pairwise matrix `sum_matrix` used to sum votes. For a simple participant, the weight is 1. For a delegate, the weight is greater than or equal to 0 (in case he has no delegation).

227a ⟨*voting.ml* 219a⟩+≡ ◁226 227b▷

```
let sum_vote_to_pairwise_matrix sum_matrix vote weight =
  assert(weight >= 0);
  let size = Array.length sum_matrix in
  let m = vote_to_pairwise_matrix ~size:size ~vote:vote in
  multiply_matrix ~dst:m ~by:weight;
  add_to_matrix ~dst:sum_matrix ~src:m
```

Function `winners_from_votes` computes the list of winning options given the `pairwise_matrix` of votes. It first try to find an `unambiguous_winner` and, if it does nto exists, falls back to the Clone Proof Schwartz Sequential Dropping.

227b ⟨*voting.ml* 219a⟩+≡ ◁227a 228▷

```
let winners_from_summed_votes pairwise_matrix =
  try
    [unambiguous_winner pairwise_matrix]
  with
  | Not_found ->
      cloneproof_schwartz_sequential_dropping_winner pairwise_matrix
```

40.8 Automatic tests

228 ⟨*voting.ml* 219a⟩+≡ ◁227b 230▷

```
let _ =
  if Config.do_autotests then begin
  (* we consider a vote with answers 0:A, 1:B, 2:C, 3:D *)
    printf "  pairwise_matrix autotests...";
    let vote1 = [1; 3; 2] in (* B, D, C *)
    let vote2 = [3; 1] in (* D, B *)
    let m1 = vote_to_pairwise_matrix 4 vote1 in
    let m2 = vote_to_pairwise_matrix 4 vote2 in
  (* we want following matrices:
     m1=
     |     0     1     2     3
     ____________________________
     0|     0     0     0     0
     1|     1     0     1     1
     2|     1     0     0     0
     3|     1     0     1     0
     m2=
     |     0     1     2     3
     ____________________________
     0|     0     0     0     0
     1|     1     0     1     0
     2|     0     0     0     0
     3|     1     1     1     0
     sum=
     |     0     1     2     3
     ____________________________
     0|     0     0     0     0
     1|     2     0     2     1
     2|     1     0     0     0
     3|     2     1     2     0
   *)
(*   printf "\nm1=\n"; *)
(*   print_matrix m1; *)
    assert(m1.(0).(0)=0 && m1.(0).(1)=0 && m1.(0).(2)=0 && m1.(0).(3)=0);
    assert(m1.(1).(0)=1 && m1.(1).(1)=0 && m1.(1).(2)=1 && m1.(1).(3)=1);
    assert(m1.(2).(0)=1 && m1.(2).(1)=0 && m1.(2).(2)=0 && m1.(2).(3)=0);
    assert(m1.(3).(0)=1 && m1.(3).(1)=0 && m1.(3).(2)=1 && m1.(3).(3)=0);
(*   printf "m2=\n"; *)
(*   print_matrix m2; *)
    assert(m2.(0).(0)=0 && m2.(0).(1)=0 && m2.(0).(2)=0 && m2.(0).(3)=0);
    assert(m2.(1).(0)=1 && m2.(1).(1)=0 && m2.(1).(2)=1 && m2.(1).(3)=0);
    assert(m2.(2).(0)=0 && m2.(2).(1)=0 && m2.(2).(2)=0 && m2.(2).(3)=0);
    assert(m2.(3).(0)=1 && m2.(3).(1)=1 && m2.(3).(2)=1 && m2.(3).(3)=0);
    add_to_matrix ~dst:m1 ~src:m2;
(*   printf "sum=\n"; *)
(*   print_matrix m1; *)
    assert(m1.(0).(0)=0 && m1.(0).(1)=0 && m1.(0).(2)=0 && m1.(0).(3)=0);
    assert(m1.(1).(0)=2 && m1.(1).(1)=0 && m1.(1).(2)=2 && m1.(1).(3)=1);
    assert(m1.(2).(0)=1 && m1.(2).(1)=0 && m1.(2).(2)=0 && m1.(2).(3)=0);
    assert(m1.(3).(0)=2 && m1.(3).(1)=1 && m1.(3).(2)=2 && m1.(3).(3)=0);
    multiply_matrix ~dst:m1 ~by:2;
(*   printf "sum=\n"; *)
(*   print_matrix m1; *)
    assert(m1.(0).(0)=0 && m1.(0).(1)=0 && m1.(0).(2)=0 && m1.(0).(3)=0);
    assert(m1.(1).(0)=4 && m1.(1).(1)=0 && m1.(1).(2)=4 && m1.(1).(3)=2);
```

```
  assert(m1.(2).(0)=2 && m1.(2).(1)=0 && m1.(2).(2)=0 && m1.(2).(3)=0);
  assert(m1.(3).(0)=4 && m1.(3).(1)=2 && m1.(3).(2)=4 && m1.(3).(3)=0);
  printf "done\n"
end
```

```
let _ =
  if Config.do_autotests then begin
    printf "  condorcet_voting autotests...";
    (* tests on zero size matrix (i.e. no responses to a question) *)
    let em = empty_pairwise_matrix ~size:0 in
    (try
      let _ = condorcet_winner em in
      assert(false)
    with Not_found -> ());
    assert(cloneproof_schwartz_sequential_dropping_winner ~mat:em = []);
  (* unambiguous pairwise matrix *)
    let upm = Array.make_matrix 4 4 0 in
    upm.(0).(1) <- 63; upm.(0).(2) <- 89; upm.(0).(3) <- 57;
    upm.(1).(0) <- 87; upm.(1).(2) <- 78; upm.(1).(3) <- 73;
    upm.(2).(0) <- 69; upm.(2).(1) <- 72; upm.(2).(3) <- 74;
    upm.(3).(0) <- 67; upm.(3).(1) <- 51; upm.(3).(3) <- 52;
  (* ambiguous pairwise matrix *)
    let apm = Array.make_matrix 4 4 0 in
    apm.(0).(1) <- 40; apm.(0).(2) <- 22; apm.(0).(3) <- 13;
    apm.(1).(0) <- 37; apm.(1).(2) <- 50; apm.(1).(3) <- 50;
    apm.(2).(0) <- 30; apm.(2).(1) <- 35; apm.(2).(3) <- 25;
    apm.(3).(0) <- 20; apm.(3).(1) <- 60; apm.(3).(3) <- 20;
    assert(unambiguous_winner upm = 1);
    (try
      let _ = unambiguous_winner apm in
      assert(false)
    with Not_found -> ());
    let sd = pairwise_matrix_to_ordered_defeats ~mat:apm in
    assert(sd = [(3, 0, 20);
                 (2, 3, 25);
                 (2, 0, 30);
                 (0, 1, 40);
                 (1, 2, 50);
                 (3, 1, 60)]);
    assert(condorcet_winner ~mat:apm = 3);
    assert(cloneproof_schwartz_sequential_dropping_winner ~mat:apm = [3]);
  (* test transitive defeats construciton *)
    let cmp_cpl (x1, y1) (x2, y2) =
      if compare x1 x2 <> 0 then
        compare x1 x2
      else compare y1 y2 in
    let defeats = [(0,1,-1); (1,2,-1); (2,0,-1); (1,3,-1)] in
    assert(List.sort cmp_cpl (transitive_defeats defeats) =
           [(0,0); (0,1); (0,2); (0,3);
            (1,0); (1,1); (1,2); (1,3);
            (2,0); (2,1); (2,2); (2,3)]);
  (* ambiguous matrix with a two part tie *)
    let a2tpm = Array.make_matrix 2 2 0 in
    a2tpm.(0).(1) <- 50;
    a2tpm.(1).(0) <- 50;
    (try
      let _ = unambiguous_winner a2tpm in
      assert(false)
    with Not_found -> ());
    assert(cloneproof_schwartz_sequential_dropping_winner ~mat:a2tpm = [0; 1]);
  (* ambiguous matrix with a cyclic three part tie *)
    let a3tpm = Array.make_matrix 3 3 0 in
```

```
  a3tpm.(0).(1) <- 0; a3tpm.(0).(2) <- 50;
  a3tpm.(1).(0) <- 50; a3tpm.(1).(2) <- 0;
  a3tpm.(2).(0) <- 0; a3tpm.(2).(1) <- 50;
  (try
    let _ = unambiguous_winner a3tpm in
    assert(false)
  with Not_found -> ());
  assert(cloneproof_schwartz_sequential_dropping_winner ~mat:a3tpm =
         [0; 1; 2]);
  printf "done\n"
end
```

Chapter 41

Position base

Module `Posbase` is used to store and manipulate the Position base.

232a ⟨*posbase.ml* 232a⟩≡ 232b ▷
```
(* copyright 2003-2004 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Printf
open Participants
```

41.1 Exceptions

Here is the set of exceptions raised by following code.

232b ⟨*posbase.ml* 232a⟩+≡ ◁ 232a 232c ▷
```
exception Question_not_found
exception Question_already_in_base
exception Response_already_made
exception Vote_choice_not_found of int (* number of choice not found *)
exception Duplicated_vote_choice of int (* the duplicated choice *)
```

41.2 Data structures

Here are data structures that define the various part of the position base.

A question has two possible status: `Tagging_only` where only taggers can manipulate and see it, and `Public` where everybody can see and vote on it.

232c ⟨*posbase.ml* 232a⟩+≡ ◁ 232b 232d ▷
```
type question_status =
  | Tagging_only
  | Public
```

A stored question contains description of the question itself `q_desc`, its author `q_author`, its optional `limit_date` and its status. If `limit_date` is set to `None`, there is no limit date, otherwise the limit date is put as the number of seconds since 00:00:00 GMT, Jan. 1, 1970.

232d ⟨*posbase.ml* 232a⟩+≡ ◁ 232c 233a ▷
```
type question = { q_desc: string;
                  q_author: authenticated_participant;
                  limit_date: Int64.t option;
                  mutable status : question_status;
                }
```

A response to a question contains the response itself `r_desc`, its author `r_author` and an optional `external_link` arguing this response.

233a ⟨*posbase.ml* 232a⟩+≡ ◁232d 233b▷

```
type response = { r_desc: string;
                  r_author: authenticated_participant;
                  mutable external_link: string option;
                }
```

The author of a vote is either a `Delegate` or a simple participant whose name has been `Individual`.

233b ⟨*posbase.ml* 232a⟩+≡ ◁233a 233c▷

```
type vote_author =
    Delegate of authenticated_participant
  | Individual of string
```

A `position` in the Position base is made of a `question`, its attached `responses`, the expressed `votes` and the optional `elected_responses` (that is set if the votes have been computed).

233c ⟨*posbase.ml* 232a⟩+≡ ◁233b 233d▷

```
type position = { question: question;
                  mutable responses: response array;
                  votes: (vote_author, int list) Hashtbl.t;
                  mutable elected_responses: int list;
                }
```

The different questions (and their responses and vote) are stored in a hash table `positions` indexed by the question identifiers. Those identifiers are managed using `Id.Question` module in `ids` table.

We also use `Id.Question` module to store timestamps associated to each question. Those timestamps are updated whenever a questions is created or is added a new response.

233d ⟨*posbase.ml* 232a⟩+≡ ◁233c 234a▷

```
type position_base = {
    ids : Id.Question.t;
    positions : (int, position) Hashtbl.t;
     (* question id |-> position *)
  }

let new_position_base () = {
  ids = Id.Question.create ();
  positions = Hashtbl.create 3;
}

let the_position_base = ref (new_position_base ())
```

In all following code, we use following conventions:

- `q_id`: a question identifier;
- `q_desc`: a question descriptor, aka question title;

41.3 Manipulation on question identifiers and ids

Functions `id_of_desc` and `desc_of_id` allow to convert question identifiers to/from question descriptors. They raise `Not_found` in case the searched item does not exists in the Position base.

234a ⟨*posbase.ml* 232a⟩+≡ ◁233d 234b▷

```
let id_of_desc ~q_desc =
  Id.Question.rev_find !the_position_base.ids q_desc

let desc_of_id ~q_id =
  Id.Question.find !the_position_base.ids q_id
```

Function `max_id` returns the biggest question identifier.

234b ⟨*posbase.ml* 232a⟩+≡ ◁234a 234c▷

```
let max_id () = Id.Question.length !the_position_base.ids - 1
```

41.4 Response handling

Function `update_timestamp` update to the latest one the timestamp of question `q_id`.

todo: In current code, the classification is totally decoupled from position base. Thus, one could change classification of a question and update the timestamp of this question in position base even if this question does not exist in position base. Right now, we remain silent in that case, i.e. exception `Not_found` is not raised. TO DO

234c ⟨*posbase.ml* 232a⟩+≡ ◁234b 235a▷

```
let update_timestamp ~q_id =
  try
    Id.Question.update_timestamp !the_position_base.ids q_id
  with Not_found -> ()
```

When we add a response `r_desc` from `author` to a question `q_id`, we first check that this response is not already present. In case the response has already been seen, we raise `Response_already_made`.

The external link (`extlink`) could be provided at a later time.

In case the question `q_id` cannot be found, we raise `Question_not_found`. In case the response is badly formatted, we raise `Norm.Invalid_format`.

235a ⟨*posbase.ml* 232a⟩+≡ ◁234c 235b▷

```
let add_response ~q_id ~r_desc ~author ?(extlink = None) () =
  let r_desc = Norm.normalize_response r_desc in
  Norm.check_response r_desc;
  (match extlink with
  | None -> ()
  | Some link -> Norm.check_link link
  );
  try
    let q = Hashtbl.find !the_position_base.positions q_id in
    (* look for the same response *)
    let len = Array.length q.responses in
    for i = 0 to len - 1 do
      if q.responses.(i).r_desc = r_desc then raise Response_already_made
    done;
    (* we have a new response so add it *)
    let new_resp = {r_desc = r_desc;
                    r_author = author;
                    external_link = extlink;} in
    let copy_resp i = if i < len then q.responses.(i) else new_resp in
    q.responses <- Array.init (len + 1) copy_resp;
    update_timestamp ~q_id;
    (* add to RSS feed *)
    let q_desc = desc_of_id ~q_id in
    let link_as_string l = match l with None -> "" | Some link -> link in
    let resp_list =
      List.map (fun r -> (r.r_desc, link_as_string r.external_link))
        (Array.to_list q.responses) in
    let tag_list = Classification.get_question_tags_as_label ~q_id in
    Rss.add ~q_id ~q_desc ~tag_list ~resp_list
  with Not_found ->
    raise Question_not_found
```

41.5 Question handling

To each no question, we add the following default response that signifies the question is irrelevant.

235b ⟨*posbase.ml* 232a⟩+≡ ◁235a 236a▷

```
let default_rejected_question_response = "Question rejected"
```

To add a new question, we just take two mandatory parameters: the description `q_desc` of a question and the `author`. The limit date (`ldate`) could be provided or modified at a later time. This functions returns the question identifier.

The created question is in `Tagging_only` status.

This function raises `Question_already_in_base` in case the question already exists in the position base. It raises `Norm.Invalid_format` in case the question descriptor is badly formatted.

Assumption: The question is not already in the position base.

236a ⟨*posbase.ml* 232a⟩+≡ ◁235b 236b▷

```
let add_question ~q_desc ~author ?(ldate = None) () =
  let q_desc = Norm.normalize_question q_desc in
  Norm.check_question ~q_desc;
  if Id.Question.rev_mem !the_position_base.ids q_desc then
    raise Question_already_in_base;
  let q_id = Id.Question.add !the_position_base.ids q_desc in
  Hashtbl.add !the_position_base.positions
    q_id { question = { q_desc = q_desc;
                        q_author = author;
                        limit_date = ldate;
                        status = Tagging_only;
                      };
           responses = Array.make 0
             {r_desc = "";
              r_author = Anonymous;
              external_link = None};
           votes = Hashtbl.create 3;
           elected_responses = [];
         };
  add_response ~q_id ~r_desc:default_rejected_question_response ~author ();
  (* add question's specific tag *)
  let question_specific_tag = Printf.sprintf "question %d" q_id in
  let tag_id =
    try
      Classification.create_tag question_specific_tag
    with Id.Tag.Already_exists ->
      Classification.get_tag_id ~tag_label:question_specific_tag in
  Classification.tag_question ~q_id ~tag_id;
  (* add to RSS feed *)
  let tag_list = Classification.get_question_tags_as_label ~q_id in
  Rss.add ~q_id ~q_desc ~tag_list
    ~resp_list:[(default_rejected_question_response, "")];
  q_id
```

Function `get_position` returns all the information for question of `q_id`. In case the question is not found, it raises `Not_found`.

fixme: No check of rights of requester FIXME

236b ⟨*posbase.ml* 232a⟩+≡ ◁236a 237a▷

```
let get_position ~q_id =
  Hashtbl.find !the_position_base.positions q_id
```

41.6 Vote handling

Using `check_vote`, we check that a `vote` given on `question` is valid:

- each chosen response effectively exists as available responses (if not we raise `Vote_choice_not_found`);
- there is no duplicate choice (if not, we raise `Duplicated_vote_choice`).

Note that as we can't have duplicate or unavailable response, we are sure that won't have to many votes.

237a ⟨*posbase.ml* 232a⟩+≡ ◁236b 237b▷

```
let check_vote question vote =
  let len = Array.length question.responses in
  List.iter (fun i -> if i < 0 || i >= len then
    raise (Vote_choice_not_found i)) vote;
  let check_duplicate_vote seen_votes a_vote =
    if List.exists (fun v -> v = a_vote) seen_votes then
      raise (Duplicated_vote_choice a_vote) in
  ignore(List.fold_left
           (fun seen_votes a_vote ->
             check_duplicate_vote seen_votes a_vote;
             a_vote :: seen_votes) [] vote)
```

To set a vote, we should give the identifier of the question on which we vote (`q_id`), the vote `author` and the `vote` itself as a list of integers that describe the ordered list of responses.

In case the question `q_id` cannot be found, we raise `Question_not_found`. It can also raise `Duplicated_vote_choice` or `Vote_choice_not_found` (see `check_vote` above).

237b ⟨*posbase.ml* 232a⟩+≡ ◁237a 237c▷

```
let set_vote ~q_id ~author ~vote =
  try
    let q = Hashtbl.find !the_position_base.positions q_id in
    check_vote q vote;
    Hashtbl.replace q.votes author vote
  with Not_found ->
    raise Question_not_found
```

Function `get_vote` returns vote of participant with `login` for question identifier `q_id`. In case the question is not found, this function raises `Question_not_found`. If `login` has not voted, it returns an empty list of choices.

237c ⟨*posbase.ml* 232a⟩+≡ ◁237b 238a▷

```
let get_vote ~q_id ~login =
  try
    let vote_key =
      if Participants.is_delegate login then
        Delegate (Participants.unsafe_authenticated_of_string login)
      else Individual login in
    let p = Hashtbl.find !the_position_base.positions q_id in
    if Hashtbl.mem p.votes vote_key then
      Hashtbl.find p.votes vote_key
    else
      []
  with Not_found ->
    raise Question_not_found
```

41.7 Computations on votes

Function `weight_of_vote_author` returns the weight of the vote author `a`.

238a ⟨*posbase.ml* 232a⟩+≡ ◁237c 238b▷
```
let weight_of_vote_author a =
  match a with
  | Individual _ -> 1
  | Delegate _ -> 0
```

Function `update_position` computes the new winners (in the `elected_responses` field) for position `p` in the Position base. The algorithm followed is the same as given in §40.7.

238b ⟨*posbase.ml* 232a⟩+≡ ◁238a 238c▷
```
let update_position p =
  let size = Array.length p.responses in
  (* empty pairwise matrix *)
  let pairwise_sum = Voting.empty_pairwise_matrix ~size:size in
  (* vote summation *)
  let add_vote matrix author vote =
    let weight = weight_of_vote_author author in
    Voting.sum_vote_to_pairwise_matrix matrix vote weight in
  Hashtbl.iter (add_vote pairwise_sum) p.votes;
  (* winners computation *)
  let winners = Voting.winners_from_summed_votes pairwise_sum in
  p.elected_responses <- winners
```

Function `update_elected_responses` computes the set of winners for question of identifier `q_id` and updates the `elected_responses` field of the question in the Position base.

238c ⟨*posbase.ml* 232a⟩+≡ ◁238b 238d▷
```
let update_elected_responses ~q_id =
  let p = get_position q_id in
  update_position p
```

Function `update_all_elected_responses` is used to recompute all votes of the Position base.

238d ⟨*posbase.ml* 232a⟩+≡ ◁238c 238e▷
```
let update_all_elected_responses () =
  Hashtbl.iter (fun _ p -> update_position p) !the_position_base.positions
```

Function `remove_votes_from_author` removes all votes from `author` on questions that have the same classification as `domain`. We do *not* recompute the voting result on modified positions.

238e ⟨*posbase.ml* 232a⟩+≡ ◁238d 238f▷
```
let remove_votes_from_author ~author ~domain =
  failwith "internal error: not implemented"
```

41.8 Manipulation of question status

Function `set_status` set new question `status` for question of identifier `q_id`. It raises `Not_found` exception if `q_id` is invalid.

238f ⟨*posbase.ml* 232a⟩+≡ ◁238e 239a▷
```
let set_status ~q_id ~status =
  let q = get_position ~q_id in
  q.question.status <- status;
  update_timestamp ~q_id
```

41.9 XML support

41.9.1 XML export

Auxiliary function `responses_to_cduce_xml` transforms an array of responses into a data type compatible with CDuce module interface.

239a ⟨*posbase.ml* 232a⟩+≡ ◁238f 239b▷

```
let responses_to_cduce_xml responses =
  let cduce_xml response =
    { Xml.r_desc = response.r_desc;
      Xml.r_author = Participants.authenticated_to_string response.r_author;
      Xml.r_links
        = match response.external_link with
        | None -> []
        | Some link -> [ link ]; } in
  Array.map cduce_xml responses
```

Auxiliary function `votes_to_cduce_xml` transforms a set of votes into a data type compatible with CDuce module interface. We decided to store the `voter` as a string, so as to enable anonymous voters in the future.

239b ⟨*posbase.ml* 232a⟩+≡ ◁239a 239c▷

```
let votes_to_cduce_xml votes =
  let to_list voter choices accu =
    let s =
      match voter with
      | Delegate auth -> Participants.authenticated_to_string auth
      | Individual name -> name in
    { Xml.voter = s;
      Xml.choices = choices } :: accu in
  Hashtbl.fold to_list votes []
```

Function `to_cduce_xml` produces a version of the tags in Classification base compatible with needed CDuce data structure for XML export (see code chunk 258a).

239c ⟨*posbase.ml* 232a⟩+≡ ◁239b 240a▷

```
let to_cduce_xml () =
  let to_list id p accu =
    let str_timestamp =
      let timestamp = Id.Question.get_timestamp !the_position_base.ids id in
      Timestamp.to_string timestamp in
    (id, { Xml.q_timestamp = str_timestamp;
           Xml.q_desc = p.question.q_desc;
           Xml.q_author
             = Participants.authenticated_to_string p.question.q_author;
           Xml.limit_date
             = (match p.question.limit_date with
                | None -> ""
                | Some d -> Int64.to_string d);
           Xml.status
             = (match p.question.status with
                | Tagging_only -> Xml.Tagging_only
                | Public -> Xml.Public);
           Xml.responses = responses_to_cduce_xml p.responses;
           Xml.votes = votes_to_cduce_xml p.votes;
           Xml.elected = p.elected_responses; }) :: accu in
  Hashtbl.fold to_list !the_position_base.positions []
```

41.9.2 XML import

Helper function `responses_of_cduce_xml` transforms a `Xml`'s `responses_array` into a response array compatible with in memory Position base.

240a ⟨*posbase.ml* 232a⟩+≡ ◁239c 240b▷

```
let responses_of_cduce_xml responses_array =
  let to_mem response =
    { r_desc = response.Xml.r_desc;
      r_author
        = Participants.unsafe_authenticated_of_string response.Xml.r_author;
      external_link
        = match response.Xml.r_links with
        | [] -> None
        | [ link ] -> Some link
        | _ -> failwith "internal error: should not have several links"; } in
  Array.map to_mem responses_array
```

Helper function `votes_of_cduce_xml` transforms a `Xml`'s `votes_list` into a data structure compatible with in memory Position base.

240b ⟨*posbase.ml* 232a⟩+≡ ◁240a 241a▷

```
let votes_of_cduce_xml votes_list =
  let votes_h = Hashtbl.create 3 in
  let of_list v =
    let voter =
      match v.Xml.voter with
      | name when Participants.delegate_naming name ->
          Delegate (Participants.unsafe_authenticated_of_string name)
      | name -> Individual name in
    Hashtbl.add votes_h voter v.Xml.choices in
  List.iter of_list votes_list;
  votes_h
```

Function `of_cduce_xml` restore the in memory Position base from a `question_list` as defined in `Xml` module.

241a ⟨*posbase.ml* 232a⟩+≡ ◁240b 241b▷

```
let of_cduce_xml question_list =
  let ids = Id.Question.create () in
  let positions = Hashtbl.create 3 in
  let of_list (id, question) =
    let auth_participant
        = Participants.unsafe_authenticated_of_string question.Xml.q_author in
    let limit_date =
      match question.Xml.limit_date with
      | "" -> None
      | date -> Some (Int64.of_string date) in
    let status =
      match question.Xml.status with
      | Xml.Tagging_only -> Tagging_only
      | Xml.Public -> Public in
    let posbase_question = { q_desc = question.Xml.q_desc;
                             q_author = auth_participant;
                             limit_date = limit_date;
                             status = status; } in
    let position = { question = posbase_question;
                     responses = responses_of_cduce_xml question.Xml.responses;
                     votes = votes_of_cduce_xml question.Xml.votes;
                     elected_responses = question.Xml.elected; } in
    let timestamp = Timestamp.of_string question.Xml.q_timestamp in
    Id.Question.force ids id timestamp question.Xml.q_desc;
    Hashtbl.add positions id position in
  List.iter of_list question_list;
  the_position_base := { ids = ids; positions = positions }
```

Helper function `comparable_base` returns the Position base as a data structure than can be easily compared with the equal operator. Useful for tests.

241b ⟨*posbase.ml* 232a⟩+≡ ◁241a 241c▷

```
let comparable_base () =
  let base_to_lists base =
    let l1 =
      Id.Question.fold (fun id q_desc accu -> (id, q_desc) :: accu)
        base.ids [] in
    let l2 =
      Hashtbl.fold (fun id p accu -> (id, p) :: accu) base.positions [] in
    (l1, l2) in
  base_to_lists !the_position_base
```

41.10 Timestamps

Function `timestamp_list` returns the complete set of participant's timestamps in `block` as well as the `number` of them.

241c ⟨*posbase.ml* 232a⟩+≡ ◁241b 241d▷

```
let timestamp_list () = Id.Question.timestamp_list !the_position_base.ids
```

Function `get_timestamp` returns the timestamp of question `q_id`.

241d ⟨*posbase.ml* 232a⟩+≡ ◁241c 242▷

```
let get_timestamp ~q_id = Id.Question.get_timestamp !the_position_base.ids q_id
```

41.11 Automatic tests

242 ⟨*posbase.ml* 232a⟩+≡ ◁241d

```
let _ =
  if Config.do_autotests then begin
    printf "  posbase autotests...";
    (* change flag_bases_name during tests to avoid cluterring current
       directory with a dummy RSS feed *)
    let previous_flag_bases_name = !Srvflags.flag_bases_name in
    let tmp_base_filename = Filename.temp_file "toto" ".dmxp" in
    Srvflags.flag_bases_name := tmp_base_filename;

    assert(max_id () = -1);
    let q_desc = "A good question" in
    let voter1 = Individual("voter 1") in
    let voter2 = Individual("voter 2") in
    let q_id = add_question ~q_desc ~author:Anonymous () in
    assert(max_id () = 0);
    add_response ~q_id ~r_desc:"A" ~author:Anonymous ();
    add_response ~q_id ~r_desc:"B" ~author:Anonymous ();
    add_response ~q_id ~r_desc:"C" ~author:Anonymous ();
    add_response ~q_id ~r_desc:"D" ~author:Anonymous ();
    set_vote ~q_id ~author:voter1 ~vote:[1; 3; 2]; (* B, D, C *)
    set_vote ~q_id ~author:voter2 ~vote:[3; 1]; (* D, B *)

    assert(desc_of_id ~q_id = q_desc);
    assert(id_of_desc ~q_desc = q_id);
    (try
      ignore(id_of_desc ~q_desc:"toto");
      assert(false)
    with Not_found -> assert(true));

  (* test bad behavior *)
    (try
      ignore(add_question ~q_desc ~author:Anonymous ());
      assert(false);
    with Question_already_in_base -> ());

  (* following tests are not doable. question ids are checked at creation. *)

  (*     (try  *)
  (*       add_response ~q_id:"toto" ~r_desc:"A" ~author:Anonymous (); *)
  (*       assert(false); *)
  (*     with Question_not_found -> ()); *)
  (*     (try  *)
  (*       set_vote ~q_desc:"toto" ~author:voter1 ~vote:[1; 3; 2]; (\* B, D, C *\) *)
  (*       assert(false); *)
  (*     with Question_not_found -> ()); *)

    (try
      add_response ~q_id ~r_desc:"A" ~author:Anonymous ();
      assert(false);
    with Response_already_made -> ());
    (try
      set_vote ~q_id ~author:voter1 ~vote:[1; 3; 2; 15]; (* B, D, C, bad *)
      assert(false);
    with Vote_choice_not_found i -> assert(i = 15));
    (try
```

```
      set_vote ~q_id ~author:voter1 ~vote:[1; 3; 2; 0; 1]; (* B, C, D, A, B *)
      assert(false);
    with Duplicated_vote_choice 1 -> ());
    update_elected_responses ~q_id;
    let p = get_position ~q_id in
    assert(p.elected_responses = [1; 3]);

    (* question status *)
    assert(p.question.status = Tagging_only);
    set_status ~q_id ~status:Public;
    assert(p.question.status = Public);
    (try
      set_status ~q_id:42 ~status:Public;
      assert(false)
    with Not_found -> assert(true));

    (* check export/import *)
    let old_base = comparable_base () in
    let saved_base = to_cduce_xml () in
    of_cduce_xml saved_base;
    assert(old_base = comparable_base ());

    Classification.initialize ();

(* fixme: tests to restore. *)

(*     Classification.add_question_under_path ~path:[] ~q_desc; *)
(*     let c = Classification.classification_of_question ~q_desc in *)
(*     remove_votes_from_author ~author:voter1 ~domain:c; *)
(*     update_elected_responses ~q_desc; *)
(*     let p2 = get_position ~q_desc in *)
(*     assert(p2.elected_responses = Some [3]); (\* winner from voter2 choice *\) *)
(*     remove_votes_from_author ~author:voter2 ~domain:c; *)
(*     update_elected_responses ~q_desc; *)
(*     let p3 = get_position ~q_desc in *)
(*     assert(p3.elected_responses = Some [0; 1; 2; 3; 4]); (\* no vote, so all  *)
(*                                                            candidates win *\) *)

    (* erase created files *)
    Sys.remove tmp_base_filename;
    Sys.remove ((Filename.chop_extension tmp_base_filename) ^ ".rss");
    (* reset and cleanup before doing real work *)
    Classification.initialize ();
    the_position_base := new_position_base (); (* reset the_position_base *)
    Srvflags.flag_bases_name := previous_flag_bases_name;
    Rss.clear_feed ();
    printf "done\n"
  end
```

Chapter 42

DTD for demexp XML format

This chapter defines the DTD (Document Type Definition) for the XML format used to store and load demexp bases.

As a general rule, the structure is quite strict. The order of elements is given. As only machines should read and write this kind of file, this does not impact usability of this format.

As a general rule, all identifiers are natural integers.

42.1 Top level structure

A demexp XML file is made of four parts, corresponding to the four bases stored in memory. The `demexp_base` tag has a mandatory attribute, `version`, that gives the version number of the XML demexp format used. Only version `0.3` is defined for now.

When writing an XML file using this DTD, one should use the use tag `<?xml version="1.0" encoding="utf-8" ?>` indicating that we follow XML conventions and are using UTF-8 enconding.

244 ⟨*demexp.dtd* 244⟩≡ 245b ▷

```
<!ELEMENT demexp_base (participant_base,
                       delegation_base,
                       question_base,
                       classification_base)>

  <!ATTLIST demexp_base version CDATA #REQUIRED>
```

Here is an example of the top structure of a file in demexp XML format.

245a ⟨*example-demexp-base.xml* 245a⟩≡

```
<?xml version="1.0" encoding="utf-8" ?>

<demexp_base version="0.3">

  <participant_base>
    ⟨example of participant base 246a⟩
  </participant_base>

  <delegation_base>
    ⟨example of delegation base 246c⟩
  </delegation_base>

  <question_base>
    ⟨example of question base 248⟩
  </question_base>

  <classification_base>
    ⟨example of classification base 249b⟩
  </classification_base>

</demexp_base>
```

42.2 Participant base

A `participant_base` is made of zero or more `participant`s.

A `participant` is defined by its `login`, a `password` (in fact, the MD5 hash of the password), and zero or more `group`s to which this participant belong.

The `participant` tag has a mandatory `kind` attribute that indicates if the participant identified by this `login` is an `individual` or a `delegate`. It has also a mandatory `p_id` attribute that stores the unique identifier of the participant, as a natural integer.

245b ⟨*demexp.dtd* 244⟩+≡ ◁244 246b▷

```
<!ELEMENT participant_base (participant*)>

  <!ELEMENT participant (login, password, group*)>

    <!ATTLIST participant kind (individual | delegate) #REQUIRED>

    <!ATTLIST participant p_id CDATA #REQUIRED>
    <!-- p_id is a natural integer -->

    <!ELEMENT login (#PCDATA)>

    <!ELEMENT password (#PCDATA)>

    <!ELEMENT group (#PCDATA)>
```

Here is an example of participant base.

246a ⟨*example of participant base* 246a⟩≡ (245a)

```
<participant p_id="0" kind="individual"> <!-- or "delegate" -->
  <login>root</login>
  <password>6a745aea5b3c6aa059ccfdae74de1633</password>
  <group>admin</group>
  <group>classifier</group>
</participant>

<participant p_id="1" kind="individual"> <!-- or "delegate" -->
  <login>david</login>
  <password>6a745aea5b3c6aa059ccfdae74de1633</password>
  <group>admin</group>
  <group>classifier</group>
</participant>

<participant p_id="2" kind="individual"> <!-- or "delegate" -->
  <login>fred</login>
  <password>6a745aea5b3c6aa059ccfdae74de1633</password>
  <group>admin</group>
  <group>classifier</group>
</participant>
```

42.3 Delegation base

todo: To be latter defined. TO DO

246b ⟨*demexp.dtd* 244⟩+≡ ◁245b 247a▷

```
<!ELEMENT delegation_base (delegation*)>

  <!ELEMENT delegation (#PCDATA)>
```

Here is an example of delegation.

246c ⟨*example of delegation base* 246c⟩≡ (245a)

```
<delegation></delegation>
```

42.4 Question base

A `question_base` is made of zero or more `question`s.

Each `question` contains a `desc`ription, its `author`, a `limit_date`, zero or more `response`s, zero or more `vote`s and zero or more `elected` responses. A `question` has a mandatory `q_id` attribute that stores the question identifier. It has also a mandatory `status` attribute that indicates whether a question is "`tagging_only`" or "`public`".

The `limit_date` tag is either an empty tag (`<limit_date/>`) or a date in ISO 8601 format (see http://www.w3.org/TR/NOTE-datetime for more information on this format).

247a ⟨*demexp.dtd* 244⟩+≡ ◁246b 247b▷

```
<!ELEMENT question_base (question*)>

  <!ELEMENT question (desc, author, limit_date,
                      response*, vote*, elected*)>

    <!ATTLIST question q_id CDATA #REQUIRED>
    <!-- q_id is a natural integer -->

    <!ATTLIST question status (tagging_only | public) #REQUIRED>

    <!ELEMENT desc (#PCDATA)>

    <!ELEMENT author (#PCDATA)>

    <!ELEMENT limit_date (#PCDATA)>
    <!-- limit_date is an empty tag or a date in ISO 8601 format,
         e.g. "2004-03-01" -->
```

A `response` is made of a `desc`ription and an `author` (see code chunk 247a for their definition), as well as an optional `link`. The prefered form for the link is an URI/URL like http://www.demexp.org but not precise naming scheme is yet defined. The ordering of responses define the response identifier, starting at 0.

247b ⟨*demexp.dtd* 244⟩+≡ ◁247a 247c▷

```
<!ELEMENT response (desc, author, link?)>

  <!ELEMENT link (#PCDATA)>
  <!-- prefered form is URI/URL, like http://www.demexp.org -->
```

A `vote` is made of zero or more empty `choice` tag, each one containing a mandatory `is` attribute containing the response number chosen.

A `vote` contains a mandatory `voter` attributes that uniquely identifies the voter.

247c ⟨*demexp.dtd* 244⟩+≡ ◁247b 247d▷

```
<!ELEMENT vote (choice+)>

  <!ATTLIST vote voter CDATA #REQUIRED>

  <!ELEMENT choice EMPTY>

    <!ATTLIST choice is CDATA #REQUIRED>
    <!-- is is a reference to a response id -->
```

Finally, the `elected` tag contains one of the computed winning response, stored as a natural integer. Several of this tag are present in case of multiple winners.

247d ⟨*demexp.dtd* 244⟩+≡ ◁247c 249a▷

```
<!ELEMENT elected (#PCDATA)>
<!-- elected is a natural integer -->
```

Here is an example of question base.

248 ⟨*example of question base* 248⟩≡ (245a)

```
<question q_id="0">
  <description>Should we use demexp to make choices on demexp?</description>
  <author>david</author>
  <limit_date/>

  <response>
    <description>Question rejected</description>
    <author>fred</author>
  </response>

  <response>
    <description>Yes</description>
    <author>fred</author>
    <link>http://savannah.nongnu.org/projects/demexp</link>
  </response>

  <response>
    <description>No</description>
    <author>david</author>
  </response>

  <vote voter="fred">
    <choice is="1"/>
    <choice is="2"/>
  </vote>

  <vote voter="david">
    <choice is="2"/>
    <choice is="1"/>
  </vote>

  <elected>1</elected>
  <elected>2</elected> <!-- several winners -->

</question>
```

42.5 Classification base

A `classification_base` is made of two parts:

`tags`: the set of tags with their identifiers. Each tag is described in a `tag` marker;

`tagged_questions`: the set of tags assigned to each question, described in `tagged` markers.

A `tag` is made of a mandatory `t_id` attribute which is the unique identifier of the tag, stored as a natural integer, and contains the label of the tag.

A `tagged` question is made of a mandatory `q_id` attribute which identifies the question on which the set of tags is put. Each attached tag is described in a `w` marker (w for with).

249a ⟨*demexp.dtd* 244⟩+≡ ◁247d

```
<!ELEMENT classification_base (tags?, tagged_question?)>

  <!ELEMENT tags (tag*)>

    <!ELEMENT tag (#PCDATA)>

      <!ATTLIST tag t_id CDATA #REQUIRED>
      <!-- t_id is a natural integer -->

  <!ELEMENT tagged_questions (tagged*)>

    <!ELEMENT tagged (w*)>

      <!ATTLIST tag q_id CDATA #REQUIRED>
      <!-- q_id is a natural integer -->

      <!ELEMENT w (#PCDATA)>
      <!-- w is a natural integer -->
```

Here is an example of classification base.

249b ⟨*example of classification base* 249b⟩≡ (245a)

```
<tags>
  <tag t_id="1">demexp</tag>
</tags>

<tagged_questions>
   <tagged q_id="0">
     <w>1</w>
   </tagged>
 </tagged_questions>
```

Chapter 43

XML export and import

To read and write XML file format of demexp, we are using CDuce[1]. CDuce is a language specially tailored to manipulate XML. It has a natural binding with OCaml.

Note: I [david] still have mixed feelings about the use of CDuce. On one side, it provides a complete parsing of XML files, with all the complicated parts like character encodings. Moreover, as a typed language, it provides nice properties on parsed values. On the other side, this is a very big beast with a lot of dependencies. It complicates the server a lot! Time will tell if integrating CDuce is a good decision.

43.1 Definition of data types

Firstly, we define the data type corresponding to our XML format. This is basically a typed grammar where we define the ordering and number of occurences of each element. This grammar should correspond to the DTD defined in chapter 42 with types added.

todo: We should mechanically check that the defined data type indeed corresponds to the DTD. TO DO

250 ⟨*xml.cd* 250⟩≡ 252a ▷

```
(* copyright 2004 Serge LEBLANC *)
(* copyright 2005 David MENTRE *)
(* this software is under GNU GPL *)

type Xml_int = [ '0'--'9'+ ] ;;
type Xml_version = [ '0'--'9'+ ('.' '0'--'9'+)* ] ;;
type Xml_id = Xml_int ;;
type Xml_timestamp = Latin1 ;; (* in fact int32 string representation *)
type Xml_kind = "individual" | "delegate" ;;
type Xml_login = <login>Latin1 ;;
type Xml_password = <password>Latin1 ;;
type Xml_group = <group>Latin1 ;;
type Xml_participant =
    <participant p_id=Xml_id
      kind=Xml_kind p_ts=?Xml_timestamp>[ Xml_login
                                          Xml_password Xml_group* ] ;;
(* todo: make p_ts mandatory in future release *)
type Xml_participant_base = <participant_base>[ Xml_participant* ] ;;
type Xml_tag = <tag t_id=Xml_id t_ts=?Xml_timestamp>Latin1 ;;
type Xml_tags = <tags>[ Xml_tag* ] ;;
type Xml_w = <w>Xml_id ;;
type Xml_tagged = <tagged q_id=Xml_id>[ Xml_w* ] ;;
```

[1] http://cduce.org/

```
type Xml_tagged_questions = <tagged_questions>[ Xml_tagged* ] ;;
type Xml_classification_base =
    <classification_base>[ Xml_tags Xml_tagged_questions ] ;;
type Xml_description = <desc>Latin1 ;;
type Xml_author = <author>Latin1 ;;
type Xml_limit_date = <limit_date>Latin1 ;;
type Xml_link = <link>Latin1 ;;
type Xml_vote = <vote voter=Latin1>[ <choice is=Xml_int>[]* ] ;;
type Xml_elected = <elected>Xml_int ;;
type Xml_response =
    <response>[ Xml_description Xml_author Xml_link* ] ;;
type Xml_status = "tagging_only" | "public" ;;
type Xml_question =
    <question q_id=Xml_id status=Xml_status
      q_ts=?Xml_timestamp>[ Xml_description
                            Xml_author
                            Xml_limit_date
                            Xml_response*
                            Xml_vote*
                            Xml_elected* ] ;;
(* todo: make q_ts mandatory in future release *)
type Xml_question_base = <question_base>[ Xml_question* ] ;;

type Xml_demexp_base =
    <demexp_base version=Xml_version>[ Xml_participant_base
                                       Xml_question_base
                                       Xml_classification_base ] ;;
```

We then define the CDuce data type that corresponds to the OCaml interface of our module (see code chunk 258a for corresponding OCaml code). Note that `Ocaml_int` integers are of limited scale, while XML ones (`Xml_int`) are arbitrarily large.

252a ⟨*xml.cd* 250⟩+≡ ◁250 252b▷

```
type Ocaml_string = Latin1 ;;
type Ocaml_timestamp = Ocaml_string ;;
type Ocaml_kind = 'Individual | 'Delegate ;;
type Ocaml_participant = { p_timestamp = Ocaml_timestamp;
                           kind = Ocaml_kind;
                           login = Ocaml_string;
                           password = Ocaml_string;
                           groups = [ Ocaml_string* ] } ;;
type Ocaml_response = { r_desc = Ocaml_string;
                        r_author = Ocaml_string;
                        r_links = [ Ocaml_string* ] } ;;
type Ocaml_vote = { voter = Ocaml_string;
                    choices = [ Caml_int* ] } ;;
type Ocaml_elected = Caml_int ;;
type Ocaml_tag = Ocaml_string ;;
type Ocaml_tagid_list = [ Caml_int* ] ;;
type Ocaml_status = 'Tagging_only | 'Public ;;
type Ocaml_question = { q_timestamp = Ocaml_timestamp;
                        q_desc = Ocaml_string;
                        q_author = Ocaml_string;
                        limit_date = Ocaml_string;
                        status = Ocaml_status;
                        responses = [ Ocaml_response* ];
                        votes = [ Ocaml_vote* ];
                        elected = [ Ocaml_elected* ] } ;;
type Ocaml_xml_content = { version = Ocaml_string;
                           participants = [ (Caml_int, Ocaml_participant)* ];
                           questions = [ (Caml_int, Ocaml_question)* ];
                           tags = [ (Caml_int, Ocaml_timestamp, Ocaml_tag)* ];
                           tagged_questions = [ (Caml_int,
                                                 Ocaml_tagid_list)* ] } ;;
```

43.2 XML export

Helper function `ocaml2xml_int` transform an OCaml integer `i` into its XML counterpart. This function basically maps the string representation of `i` onto the XML type `Xml_int`. It raises an exception if `i` is not a valid XML integer.

252b ⟨*xml.cd* 250⟩+≡ ◁252a 253a▷

```
let ocaml2xml_int (i : Caml_int) : Xml_int =
  let s = string_of i in
    match s with
        x&Xml_int -> x
      | _ -> raise ("unvalid id : " @ s) ;;
```

Function `ocaml2xml_participants` transforms a list of participants `a` into its XML counterpart. This function iterates over all participants, extract needed fields and rewrite them in XML.

253a ⟨*xml.cd* 250⟩+≡ ◁252b 253b▷

```
let ocaml2xml_participants
    (a : [(Caml_int,Ocaml_participant)*]) : [Xml_participant*] =
  let trans_groups ([ Ocaml_string* ] -> [ Xml_group* ])
      g&[Ocaml_string+] -> (map g with s&Latin1 -> <group>s)
    | [] -> []
  in
    let trans_kind (Ocaml_kind -> Xml_kind)
      | 'Individual -> "individual"
      | 'Delegate -> "delegate"
    in
      map a with
        (i, { p_timestamp = ts; kind=q; login=l; password=p; groups=g })
                -> <participant kind=(trans_kind q)
                                p_id=(ocaml2xml_int i)
                                p_ts=ts>[ <login>l
                                          <password>p
                                          !(trans_groups g) ] ;;
```

Function `ocaml2xml_tags` transforms a list of tags `a` into XML. This is just a map over the list.

253b ⟨*xml.cd* 250⟩+≡ ◁253a 253c▷

```
let ocaml2xml_tags
     (a : [(Caml_int,Ocaml_string,Ocaml_string)*]) : [Xml_tag*] =
   map a with (i, ts, l) -> <tag t_id=(ocaml2xml_int i) t_ts=ts>l ;;
```

Function `ocaml2xml_tagged` transforms a list of tagged questions `a` into XML. This is made with a map over questions, with a map over tag identifiers within each question.

253c ⟨*xml.cd* 250⟩+≡ ◁253b 254a▷

```
let ocaml2xml_tagged (a : [(Caml_int,Ocaml_tagid_list)*]) : [Xml_tagged*] =
   map a with (i, tagid_list) ->
     <tagged q_id=(ocaml2xml_int i)>(map tagid_list with t&Int ->
                                                  <w>(ocaml2xml_int t)) ;;
```

Function `ocaml2xml_questions` transforms a list of questions `a` in XML. This is just a map over the list of questions, with the help of functions `trans_responses`, `trans_votes` and `trans_elected` to transform subparts of a question.

254a ⟨*xml.cd* 250⟩+≡ ◁253c 254b▷

```
let ocaml2xml_questions
    (a : [(Caml_int,Ocaml_question)*]) : [Xml_question*] =
  let trans_responses (r : [Ocaml_response*]) : [Xml_response*] =
    let trans_links (l : [Ocaml_string*]) : [Xml_link*] =
      map l with s&Latin1 -> <link>s
    in
      map r with
          { r_desc=d; r_author=a; r_links=l } ->
            <response>[ <desc>d
                        <author>a
                        !(trans_links l) ] in
  let trans_votes (v : [Ocaml_vote*]) : [Xml_vote*] =
    map v with { voter=id; choices=c } ->
        <vote voter=id>(map c with i&Int -> <choice is=(ocaml2xml_int i)>[]) in
  let trans_elected (e : [Ocaml_elected*]) : [Xml_elected*] =
    map e with i&Int -> <elected>(ocaml2xml_int i) in
  let trans_status (Ocaml_status -> Xml_status)
      | ‘Tagging_only -> "tagging_only"
      | ‘Public -> "public" in
  map a with (i,{ q_timestamp=ts; q_desc=d; q_author=a; limit_date=l;
                  status=s; responses=r; votes=v; elected=e }) ->
                     <question q_id=(ocaml2xml_int i)
                               status=(trans_status s)
                               q_ts=ts>
                          [ <desc>d
                            <author>a
                            <limit_date>l
                            !(trans_responses r)
                            !(trans_votes v)
                            !(trans_elected e) ] ;;
```

Finally, the `save` function saves in file named `f` the data base content `c`. The text file is encoded in UTF-8 character set.

254b ⟨*xml.cd* 250⟩+≡ ◁254a 255a▷

```
let save (f : Latin1) (c : Ocaml_xml_content) : [] =
  let content = match c with
   { version=v; participants=p; questions=q; tags=t; tagged_questions=tq } ->
     <demexp_base version=v>[ <participant_base>(ocaml2xml_participants p)
                              <question_base>(ocaml2xml_questions q)
                              <classification_base>[ <tags>(ocaml2xml_tags t)
                                                     <tagged_questions>
                                                          (ocaml2xml_tagged tq)
                                                   ]
                            ] in
  dump_to_file_utf8 f ("<?xml version=\"1.0\" encoding=\"utf-8\" ?>"
                       @ (print_xml_utf8 content)) ;;
```

43.3 XML import

Function `xml2ocaml_participants` transforms a set of `Xml_participant` into an OCaml data structure. We basically do a pattern matching on the components of a `<participant>`.

255a ⟨*xml.cd* 250⟩+≡ ◁254b 255b▷

```
let xml2ocaml_participants
    (p : [ Xml_participant* ]) : [ (Caml_int,Ocaml_participant)* ] =
  let trans_kind (Xml_kind -> Ocaml_kind)
  | "individual" -> 'Individual
  | "delegate" -> 'Delegate in
  map p with
  | <participant p_id=i kind=k p_ts=ts>[ <login>l <password>p; g ] ->
    (match (int_of i) with
     | x&Caml_int -> (x, { p_timestamp = ts;
                           kind=(trans_kind k);
                           login=l;
                           password=p;
                           groups=(map g with <group>s -> s) })
     | _ -> raise "Invalid p_id value")
  |<participant p_id=i kind=k>[ <login>l <password>p; g ] -> (* no timestamp *)
    (match (int_of i) with
     | x&Caml_int -> (x, { p_timestamp = "0";
                           kind=(trans_kind k);
                           login=l;
                           password=p;
                           groups=(map g with <group>s -> s) })
     | _ -> raise "Invalid p_id value") ;;
```

Function `xml2ocaml_tags` transforms a set of `Xml_tag` into an OCaml list of tags.

255b ⟨*xml.cd* 250⟩+≡ ◁255a 256▷

```
let xml2ocaml_tags
    (t : [ Xml_tag* ]) : [ (Caml_int,Ocaml_string, Ocaml_string)* ] =
  map t with
  | <tag t_id=i>l ->
      (match (int_of i) with
       | x&Caml_int -> (x, "0", l)
       | _ -> raise "Invalid t_id value")
  | <tag t_id=i t_ts=ts>l ->
      (match (int_of i) with
       | x&Caml_int -> (x, ts, l)
       | _ -> raise "Invalid t_id value");;
```

Function `xml2ocaml_questions` transforms an XML set of `Xml_question` into the corresponding array of OCaml data structure.

256 ⟨*xml.cd* 250⟩+≡ ◁255b 257a▷

```
let xml2ocaml_questions
    (q : [ Xml_question* ]) : [ (Caml_int, Ocaml_question)* ] =
  let trans_response (r : [ Xml_response* ]) : [ Ocaml_response* ] =
    map r with
      <response>[ <desc>d <author>a l::Xml_link* ] ->
        { r_desc=d;
          r_author=a;
          r_links=(map l with <link>s -> s) } in
  let trans_vote (v : [ Xml_vote* ]) : [ Ocaml_vote* ] =
    map v with <vote voter=id>s ->
      { voter=id;
        choices=(map s with <choice is=i>[] ->
          match (int_of i) with
          | x&Caml_int -> x
          | _ -> raise "Invalid choice value" ) } in
  let trans_elected (e : [ Xml_elected* ]) : [ Ocaml_elected* ] =
    map e with <elected>i ->
      match (int_of i) with
      | x&Caml_int -> x
      | _ -> raise "Invalid elected value" in
  let trans_status (Xml_status -> Ocaml_status)
      | "public" -> 'Public
      | "tagging_only" -> 'Tagging_only in
  map q with
  | <question q_id=i status=s>[ <desc>d
                               <author>a
                               <limit_date>l
                               r::Xml_response*
                               v::Xml_vote*
                               e::Xml_elected* ] ->
     (match (int_of i) with
      | x&Caml_int -> (x, { q_timestamp = "0";
                            q_desc=d;
                            q_author=a;
                            limit_date=l;
                            status=(trans_status s);
                            responses=(trans_response r);
                            votes=(trans_vote v);
                            elected=(trans_elected e) })
      | _ -> raise "Invalid q_id value")
  | <question q_id=i status=s q_ts=ts>[ <desc>d
                                        <author>a
                                        <limit_date>l
                                        r::Xml_response*
                                        v::Xml_vote*
                                        e::Xml_elected* ] ->
      (match (int_of i) with
       | x&Caml_int -> (x, { q_timestamp = ts;
                             q_desc=d;
                             q_author=a;
                             limit_date=l;
                             status=(trans_status s);
                             responses=(trans_response r);
                             votes=(trans_vote v);
                             elected=(trans_elected e) })
```

```
      | _ -> raise "Invalid q_id value") ;;
```

Function `xml2ocaml_tagged_questions` transforms a set of XML `<tagged>` into its OCaml counterpart. We basically iterate over `<tagged>` markers, then, for each one of them, we iterate over the `<w>` markers, returning matching values as OCaml integer.

257a ⟨*xml.cd* 250⟩+≡ ◁256 257b▷

```
let xml2ocaml_tagged_questions
    (t : [ Xml_tagged* ]) : [ (Caml_int,Ocaml_tagid_list)* ] =
  let trans_w (wlist : [ Xml_w* ]) : Ocaml_tagid_list =
    map wlist with <w>i ->
      match (int_of i) with
      | n&Caml_int -> n
      | _ -> raise "Invalid w value" in
  map t with <tagged q_id=i>wlist ->
    match (int_of i) with
    | x&Caml_int ->
        (x, (trans_w wlist))
    | _ -> raise "Invalid q_id value" ;;
```

Function `load` loads the XML of file named `f` and gives it back as an OCaml compatible data structure, as defined in code chunk 258a.

257b ⟨*xml.cd* 250⟩+≡ ◁257a

```
let load (f : Latin1) : Ocaml_xml_content =
  let d : Xml_demexp_base = match load_xml f with
    | x&Xml_demexp_base -> x
    | _ -> raise "Not a Demexp document" in
  { version = (match d with <demexp_base version=v>_ -> v);
    participants
      = (xml2ocaml_participants ([d]/Xml_participant_base/Xml_participant));
    questions = (xml2ocaml_questions ([d]/Xml_question_base/Xml_question));
    tags = (xml2ocaml_tags ([d]/Xml_classification_base/Xml_tags/Xml_tag));
    tagged_questions
      = (xml2ocaml_tagged_questions
           ([d]/Xml_classification_base/Xml_tagged_questions/Xml_tagged))
  } ;;
```

43.4 OCaml interface to CDuce code

This OCaml interface defines the type of the above CDuce module as seen on the OCaml side. We basically rewrite all the information given in the Cduce data type (cf. code chunk 252a).

Note: As this file is just an interface, it can be used by previously defined modules (no issue of module dependency).

We use a `string` to read and write timestamps from XML files. Error handling is thus not done in CDuce but in our OCaml code when convertion from `string` to `Timestamp.t`.

258a ⟨*xml.mli* 258a⟩≡ 258b ▷

```
type timestamp = string

type participant_kind = Individual | Delegate

type participant = {
    p_timestamp : timestamp;
    kind : participant_kind;
    login : string;
    password : string;
    groups : string list;
  }

type response = {
    r_desc : string;
    r_author : string;
    r_links : string list; (* optional external link *)
  }

type vote = {
    voter : string;
    choices : int list;
  }
```

For `responses`, we use an `array` instead of a `list` because it simplifies coding.

258b ⟨*xml.mli* 258a⟩+≡ ◁ 258a 259 ▷

```
type question_status = Tagging_only | Public

type question = {
    q_timestamp : timestamp;
    q_desc : string;
    q_author : string;
    limit_date : string;
    status : question_status;
    responses : response array;
    votes : vote list;
    elected : int list;
  }

type tag = string

type tagid_list = int list
```

In following `(int * _something_) list` patterns, the `int` stores the identifier of the corresponding element (participant, tag or question).

259 ⟨*xml.mli* 258a⟩+≡ ◁258b

```
type xml_content = {
    version : string;
    participants : (int * participant) list;
    tags : (int * timestamp * tag) list;
    tagged_questions : (int * tagid_list) list;
    questions : (int * question) list;
}

val save: string -> xml_content -> unit
 (* save filename data *)

val load: string -> xml_content
 (* load filename *)
```

Chapter 44

Input/Ouput

The module `Io` is used to save and load the databases in binary format. The `Marshal` module is used for this. This implies that saving of databases is not type safe, it works accross platforms supported by OCaml but only for a given release of the OCaml compiler. In other words, the binary saving of databases is used only for a local use and not for long term storage of databases.

All bases are a reference on the base itself to allow easy saving and loading.

By default, the bases are saved and loaded in current directory.

260a ⟨*io.ml* 260a⟩≡ 260b ▷

```
(* copyright 2003-2004 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Printf
open Srvflags
open Sys
```

The suffix for bases in XML format is `.dmxp` (DeMeXP).

44.1 File rotation

To avoid any issues if a database can't be saved (e.g. server crashing while bases are saved), we rotate amongs several copies of the same file. Function `rotate_file` takes a file name of `name` and it renames files such that file named *name* is renamed *name*`.0`, file named *name*`.0` is renamed *name*`.1`, ..., until *name*`.max_rotation` is reached.

260b ⟨*io.ml* 260a⟩+≡ ◁ 260a 260c ▷

```
let rotate_file name max_rotation =
  for i = max_rotation downto 1 do
    let previous = name ^ "." ^ (string_of_int (i - 1)) in
    if file_exists previous then
      rename previous (name ^ "." ^ (string_of_int i))
  done;
  if file_exists name then
    rename name (name ^ ".0")
```

44.2 Data bases saving in XML format

We define the current version of the XML format.

260c ⟨*io.ml* 260a⟩+≡ ◁ 260b 261a ▷

```
let xml_version = "0.6"
```

Auxiliary function `build_cduce_data` calls the functions defined in respective database modules to build the final CDuce compatible data structure.

261a ⟨*io.ml* 260a⟩+≡ ◁260c 261b▷

```
let build_cduce_data () =
  { Xml.version = xml_version;
    Xml.participants = Participants.to_cduce_xml ();
    Xml.questions = Posbase.to_cduce_xml ();
    Xml.tags = Classification.tags_to_cduce_xml ();
    Xml.tagged_questions = Classification.tagged_questions_to_cduce_xml (); }
```

Function `save_bases_xml` save the bases stored in binary in a text file in XML format. The file name is automatically generated using a default name or a command line option (in `flag_bases_name` global variable).

261b ⟨*io.ml* 260a⟩+≡ ◁261a 261c▷

```
let save_bases_xml () =
  flush_all (); (* to have log messsage in correct order *)
  let filename = !flag_bases_name in

  rotate_file filename Config.maximum_file_rotation;

  dbg "saving to file \"%s\"." filename;
  let xml_content = build_cduce_data () in
  Xml.save filename xml_content;
  dbg " done."
```

44.3 Data bases loading from XML format

261c ⟨*io.ml* 260a⟩+≡ ◁261b

```
let load_bases_xml () =
  flush_all (); (* to have log messsage in correct order *)
  let filename = !flag_bases_name in
  if file_exists filename then (
    log "loading from file \"%s\"." filename;
    let xml_content = Xml.load filename in

    if xml_content.Xml.version <> xml_version then
      log "warning: the loaded version (%s) is different from current one (%s). Errors might ha

    Participants.of_cduce_xml xml_content.Xml.participants;
    Classification.of_cduce_xml xml_content.Xml.tags
      xml_content.Xml.tagged_questions;
    Posbase.of_cduce_xml xml_content.Xml.questions;

    log " done."
   ) else
    log " File \"%s\" does not exists. Don't load bases." filename;
  flush_all ()
```

Chapter 45

Handling of server work

The module `Work` defines routines needed to handling the work requested by the clients.

262a ⟨*work.ml* 262a⟩≡ 262b ▷
```
(* copyright 2003-2005 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Messages_aux
open Messages_clnt
open Srvflags
```

45.1 Client context

We define a context used by each client. This context contains:

- `auth_login`: the authenticated login of the remote client;

fixme: We should store a creation date to do garbage collection (or any other mean to avoid cookie addition). It is a potential DoS: a malicious client could create cookies at vitam. FIXME

262b ⟨*work.ml* 262a⟩+≡ ◁ 262a 262c ▷
```
type client_context = {
    mutable auth_login : Participants.authenticated_participant;
  }
```

This context is stored in a hash-table indexed by client cookie: `context_table`. This cookie is generated at first login and removed when the client unconnect itself.

fixme: We should remove the cookie also when the tcp socket is closed. FIXME

262c ⟨*work.ml* 262a⟩+≡ ◁ 262b 262d ▷
```
let context_table : (cookie_t, client_context) Hashtbl.t = Hashtbl.create 3
```

The helper function `clear_context_table` reset it to a clean state.

262d ⟨*work.ml* 262a⟩+≡ ◁ 262c 262e ▷
```
let clear_context_table () = Hashtbl.clear context_table
```

Helper function `get_cookie` returns a valid new cookie. It checks that the cookie is valid and is not already used

262e ⟨*work.ml* 262a⟩+≡ ◁ 262d 263a ▷
```
let rec get_cookie () =
  let c = Random.bits () in
  if Hashtbl.mem context_table c then get_cookie ()
  else c
```

Helper function `store_new_context` creates an new context for participant authenticated as `auth` and returns the generated cookie.

263a ⟨*work.ml* 262a⟩+≡ ◁262e 263b▷
```
let store_new_context auth =
  let cookie = get_cookie () in
  let ctxt = { auth_login = auth } in
  Hashtbl.add context_table cookie ctxt;
  cookie
```

Helper function `get_auth_login` returns the authenticated login for the given `cookie`.

263b ⟨*work.ml* 262a⟩+≡ ◁263a 263c▷
```
let get_auth_login cookie =
  try
    (Hashtbl.find context_table cookie).auth_login
  with Not_found -> Participants.Anonymous
```

45.2 Handling of server RPC calls

Auxiliary function `do_if_administrator` executes function `f` if the client context identified by `cookie` has administrator rights. Otherwise it calls `error_f`.

263c ⟨*work.ml* 262a⟩+≡ ◁263b 263d▷
```
let do_if_administrator cookie f error_f =
  if Participants.is_administrator (get_auth_login cookie) then (
    f ()
   ) else
    error_f ()
```

Auxiliary function `do_if_classifier` executes function `f` if the client context identified by `cookie` has classifier rights. Otherwise it calls `error_f`.

263d ⟨*work.ml* 262a⟩+≡ ◁263c 264a▷
```
let do_if_classifier cookie f error_f =
  if Participants.is_classifier (get_auth_login cookie) then (
    f ()
   ) else
    error_f ()
```

45.2.1 Login methods

We define the call-backs, one for each RPC method defined in the server.

`login` simply authenticate the participant against the Participant base using the given `login` and `password` and returns a session cookie. In case the authentification fails, an `Anonymous` context is created.

264a ⟨*work.ml* 262a⟩+≡ ◁263d 264b▷

```
let login (client_protocol_version, login, password) =
  dbg "RPC login(%d, \"%s\", **passw**)" client_protocol_version login;
  let auth = Participants.authenticate_participant login password in
  let cookie = store_new_context auth in
  match auth with
  | Participants.Authenticated_individual _ ->
      dbg " => individual participant '%s' logged in (cookie:%d)" login cookie;
      { login_return_code = rt_ok;
        server_protocol_version = Rtypes.int_of_uint4 protocol_version;
        login_cookie = cookie; }
  | Participants.Authenticated_delegate _ ->
      dbg " => delegate participant '%s' logged in (cookie:%d)" login cookie;
      { login_return_code = rt_ok;
        server_protocol_version = Rtypes.int_of_uint4 protocol_version;
        login_cookie = cookie; }
  | Participants.Anonymous ->
      dbg " => participant '%s' failed to log in, remains Anonymous (cookie:%d)"
        login cookie;
      { login_return_code = rt_bad_login;
        server_protocol_version = Rtypes.int_of_uint4 protocol_version;
        login_cookie = cookie; }
```

`googbye` terminates a client session by removing its server side context.

264b ⟨*work.ml* 262a⟩+≡ ◁264a 264c▷

```
let goodbye cookie =
  dbg "RPC goodbye(%d)" cookie;
  if Hashtbl.mem context_table cookie then
    Hashtbl.remove context_table cookie
  else
    log " => warning: goodbye with invalid cookie (%d)" cookie
```

45.2.2 Timestamp method

264c ⟨*work.ml* 262a⟩+≡ ◁264b 264d▷

```
let last_timestamp_list_creation = ref 0.0
let timestamp_list = ref None
```

264d ⟨*work.ml* 262a⟩+≡ ◁264c 265a▷

```
let build_timestamp_list () =
  { gt_return_code = rt_ok;
    gt_participant = Timestamp.compress (Participants.timestamp_list ());
    gt_question = Timestamp.compress (Posbase.timestamp_list ());
    gt_tag = Timestamp.compress (Classification.tag_timestamp_list ()); }
```

Helper function `update_timestamp_list` stores end returns the timestamps of questions, tags and participants in a compressed format.

265a ⟨*work.ml* 262a⟩+≡ ◁264d 265b▷

```
let update_timestamp_list () =
  let timer = Perf.timer_start () in
  let tl = build_timestamp_list () in
  last_timestamp_list_creation := Unix.time ();
  timestamp_list := Some tl;
  Perf.timer_stop_and_record "Work.update_timestamp_list" timer;
  tl
```

`get_timestamps` returns the timestamps of questions, tags and participants in a compressed format.

265b ⟨*work.ml* 262a⟩+≡ ◁265a 265c▷

```
let get_timestamps (cookie) =
  dbg "RPC get_timestamps(%d)" cookie;
  let age = Unix.time () -. !last_timestamp_list_creation in
  match !timestamp_list with
  | None ->
      dbg "update timestamp list";
      update_timestamp_list ()
  | Some _ when age >= 120.0 -> (* older than 2 minutes *)
      dbg "update timestamp list";
      update_timestamp_list ()
  | Some tl -> tl
```

45.2.3 Question methods

`new_question` add a new question with description `q_desc` in the Position base.

265c ⟨*work.ml* 262a⟩+≡ ◁265b 265d▷

```
let new_question (cookie, q_desc) =
  log "@[RPC add_question(%d,@ \"%s\")@]" cookie q_desc;
  let auth = get_auth_login cookie in
  try
    let id = Posbase.add_question ~q_desc ~author:auth () in
    log " => question %d added." id;
    Io.save_bases_xml ();
    { question_id_return_code = rt_ok; question_id_id = id }
  with Posbase.Question_already_in_base ->
    log " !! question already in base";
    { question_id_return_code = rt_already_exists; question_id_id = 0 }
  | Norm.Invalid_format ->
      log " !! invalid quesiton format";
      { question_id_return_code = rt_bad_format; question_id_id = -1 }
```

`get_question_id` returns the identifier of question of descriptor `q_desc`.

265d ⟨*work.ml* 262a⟩+≡ ◁265c 266a▷

```
let get_question_id (cookie, q_desc) =
  dbg "@[RPC get_question_id(%d,@ \"%s\")@]" cookie q_desc;
  try
    let id = Posbase.id_of_desc q_desc in
    dbg " => found, id:%d" id;
    { question_id_return_code = rt_ok; question_id_id = id }
  with Not_found ->
    dbg " !! not found";
    { question_id_return_code = rt_not_found; question_id_id = 0 }
```

`add_response` add a new response to a question.

266a ⟨*work.ml* 262a⟩+≡ ◁265d 266b▷

```
let add_response (cookie, id, response, external_link) =
  log "@[RPC add_response(%d, %d,@ \"%s\",@ \"%s\")@]"
    cookie id response external_link;
  try
    let auth = get_auth_login cookie in
    if external_link <> "" then
      Posbase.add_response
        ~q_id:id ~r_desc:response ~author:auth
        ~extlink:(Some external_link) ()
    else
      Posbase.add_response
        ~q_id:id ~r_desc:response ~author:auth ~extlink:None ();
    log " => response added.";
    Io.save_bases_xml ();
    rt_ok
  with
  | Posbase.Response_already_made ->
      log " !! response already exists";
      rt_already_exists
  | Not_found ->
      log " !! question not found";
      rt_not_found
  | Norm.Invalid_format ->
      log " !! invalid response format";
      rt_bad_format
```

Binding to `Posbase.max_id`.

266b ⟨*work.ml* 262a⟩+≡ ◁266a 267▷

```
let max_question_id cookie =
  dbg "RPC max_question_id(%d)" cookie;
  let max = Posbase.max_id () in
  dbg " => return max_question_id:%d" max;
  { max_question_id_rc = rt_ok; max_question_id = max; }
```

Auxiliary function `get_as_question` returns the details of question of identifier `q_id` as a data structure suitable to be sent on the network. It can raise `Not_found` exception if the `q_id` is invalid. Most of the code consists in rewriting data from `Posbase` data structures to `Messages_aux` ones.

267 ⟨*work.ml* 262a⟩+≡ ◁266b 268▷

```
let get_as_question ~q_id =
  let module P = Posbase in
  let q = P.get_position ~q_id in
  let status =
    match q.P.question.P.status with
    | P.Tagging_only -> tagging_only
    | P.Public -> public in
  let limit_date =
    match q.P.question.P.limit_date with
      None -> Int64.zero
    | Some date -> date in
  let translate_response r =
    let link =
      match r.P.external_link with
        None -> ""
      | Some link -> link in
    { r_info_desc = r.P.r_desc;
      r_info_link = link; } in
  { q_id = q_id;
    q_timestamp = P.get_timestamp ~q_id;
    q_desc = q.P.question.P.q_desc;
    q_info_limit_date = limit_date;
    q_info_status = status;
    q_info_responses = Array.map translate_response q.P.responses;
    q_info_num_votes = Hashtbl.length q.P.votes;
    q_info_elected_responses = Array.of_list q.P.elected_responses; }
```

In `question_info`, we return information on all questions of which identifiers are in [`base_id`, `base_id` + `number`). In case a question is not public, it is returned only if the client has classifier rights.

268 ⟨*work.ml* 262a⟩+≡ ◁267 269a▷

```
let question_info (cookie, base_id, number) =
  dbg "RPC question_info(@[%d,@ %d,@ %d@])" cookie base_id number;
  let questions = ref [] in
  let rec fill_questions q_id max_id =
    if q_id < max_id then (
      try
        let q = get_as_question ~q_id in
        if q.q_info_status = public then
          questions := q :: !questions (* public question *)
        else
          (* providing information is restricted to classifier *)
          do_if_classifier cookie
            (fun () -> questions := q :: !questions)
            (fun () -> ());
        fill_questions (q_id + 1) max_id;
      with Not_found ->
        fill_questions (q_id + 1) max_id
     ) in
  if number > Rtypes.int_of_uint4 Messages_aux.max_number_ids then (
    dbg " !! request to much ids";
    { question_info_rc = rt_request_too_much_ids;
      question_info = Array.of_list []; }
   ) else (
    fill_questions base_id (base_id + number);
    let add_to_str str q = str ^ (string_of_int q.q_id) ^ "," in
    dbg " => return info for: %s" (List.fold_left add_to_str "" !questions);
    { question_info_rc = rt_ok;
      question_info = Array.of_list !questions; }
   )
```

Binding to `Posbase.set_status`.

269a ⟨*work.ml* 262a⟩+≡ ◁268 269b ▷

```
let set_question_status (cookie, q_id, new_status) =
  do_if_classifier cookie
    (fun () ->
      try
        let status, str_status =
          match new_status with
          | s when s = public -> Posbase.Public, "public"
          | s when s = tagging_only -> Posbase.Tagging_only, "tagging_only"
          | _ -> failwith "bad argument" in
        log "RPC set_question_status(@[%d,@ %d,@ %s@])"
          cookie q_id str_status;
        Posbase.set_status ~q_id ~status;
        log " => ok.";
        Io.save_bases_xml ();
        rt_ok
      with Failure "bad argument" ->
        log " !! invalid status argument";
        rt_bad_status
      | Not_found ->
          log " !! question not found";
          rt_not_found
    )
    (fun () -> rt_not_enough_rights)
```

Helper function `string_of_int_list` returns the list of integers `l` as a printable string.

269b ⟨*work.ml* 262a⟩+≡ ◁269a 270a ▷

```
let string_of_int_list l =
  (List.fold_left (fun str i -> str ^ (string_of_int i) ^ "; ") "[" l) ^ "]"
```

Binding to `Posbase.set_vote` and `Posbase.update_elected_responses`.

270a ⟨*work.ml* 262a⟩+≡ ◁269b 270b▷

```
let vote (cookie, q_id, choices_array) =
  let choices = Array.to_list choices_array in
  let choices_str = string_of_int_list choices in
  dbg "@[RPC vote(%d,@ %d,@ %s)@]" cookie q_id choices_str;
  try (
    let do_vote author =
      Posbase.set_vote ~q_id ~author ~vote:choices;
      Posbase.update_elected_responses ~q_id;
      Posbase.update_timestamp ~q_id;
      dbg " => vote stored.";
      Io.save_bases_xml ();
      rt_ok in
    let auth = get_auth_login cookie in
    let module P = Participants in
    match auth with
    | P.Anonymous ->
        dbg " !! Anonymous can't vote";
        rt_anonymous_cannot_vote
    | P.Authenticated_individual login -> do_vote (Posbase.Individual login)
    | P.Authenticated_delegate _ -> do_vote (Posbase.Delegate auth)
   )
  with
  | Not_found ->
      dbg " !! something not found";
      rt_not_found
  | Posbase.Vote_choice_not_found _ ->
      dbg " !! vote choice not found";
      rt_vote_choice_not_found
  | Posbase.Duplicated_vote_choice _ ->
      dbg " !! duplicate vote choice";
      rt_duplicate_vote_choice
  | Posbase.Question_not_found ->
      dbg " !! question not found";
      rt_not_found
```

Helper function `can_get_vote` returns true if `login` is the login of a delegate of if it is the same login of the individual identified with `cookie`.

270b ⟨*work.ml* 262a⟩+≡ ◁270a 271a▷

```
let can_get_vote cookie login =
  let module P = Participants in
  let same_login login =
    let auth = get_auth_login cookie in
    match auth with
    | P.Authenticated_individual l -> l = login
    | P.Anonymous
    | P.Authenticated_delegate _ -> false in
  (P.is_delegate login) || (same_login login)
```

Binding to `Posbase.get_vote`. We check that we can return the vote first with `can_get_vote`.

271a ⟨*work.ml* 262a⟩+≡ ◁270b 271b▷

```
let get_vote (cookie, q_id, login) =
  dbg "@[RPC get_vote(%d,@ %d,@ %s)@]" cookie q_id login;
  if can_get_vote cookie login then
    try
      let vote = Posbase.get_vote ~q_id ~login in
      dbg " => %s" (string_of_int_list vote);
      { get_vote_rc = rt_ok;
        get_vote = Array.of_list vote }
    with Posbase.Question_not_found ->
      dbg " !! question not found";
      { get_vote_rc = rt_not_found;
        get_vote = Array.of_list [] }
  else (
    dbg " !! not enough rights";
    { get_vote_rc = rt_not_enough_rights;
      get_vote = Array.of_list [] }
   )
```

45.2.4 Server administration methods

Function `schedule_server_halt` is called when we request to stop the server. Following approach is recommended by Gerd Stolpmann: disallow any further incoming calls and then call `Rpc_server.stop_server` after a certain period of time. We use a small workaround here: as the `srv` cannot be known when `stop_server` is given as a parameter of `Messages_srv.Demexp.V1.create_server` call which creates the server, we use a global variable (`server_descriptor`) which is set just after server creation (see code chunk 293). Not very clean but it works. :)

271b ⟨*work.ml* 262a⟩+≡ ◁271a 271c▷

```
let server_descriptor = ref None

let schedule_server_halt esys =
  match !server_descriptor with
  | None -> assert(false)
  | Some srv ->
      (* disallow any further incoming calls *)
      Rpc_server.set_session_filter srv (fun _ -> `Deny);
      (* After a short period of time (e.g. 7 seconds), stop the server *)
      let wait_time = if !flag_autotests then 0.1 else 7.0 in
      let g = Unixqueue.new_group esys in
      Unixqueue.once esys g wait_time (fun () -> Rpc_server.stop_server srv);
      log " => stop the server in %f seconds" wait_time
```

Method `stop_server` is called by a client to shutdown the server program.

271c ⟨*work.ml* 262a⟩+≡ ◁271b 272a▷

```
let stop_server esys cookie =
  log "RPC stop_server(%d)" cookie;
  do_if_administrator cookie
    (fun () ->
      schedule_server_halt esys;
      rt_ok)
    (fun () ->
      log " !! not enough rights";
      rt_not_enough_rights)
```

Binding to `Perf.timers_as_string`.

272a ⟨*work.ml* 262a⟩+≡ ◁271c 272b▷

```
let server_timers cookie =
  do_if_administrator cookie
    (fun () ->
      let str_timers = Perf.timers_as_string () in
      dbg " => timers:\n%s" str_timers;
      str_timers
    )
    (fun () ->
      log " !! not enough rights";
      "")
```

45.2.5 Participant administration methods

Function `max_participant_id` returns the biggest participant identifier in the current base.

272b ⟨*work.ml* 262a⟩+≡ ◁272a 273▷

```
let max_participant_id cookie =
  dbg "RPC max_participant_id(%d)" cookie;
  do_if_administrator cookie
    (fun () ->
      let max = Participants.max_id () in
      dbg " => return max_participant:%d" max;
      { max_participant_id_rc = rt_ok;
        max_participant_id = max; }
    )
    (fun () ->
      dbg " !! not enough rights";
      { max_participant_id_rc = rt_not_enough_rights;
        max_participant_id = -1; })
```

Function `participant_info` returns details on participants whose identifier is between `base_id` and `base_id` + `number`. If `number` is bigger than `max_number_ids` (as defined in `net/messages.xdr`), then `rt_request_too_much_ids` is returned. It is possible that this function returns `rt_ok` with less information that the requested `number` because one or more ids are missing.

273 ⟨*work.ml* 262a⟩+≡ ◁272b 274▷

```
let participant_info (cookie, base_id, number) =
  dbg "RPC participant_info(@[%d,@ %d,@ %d@])"
    cookie base_id number;
  do_if_administrator cookie
    (fun () ->
      if number > Rtypes.int_of_uint4 Messages_aux.max_number_ids then
        { participant_info_rc = rt_request_too_much_ids;
          participant_info = Array.of_list []; }
      else (
        let l = ref [] in
        for i = base_id to base_id + number - 1 do
          try
            let login, _, password, groups =
              Participants.details_of_id i in
            l := { info_id = i;
                   info_timestamp = Participants.get_timestamp i;
                   info_login = login;
                   info_password = password;
                   info_groups = Array.of_list groups; } :: !l;
          with Not_found -> ()
        done;
        let add_to_str str e = str ^ (string_of_int e.info_id)  ^ "," in
        dbg " => return info for: %s" (List.fold_left add_to_str "" !l);
        { participant_info_rc = rt_ok;
          participant_info = Array.of_list !l; }
       )
    )
    (fun () ->
      dbg " !! not enough rights";
      { participant_info_rc = rt_not_enough_rights;
        participant_info = Array.of_list []; })
```

Binding to function `Participants.add_participant`.

274 ⟨*work.ml* 262a⟩+≡ ◁273 275a▷

```
let add_participant (cookie, login, password, groups) =
  let group_list = Array.to_list groups in
  let groups_str =
    "[" ^
    (List.fold_left (fun g str -> str ^ ";" ^ g) "" group_list)
    ^ "]" in
  log "RPC add_participant(@[%d,@ **login**,@ **passw**, %s@])"
    cookie groups_str;
  dbg "    login:\"%s\"" login;
  do_if_administrator cookie
    (fun () ->
      try
        let id = Participants.add_participant login password group_list in
        log " => participant #%d added to Participant base" id;
        Io.save_bases_xml ();
        { add_participant_rc = rt_ok;
          add_participant_id = id; }
      with Participants.Already_in_base ->
        log " !! participant already exists";
        { add_participant_rc = rt_already_exists;
          add_participant_id = -1; }
      | Norm.Invalid_format ->
          log " !! bad format";
          { add_participant_rc = rt_bad_format;
            add_participant_id = -1; }
    )
    (fun () ->
      log " !! not enough rights";
      { add_participant_rc = rt_not_enough_rights;
        add_participant_id = -1; })
```

Function `update_participant` update or create a new participant with given information (login, password and groups).

275a ⟨*work.ml* 262a⟩+≡ ◁274 275b▷

```
let update_participant (cookie, login, password, groups) =
  let group_list = Array.to_list groups in
  let groups_str =
    "[" ^
    (List.fold_left (fun g str -> str ^ ";" ^ g) "" group_list)
    ^ "]" in
  log "RPC update_participant(@[%d,@ **login**,@ **passw**, %s@])"
    cookie groups_str;
  dbg "    login:\"%s\"" login;
  do_if_administrator cookie
    (fun () ->
      try
        Participants.update_participant login password group_list;
        log " => participant updated in Participant base";
        Io.save_bases_xml ();
        rt_ok
      with Not_found ->
        log " !! participant not found";
        rt_not_found
    )
    (fun () ->
      log " !! not enough rights";
      rt_not_enough_rights)
```

Binding to `Participants.remove_participant`.

275b ⟨*work.ml* 262a⟩+≡ ◁275a 275c▷

```
let remove_participant (cookie, login) =
  log "RPC remove_participant(@[%d,@ **login**@])" cookie;
  dbg "    login:\"%s\"" login;
  do_if_administrator cookie
    (fun () ->
      Participants.remove_participant login;
      log " => participant removed of Participant base";
      Io.save_bases_xml ();
      rt_ok
    )
    (fun () ->
      log " !! not enough rights";
      rt_not_enough_rights)
```

45.2.6 Tag administration methods

Function `max_tag_id` returns the biggest tag identifier in the current base.

275c ⟨*work.ml* 262a⟩+≡ ◁275b 276a▷

```
let max_tag_id cookie =
  dbg "RPC max_tag_id(%d)" cookie;
  let max = Classification.max_id () in
  dbg " => return max_tag:%d" max;
  { max_tag_id_rc = rt_ok;
    max_tag_id = max; }
```

Binding to `Classification.create_tag`.

276a ⟨*work.ml* 262a⟩+≡ ◁275c 276b▷

```
let create_tag (cookie, tag_label) =
  log "RPC create_tag(@[%d,@ \"%s\"@])" cookie tag_label;
  do_if_classifier cookie
    (fun () ->
      try
        let id = Classification.create_tag ~tag_label in
        log " => tag %d:\"%s\" added to Classification base" id tag_label;
        Io.save_bases_xml ();
        { create_tag_rc = rt_ok;
          create_tag_id = id; }
      with Id.Tag.Already_exists ->
        log " !! tag already exists";
        { create_tag_rc = rt_already_exists;
          create_tag_id = -1; }
    )
    (fun () ->
      log " !! not enough rights";
      { create_tag_rc = rt_not_enough_rights;
        create_tag_id = -1; })
```

Function `tag_info` has similar behavior as `participant_info` (see code chunk 273).

276b ⟨*work.ml* 262a⟩+≡ ◁276a 277a▷

```
let tag_info (cookie, base_id, number) =
  dbg "RPC tag_info(@[%d,@ %d,@ %d@])" cookie base_id number;
  if number > Rtypes.int_of_uint4 Messages_aux.max_number_ids then (
    dbg " !! request to much tags";
    { tag_info_rc = rt_request_too_much_ids;
      tag_info = Array.of_list []; }
   ) else (
    let l = ref [] in
    for i = base_id to base_id + number - 1 do
      try
        let label = Classification.get_tag_label i in
        l := { a_tag_id = i;
               a_tag_timestamp = Classification.get_tag_timestamp i;
               a_tag_label = label; } :: !l;
      with Not_found -> ()
    done;
    let add_to_str str e = str ^ (string_of_int e.a_tag_id) ^ "," in
    dbg " => return info for: %s"
      (List.fold_left add_to_str "" !l);
    { tag_info_rc = rt_ok;
      tag_info = Array.of_list !l; }
   )
```

Binding to `Classification.change_tag_label`.

277a ⟨*work.ml* 262a⟩+≡ ◁276b 277b▷

```
let update_tag (cookie, id, tag_label) =
  log "RPC update_tag(@[%d,@ %d,@ \"%s\"@])" cookie id tag_label;
  do_if_classifier cookie
    (fun () ->
      try
        Classification.change_tag_label ~tag_id:id ~tag_label;
        log " => tag %d:\"%s\" updated in Classification base"
          id tag_label;
        Io.save_bases_xml ();
        rt_ok
      with Not_found -> rt_not_found
    )
    (fun () ->
      log " !! not enough rights";
      rt_not_enough_rights)
```

45.2.7 Question tagging

Binding to `Classification.tag_question`.

277b ⟨*work.ml* 262a⟩+≡ ◁277a 278a▷

```
let tag_question (cookie, q_id, tag_id) =
  log "RPC tag_question(@[%d,@ %d,@ %d@])" cookie q_id tag_id;
  do_if_classifier cookie
    (fun () ->
      try
        Classification.tag_question ~q_id ~tag_id;
        Posbase.update_timestamp ~q_id;
        log " => tag_id:%d added to question id:%d" tag_id q_id;
        Io.save_bases_xml ();
        rt_ok
      with Not_found ->
        log " !! not found";
        rt_not_found
    )
    (fun () ->
      log " !! not enough rights";
      rt_not_enough_rights)
```

Binding to `Classification.untag_question`.

278a ⟨*work.ml* 262a⟩+≡ ◁277b 278b▷

```
let untag_question (cookie, q_id, tag_id) =
  log "RPC untag_question(@[%d,@ %d,@ %d@])" cookie q_id tag_id;
  do_if_classifier cookie
    (fun () ->
      try
        Classification.untag_question ~q_id ~tag_id;
        Posbase.update_timestamp ~q_id;
        log " => tag_id:%d removed from question id:%d" tag_id q_id;
        Io.save_bases_xml ();
        rt_ok
      with Not_found ->
        log " !! not found";
        rt_not_found
    )
    (fun () ->
      log " !! not enough rights";
      rt_not_enough_rights)
```

Binding to `Classification.get_question_tags`.

278b ⟨*work.ml* 262a⟩+≡ ◁278a 278c▷

```
let get_question_tags (cookie, q_id) =
  dbg "RPC get_question_tags(@[%d,@ %d@])" cookie q_id;
  let tags = Classification.get_question_tags ~q_id in
  (* debug *)
  let list_as_str =
    List.fold_left (fun s e -> s ^ "," ^ (string_of_int e)) "" tags in
  dbg " => %s" list_as_str;
  Array.of_list tags
```

Implements RPC `tag_set_of_question_group`. See its description for details (see code chunk 22d).

278c ⟨*work.ml* 262a⟩+≡ ◁278b 279a▷

```
let tag_set_of_question_group (cookie, base_qid, number) =
  dbg "RPC tag_set_of_question_group(@[%d,@ %d,@ %d@])"
    cookie base_qid number;
  if number > Rtypes.int_of_uint4 Messages_aux.max_number_ids then (
    dbg " !! request to much ids";
    { tag_set_group_rc = rt_request_too_much_ids;
      tag_set_group = Array.of_list []; }
   ) else (
    let l = ref [] in
    for q_id = base_qid to base_qid + number - 1 do
      let tags = Classification.get_question_tags ~q_id in
      l := { tag_set_qid = q_id;
             tag_set = Array.of_list tags; } :: !l;
    done;
    dbg " => tags for questions in %d-%d range."
      base_qid (base_qid + number - 1);
    { tag_set_group_rc = rt_ok;
      tag_set_group = Array.of_list !l }
   )
```

45.3 Work module initialization

This function is called to reset all the components of the `Work` module in a clean state.

279a ⟨*work.ml* 262a⟩+≡ ◁278c 279b▷

```
let initialize () =
  clear_context_table ();
  if Config.do_autotests then Random.init 1
  else Random.self_init ()
```

45.4 Automatic tests with a dummy client

To test the server, we define the following sequence of method calls. All the called method are defined in `Messages_clnt` module.

45.4.1 Login

We first login onto the system as normal user. We check that all reserved operations are indeed reserved (**todo**: we should check more operations.). TO DO

279b ⟨*work.ml* 262a⟩+≡ ◁279a 279c▷

```
let test_message_sequence client =
  if Config.do_autotests then (
    let r = Demexp.V1.login client (Rtypes.int_of_uint4 protocol_version,
                                    "toto", "") in
    assert(r.login_return_code = rt_bad_login);
    assert(r.server_protocol_version = Rtypes.int_of_uint4 protocol_version);
    let c = r.login_cookie in
    assert(Hashtbl.find context_table c
             = { auth_login = Participants.Anonymous });
    assert(Demexp.V1.stop_server client c = rt_not_enough_rights);
    assert(Demexp.V1.tag_question client (c, 0, 0)
             = rt_not_enough_rights);
    assert(Demexp.V1.set_question_status client (c, 0, public)
             = rt_not_enough_rights);
    Demexp.V1.goodbye client c;
    assert(Hashtbl.mem context_table c = false);
```

We then re-log into the system. We used "root" identifiers defined in code chunk 199c.

279c ⟨*work.ml* 262a⟩+≡ ◁279b 280▷

```
    let r = Demexp.V1.login client (Rtypes.int_of_uint4 protocol_version,
                                    "root", "demexp") in
    assert(r.login_return_code = rt_ok);
    let cookie = r.login_cookie in
```

45.4.2 Participant management

We add a new participant `fred`. Before and after it, we check the maximum participant identifier. We also check the two error cases: you can't update a non existing participant and you can't add twice a participant.

280 ⟨*work.ml* 262a⟩+≡ ◁279c 281▷

```
let fred = "fred" and fred_pass = "a pass" in
let groups =
  Array.of_list [Participants.administration_group; "demexp core"] in
assert(Demexp.V1.update_participant client (cookie, fred, fred_pass, groups)
         = rt_not_found);
assert(Demexp.V1.max_participant_id client cookie
         = { max_participant_id_rc = rt_ok;
             max_participant_id = 0; });
assert(Demexp.V1.add_participant client (cookie, fred, fred_pass, groups)
         = { add_participant_rc = rt_ok;
             add_participant_id = 1; });
assert(Demexp.V1.add_participant client (cookie, fred, fred_pass, groups)
         = { add_participant_rc = rt_already_exists;
             add_participant_id = -1; });
assert(Demexp.V1.add_participant client (cookie, "bad/login", "",
                                         Array.of_list [])
         = { add_participant_rc = rt_bad_format;
             add_participant_id = -1; });
assert(Demexp.V1.max_participant_id client cookie
         = { max_participant_id_rc = rt_ok;
             max_participant_id = 1; });
```

We also check that the stored information is correct. We then update the info on fred and check that once again, all is correct (especially the password which has not been modified).

281 ⟨*work.ml* 262a⟩+≡ ◁280 282a▷

```
let groups2 =
  Array.of_list [Participants.administration_group; "demexp core"; "a group"] in
assert(Demexp.V1.participant_info client (cookie, 0, 2)
         = { participant_info_rc = rt_ok;
             participant_info
               = Array.of_list
               [{ info_id = 1;
                  info_timestamp = Participants.get_timestamp 1;
                  info_login = fred;
                  info_password = Participants.md5_hash fred_pass;
                  info_groups = groups; };
                { info_id = 0;
                  info_timestamp = Participants.get_timestamp 0;
                  info_login = "root";
                  info_password = Participants.md5_hash "demexp";
                  info_groups
                    = Array.of_list [Participants.administration_group;
                                     Participants.classification_group]; };
                                ] });
assert(Demexp.V1.update_participant client (cookie, fred, fred_pass, groups2)
         = rt_ok);
assert(Demexp.V1.participant_info client (cookie, 0, 2)
         = { participant_info_rc = rt_ok;
             participant_info
               = Array.of_list
               [{ info_id = 1;
                  info_timestamp = Participants.get_timestamp 1;
                  info_login = fred;
                  info_password = Participants.md5_hash fred_pass;
                  info_groups = groups2; };
                { info_id = 0;
                  info_timestamp = Participants.get_timestamp 0;
                  info_login = "root";
                  info_password = Participants.md5_hash "demexp";
                  info_groups
                    = Array.of_list [Participants.administration_group;
                                     Participants.classification_group]; };
               ] });
```

We check that the exact number of information is returned (i.e. one login for one participant. We also check two error cases: that the requesting of unknown ids returns empty info and that an error is returned in case we request to much information.

282a ⟨*work.ml* 262a⟩+≡ ◁281 282b ▷

```
assert(Demexp.V1.participant_info client (cookie, 0, 1)
         = { participant_info_rc = rt_ok;
             participant_info
               = Array.of_list
               [{ info_id = 0;
                  info_timestamp = Participants.get_timestamp 0;
                  info_login = "root";
                  info_password = Participants.md5_hash "demexp";
                  info_groups
                    = Array.of_list [Participants.administration_group;
                                     Participants.classification_group]; };
                ] });
assert(Demexp.V1.participant_info client (cookie, 42, 1)
         = { participant_info_rc = rt_ok;
             participant_info
               = Array.of_list [] });
let too_much = Rtypes.int_of_uint4 Messages_aux.max_number_ids + 1 in
assert(Demexp.V1.participant_info client (cookie, 0, too_much)
         = { participant_info_rc = rt_request_too_much_ids;
             participant_info
               = Array.of_list [] });
```

We check that we can remove a participant.

282b ⟨*work.ml* 262a⟩+≡ ◁282a 283a ▷

```
assert(Demexp.V1.remove_participant client (cookie, fred) = rt_ok);
assert(Demexp.V1.participant_info client (cookie, 0, 2)
         = { participant_info_rc = rt_ok;
             participant_info
               = Array.of_list
               [{ info_id = 0;
                  info_timestamp = Participants.get_timestamp 0;
                  info_login = "root";
                  info_password = Participants.md5_hash "demexp";
                  info_groups
                    = Array.of_list [Participants.administration_group;
                                     Participants.classification_group]; };
                ] });
```

45.4.3 Tags

We firstly create some new tags. We check that the maximum tag identifier is correctly updated. We also check the error case when the tag already exists.

283a ⟨*work.ml* 262a⟩+≡ ◁282b 283b▷

```
assert(Demexp.V1.max_tag_id client (cookie)
         = { max_tag_id_rc = rt_ok;
             max_tag_id = -1; });
let demexp_tag = "demexp" in
assert(Demexp.V1.create_tag client (cookie, demexp_tag)
         = { create_tag_rc = rt_ok;
             create_tag_id = 0; });
let politics_tag = "politics" in
assert(Demexp.V1.create_tag client (cookie, politics_tag)
         = { create_tag_rc = rt_ok;
             create_tag_id = 1; });
assert(Demexp.V1.max_tag_id client (cookie)
         = { max_tag_id_rc = rt_ok;
             max_tag_id = 1; });
assert(Demexp.V1.create_tag client (cookie, politics_tag)
         = { create_tag_rc = rt_already_exists;
             create_tag_id = -1; });
```

We then check that added tags are correctly stored in the server. We then create a new tag, update it a tag and check that modification is done.

283b ⟨*work.ml* 262a⟩+≡ ◁283a 284a▷

```
assert(Demexp.V1.tag_info client (cookie, 0, 2)
         = { tag_info_rc = rt_ok;
             tag_info
               = Array.of_list [{ a_tag_id = 1;
                                  a_tag_timestamp
                                    = Classification.get_tag_timestamp 1;
                                  a_tag_label = politics_tag; };
                                { a_tag_id = 0;
                                  a_tag_timestamp
                                    = Classification.get_tag_timestamp 0;
                                  a_tag_label = demexp_tag; };
                               ]});
assert(Demexp.V1.create_tag client (cookie, "bad label")
         = { create_tag_rc = rt_ok;
             create_tag_id = 2; });
assert(Demexp.V1.update_tag client (cookie, 2, "good label") = rt_ok);
assert(Demexp.V1.tag_info client (cookie, 0, 3)
         = { tag_info_rc = rt_ok;
             tag_info
               = Array.of_list [{ a_tag_id = 2;
                                  a_tag_timestamp
                                    = Classification.get_tag_timestamp 2;
                                  a_tag_label = "good label"; };
                                { a_tag_id = 1;
                                  a_tag_timestamp
                                    = Classification.get_tag_timestamp 1;
                                  a_tag_label = politics_tag; };
                                { a_tag_id = 0;
                                  a_tag_timestamp
                                    = Classification.get_tag_timestamp 0;
                                  a_tag_label = demexp_tag; };
                               ]});
```

45.4.4 Question tagging

We add a tag on a question, then remove it. We also check the error case when a tag identifier does not exist.

284a ⟨*work.ml* 262a⟩+≡ ◁283b 284b▷

```
assert(Demexp.V1.tag_question client (cookie, 0(*q_id*), 0(*tag_id*))
         = rt_ok);
assert(Demexp.V1.get_question_tags client (cookie, 0(*q_id*))
         = Array.of_list [0]);
assert(Demexp.V1.tag_question client (cookie, 0(*q_id*), 1(*tag_id*))
         = rt_ok);
assert(Demexp.V1.get_question_tags client (cookie, 0(*q_id*))
         = Array.of_list [1;0]);
assert(Demexp.V1.untag_question client (cookie, 0(*q_id*), 0(*tag_id*))
         = rt_ok);
assert(Demexp.V1.get_question_tags client (cookie, 0(*q_id*))
         = Array.of_list [1]);
assert(Demexp.V1.tag_question client (cookie, 0, 4) = rt_not_found);
assert(Demexp.V1.get_question_tags client (cookie, 42(*q_id*))
         = Array.of_list []);
```

We add tags for a second question and check we can get all of them with a single RPC. Also check an error case (request to much items at once).

284b ⟨*work.ml* 262a⟩+≡ ◁284a 284c▷

```
assert(Demexp.V1.tag_question client (cookie, 1(*q_id*), 0(*tag_id*))
         = rt_ok);
let ret =
  Demexp.V1.tag_set_of_question_group client
    (cookie, 0(*base_qid*), 3(*number*)) in
assert(ret.tag_set_group_rc = rt_ok);
assert(ret.tag_set_group.(0) = { tag_set_qid = 2;
                                 tag_set = Array.of_list []; });
assert(ret.tag_set_group.(1) = { tag_set_qid = 1;
                                 tag_set = Array.of_list [0]; });
assert(ret.tag_set_group.(2) = { tag_set_qid = 0;
                                 tag_set = Array.of_list [1]; });
(* error case *)
assert(Demexp.V1.tag_set_of_question_group client (cookie, 0, 10000)
         = { tag_set_group_rc = rt_request_too_much_ids;
             tag_set_group = Array.of_list [] });
```

45.4.5 Questions

We check the initial maximum question identifier.

284c ⟨*work.ml* 262a⟩+≡ ◁284b 285a▷

```
assert(Demexp.V1.max_question_id client cookie
         = { max_question_id_rc = rt_ok; max_question_id = -1; });
```

We add a new question and check that it cannot be added twice. We also check that a question is correctly normalized.

285a ⟨*work.ml* 262a⟩+≡ ◁284c 285b▷

```
let q_desc = "Is demexp usable?" in
assert(Demexp.V1.new_question client (cookie, q_desc)
         = { question_id_return_code = rt_ok; question_id_id = 0} );
assert(Demexp.V1.new_question client (cookie, q_desc)
         = { question_id_return_code = rt_already_exists;
             question_id_id = 0});
assert(Demexp.V1.new_question client (cookie, "")
         = { question_id_return_code = rt_bad_format;
             question_id_id = -1});
assert(Demexp.V1.new_question client (cookie, " a question  spaced  ")
         = { question_id_return_code = rt_ok;
             question_id_id = 1});
let r = Demexp.V1.get_question_id client (cookie, "a question spaced") in
assert(r = { question_id_return_code = rt_ok; question_id_id = 1 });
```

We check that each newly created question has its specific tag associated to it.

285b ⟨*work.ml* 262a⟩+≡ ◁285a 285c▷

```
assert(Demexp.V1.tag_info client (cookie, 3, 2)
         = { tag_info_rc = rt_ok;
             tag_info
               = Array.of_list [{ a_tag_id = 4;
                                  a_tag_timestamp
                                    = Classification.get_tag_timestamp 4;
                                  a_tag_label = "question 1"; };
                                { a_tag_id = 3;
                                  a_tag_timestamp
                                    = Classification.get_tag_timestamp 3;
                                  a_tag_label = "question 0"; };
                               ]});
assert(Demexp.V1.get_question_tags client (cookie, 0(*q_id*))
         = Array.of_list [3;1]);
assert(Demexp.V1.get_question_tags client (cookie, 1(*q_id*))
         = Array.of_list [4;0]);
```

We check the maximum question identifier has been correctly updated.

285c ⟨*work.ml* 262a⟩+≡ ◁285b 285d▷

```
assert(Demexp.V1.max_question_id client cookie
         = { max_question_id_rc = rt_ok; max_question_id = 1; });
```

We get the identifier of above question and we add several responses to it (checking normalization for response "No"). We also check badly formatted responses.

285d ⟨*work.ml* 262a⟩+≡ ◁285c 286a▷

```
let r = Demexp.V1.get_question_id client (cookie, q_desc) in
assert(r = { question_id_return_code = rt_ok; question_id_id = 0 });
let q_id = r.question_id_id in
assert(Demexp.V1.add_response client (cookie, q_id, "Yes", "")
         = rt_ok);
assert(Demexp.V1.add_response client (cookie, q_id, "  No\n \t", "")
         = rt_ok);
assert(Demexp.V1.add_response client (cookie, q_id, "Maybe in a while", "")
         = rt_ok);
assert(Demexp.V1.add_response client (cookie, q_id, "Maybe in a while",
                                      "http://www.demexp.org")
         = rt_already_exists);
assert(Demexp.V1.add_response client (cookie, q_id, "", "")
         = rt_bad_format);
```

We then do a valid vote and check invalid ones.

286a ⟨*work.ml* 262a⟩+≡ ◁285d 286b▷

```
(* valid vote *)
assert(Demexp.V1.vote client (cookie, q_id, Array.of_list [1]) = rt_ok);
(* invalid ones *)
assert(Demexp.V1.vote client (cookie, 42, Array.of_list [1])
         = rt_not_found);
assert(Demexp.V1.vote client (cookie, q_id, Array.of_list [1; 1])
         = rt_duplicate_vote_choice);
assert(Demexp.V1.vote client (cookie, q_id, Array.of_list [1; 42])
         = rt_vote_choice_not_found);
assert(Demexp.V1.vote client (0, q_id, Array.of_list [1])
         = rt_anonymous_cannot_vote);
```

We get our own vote and check error cases.

286b ⟨*work.ml* 262a⟩+≡ ◁286a 287a▷

```
assert(Demexp.V1.get_vote client (cookie, q_id, "root")
         = { get_vote_rc = rt_ok;
             get_vote = Array.of_list [1] });
(* not our vote *)
assert(Demexp.V1.get_vote client (cookie, q_id, "bad login")
         = { get_vote_rc = rt_not_enough_rights;
             get_vote = Array.of_list [] });
(* invalid delegate *)
assert(Demexp.V1.get_vote client (cookie, q_id, "delegate_a")
         = { get_vote_rc = rt_not_enough_rights;
             get_vote = Array.of_list [] });
(* valid and voting delegate *)
assert(Demexp.V1.add_participant client (cookie, "delegate_a", "toto",
                                         Array.of_list [])
         = { add_participant_rc = rt_ok;
             add_participant_id = 2; });
let r = Demexp.V1.login client (Rtypes.int_of_uint4 protocol_version,
                                "delegate_a", "toto") in
assert(r.login_return_code = rt_ok);
let delegate_cookie = r.login_cookie in
assert(Demexp.V1.vote client (delegate_cookie, q_id, Array.of_list [0])
         = rt_ok);
Demexp.V1.goodbye client delegate_cookie;
assert(Demexp.V1.get_vote client (cookie, q_id, "delegate_a")
         = { get_vote_rc = rt_ok;
             get_vote = Array.of_list [0] });
(* invalid question *)
assert(Demexp.V1.get_vote client (cookie, 42, "root")
         = { get_vote_rc = rt_not_found;
             get_vote = Array.of_list [] });
```

We check that the response stored on the server is what we expect. And also that we get an error on an unknown question or if we request to much questions at once.

287a ⟨*work.ml* 262a⟩+≡ ◁286b 287b▷

```
(* question exists *)
let r = Demexp.V1.question_info client (cookie, q_id, 3) in
assert(r.question_info_rc = rt_ok);
assert(Array.length r.question_info = 2);
assert(r.question_info.(1).q_id = q_id);
assert(r.question_info.(1).q_desc = q_desc);
assert(r.question_info.(1).q_info_limit_date = Int64.zero);
assert(r.question_info.(1).q_info_status = tagging_only);
assert(r.question_info.(1).q_info_responses.(0).r_info_desc
        = Posbase.default_rejected_question_response);
assert(r.question_info.(1).q_info_responses.(0).r_info_link = "");
assert(r.question_info.(1).q_info_responses =
       Array.of_list [ { r_info_desc
                           = Posbase.default_rejected_question_response;
                         r_info_link = ""; };
                       { r_info_desc = "Yes";
                         r_info_link = ""; };
                       { r_info_desc = "No";
                         r_info_link = ""; };
                       { r_info_desc = "Maybe in a while";
                         r_info_link = ""; }; ]);
assert(r.question_info.(1).q_info_num_votes = 2);
assert(Array.to_list r.question_info.(1).q_info_elected_responses = [1]);
(* question does not exist *)
let r = Demexp.V1.question_info client (cookie, 42, 1) in
assert(r.question_info_rc = rt_ok);
assert(Array.length r.question_info = 0);
(* request too much questions at once *)
let r = Demexp.V1.question_info client (cookie, 42, 10000) in
assert(r.question_info_rc = rt_request_too_much_ids);
```

We change the question status and check the change is effective. We also check error cases for question status change.

Note: We cannot test the return of `rt_bad_status` by the server by calling `Demexp.V1.set_question_status` because the RPC library check at send time the validity of the `enum question_status_e` field (and returns an exception is that case). So we rather call directly `Work.set_question_status`.

287b ⟨*work.ml* 262a⟩+≡ ◁287a 288a▷

```
assert(Demexp.V1.set_question_status client (cookie, q_id, public)
        = rt_ok);
let r = Demexp.V1.question_info client (cookie, q_id, 1) in
assert(r.question_info.(0).q_info_status = public);
(* error cases *)
assert(Demexp.V1.set_question_status client (cookie, 42, public)
        = rt_not_found);
let zero = Rtypes.int4_of_int 0 in
assert(set_question_status (cookie, q_id, zero) = rt_bad_status);
```

45.4.6 Saving and loading of bases

We check that the bases can be correctly saved and restored from a file.

288a ⟨*work.ml* 262a⟩+≡ ◁287b 288b▷

```
      let saved_participant = Participants.comparable_base () in
      let saved_classification = Classification.comparable_base () in
      let saved_position = Posbase.comparable_base () in
      Io.save_bases_xml ();
      Io.load_bases_xml ();
      assert(saved_participant = Participants.comparable_base ());
      assert(saved_classification = Classification.comparable_base ());
      assert(saved_position = Posbase.comparable_base ());
```

45.4.7 Timestamps

We check that we can get all the timestamps, with their correct value.

288b ⟨*work.ml* 262a⟩+≡ ◁288a 288c▷

```
      let ts = Demexp.V1.get_timestamps client cookie in
      assert(ts.gt_return_code = rt_ok);
      let p_block = Participants.timestamp_list () in
      let p_uncompressed = Timestamp.uncompress ts.gt_participant in
      assert(p_block.{0} = p_uncompressed.{0});
      (* we do not check all participant timestamps because for the
         erased participant, it's timestamp value is unkown *)
      assert(Timestamp.uncompress ts.gt_tag
               = Classification.tag_timestamp_list ());
      assert(Timestamp.uncompress ts.gt_question
               = Posbase.timestamp_list ());
```

We finally shut down the server. We have the right to do that as we are the administrator.

288c ⟨*work.ml* 262a⟩+≡ ◁288b 288d▷

```
      assert(Demexp.V1.stop_server client cookie = rt_ok);
     )
```

45.4.8 Dummy client

With `dummy_client`, we simulate a client that speaks with the server. This "client" connects to the server, exchanges various messages and then closes the connection.

288d ⟨*work.ml* 262a⟩+≡ ◁288c

```
  let dummy_client () =
    if Config.do_autotests then (
      let do_job () =
        (* connect to server *)
        dbg "dummy client: connect to server...";
        let server_addr = Config.default_server_address in
        let server_port = Config.default_server_port in
        let client = Demexp.V1.create_client
            (Rpc_client.Inet (server_addr, server_port))
            Rpc.Tcp in
        dbg "dummy client: connected";
        test_message_sequence client;
        dbg "dummy client: I have finished my job!";
        Rpc_client.shut_down client in
      Unix.handle_unix_error do_job ()
    )
```

Chapter 46

Main server

This module defines the server itself.

46.1 General architecture

289a ⟨*demexp-server.ml* 289a⟩≡ 289b ▷
```
(* copyright 2003-2005 David MENTRE *)
(* this software is under GNU GPL. See COPYING.GPL file for details *)

open Srvflags
open Printf
open UnixLabels
```

46.2 Command line parsing

`usage_msg` contains the usage information for demexp server printed with option `--help` or when an option is not recognized.

289b ⟨*demexp-server.ml* 289a⟩+≡ ◁ 289a 289c ▷
```
let usage_msg = "demexp-server [options]\noptions are:"
```

Regular expression `listen_arg_regex` matches server names with formats "`host:port`", "`host`" or "`:port`".

289c ⟨*demexp-server.ml* 289a⟩+≡ ◁ 289b 289d ▷
```
let listen_arg_regex = Str.regexp "\\([-a-zA-Z.0-9]+\\)?\\(:\\([0-9]+\\)\\)?"
```

Helper function `parse_listen_arg` is called to parse `--listen` option. It simply tries to match the above regex and update `flag_address` and `flag_port` with effectively matched argument.

289d ⟨*demexp-server.ml* 289a⟩+≡ ◁ 289c 290a ▷
```
let parse_listen_arg str =
  if Str.string_match listen_arg_regex str 0 then (
    (try
      flag_address := Str.matched_group 1 str
    with Not_found -> () (* no match *) );
    (try
      flag_port := int_of_string (Str.matched_group 3 str)
    with
    | Not_found -> () (* no match *)
    | Failure "int_of_string" -> raise (Arg.Bad "invalid port argument"))
   ) else
    raise (Arg.Bad "invalid empty argument")
```

The available command line options. Most of them are setting global variables defined in `Srvflags`.

290a ⟨*demexp-server.ml* 289a⟩+≡ ◁289d 290b▷
```
let cmdline_options =
  [
   ("--listen", Arg.String parse_listen_arg,
    "set address on which the server will listen to (host:port)");
   ("--bases", Arg.Set_string flag_bases_name,
    "set filename to use for bases (default: \"bases.dmxp\")");
   ("--logfile", Arg.Set_string flag_log_filename,
    "redirect standard and debug logs to given file (use \"-\" for stdout)");
   ("--daemon", Arg.Set flag_daemon,
    "run as a daemon (i.e. detached from calling process)");
   ("--debug", Arg.Set flag_debug, "print debugging information on stderr");
   ("-d", Arg.Set flag_debug, "idem");
   ("--autotests", Arg.Set flag_autotests,"start the server in autotest mode");
  ]
```

Function `parse_cmdline` parses command line options. It sets global flags defined in `Srvflags`.

290b ⟨*demexp-server.ml* 289a⟩+≡ ◁290a 290c▷
```
let parse_cmdline () =
  Arg.parse cmdline_options (fun _ -> ()) usage_msg;
  if !flag_autotests then printf "Server in autotest mode \n"
```

46.3 Automatic tests

We create a "client" which is in fact a new thread started by `start_dummy_client`, so both client and server can execute simultaneously. Routine `dummy_client` is defined in code chunk 288d.

290c ⟨*demexp-server.ml* 289a⟩+≡ ◁290b 290d▷
```
let start_dummy_client () =
  if Config.do_autotests && !flag_autotests then (
    dbg "server: starting dummy client...";
    Some(Thread.create Work.dummy_client ())
   ) else
    None
```

46.4 Main

Function `save_pid` save the server PID (Process IDentifier) into a file named after bases file name, but with `.pid` suffix.

290d ⟨*demexp-server.ml* 289a⟩+≡ ◁290c 291a▷
```
let save_pid () =
  let filename = (Filename.chop_extension !flag_bases_name) ^ ".pid" in
  let oc = open_out filename in
  output_string oc (string_of_int (Unix.getpid ()));
  output_string oc "\n";
  close_out oc
```

Helper function `setup_server_socket` creates the main socket on which the server will listen to, using parameters defined in `Config` module and updated by parsing command arguments: the server address (`flag_address`) and the port number (`flag_port`) on which the server must listen; and the maximum number of clients it can handle simultaneously (`server_max_clients`).

We set `SO_REUSEADDR` option on this socket so that, if a previously server has used our IP address and port but improperly exited (the socket address is in state `TIME_WAIT`[1]), we can still bind to this address. One of the rationale behind waiting for `TIME_WAIT` timeout is to be sure that no other IP packets for the previous server could be erroneously sent to our new one. For debugging purpose, it is much easier to just avoid this timeout.

291a ⟨*demexp-server.ml* 289a⟩+≡ ◁290d 291b▷

```
let setup_server_socket () =
  log "server: opening main socket (%s:%d)" !flag_address !flag_port;
  let main_socket = socket ~domain:PF_INET ~kind:SOCK_STREAM ~protocol:0 in
  setsockopt main_socket SO_REUSEADDR true;
  try
    let host_entry = gethostbyname !flag_address in
    if Array.length host_entry.h_addr_list > 0 then (
      bind main_socket ~addr:(ADDR_INET(host_entry.h_addr_list.(0),
                                        !flag_port));
      listen main_socket ~max:Config.server_max_clients;
      main_socket
    ) else (
      printf "error: found no entry for address: \"%s\"\n"
        !flag_address;
      exit 1
     )
  with Not_found ->
    printf "error: found no entry for address: \"%s\"\n"
      !flag_address;
    exit 1
```

Helper function `restartable_run` calls `Unixqueue.run` on `unix_queue` but catch all `Cannot_represent` exceptions and upon them restart the `unix_queue`. This way, we intercept all badly formed RPC and avoid further processing of them.

291b ⟨*demexp-server.ml* 289a⟩+≡ ◁291a 292▷

```
let rec restartable_run unix_queue =
  try
    Unixqueue.run unix_queue
  with
    Rtypes.Cannot_represent a ->
      log "warning: bad entry, ignore further input (%s failed)" a;
      restartable_run unix_queue
  | Unix_error (error, fun_name, fun_param) ->
      log "warning: Unix error in function '%s' with parameter '%s': %s"
        fun_name fun_param (error_message error);
      restartable_run unix_queue
```

[1] One can see the state of TCP socket addresses using Unix command "`netstat -t -a`".

Function `setup_signal_handling` setups a signal handler for signals SIGHUP, SIGINT, SIGQUIT and SIGTERM. In that case, the server is scheduled to be halted.

292 ⟨*demexp-server.ml* 289a⟩+≡ ◁291b 293▷

```
let setup_signal_handling esys =
  let signal_handler signame _ =
    log "Signal %s received" signame;
    Work.schedule_server_halt esys in
  Sys.set_signal Sys.sigint (Sys.Signal_handle (signal_handler "SIGINT"));
  Sys.set_signal Sys.sighup (Sys.Signal_handle (signal_handler "SIGHUP"));
  Sys.set_signal Sys.sigquit (Sys.Signal_handle (signal_handler "SIGQUIT"));
  Sys.set_signal Sys.sigterm (Sys.Signal_handle (signal_handler "SIGTERM"))
```

Function `start_server` loads the databases and then start and endless loop, waiting for clients' remote procedure calls. The server is configured to reject incoming messages bigger than `maximum_message_size`.

293 ⟨*demexp-server.ml* 289a⟩+≡ ◁292 294a▷

```
let start_server () =
  Participants.make_empty_participant_base_with_root ();
  log "server: intialize empty participant base";
  Classification.initialize ();
  Io.load_bases_xml ();
  Work.initialize ();
  save_pid ();

  let socket = setup_server_socket () in
  let esys = Unixqueue.create_unix_event_system () in
  let server = Messages_srv.Demexp.V1.create_server
      ~limit:Config.server_max_clients
      ~proc_login:Work.login
      ~proc_goodbye:Work.goodbye
      ~proc_get_timestamps:Work.get_timestamps
      ~proc_new_question:Work.new_question
      ~proc_get_question_id:Work.get_question_id
      ~proc_add_response:Work.add_response
      ~proc_max_question_id:Work.max_question_id
      ~proc_question_info:Work.question_info
      ~proc_set_question_status:Work.set_question_status
      ~proc_vote:Work.vote
      ~proc_get_vote:Work.get_vote
      ~proc_max_participant_id:Work.max_participant_id
      ~proc_participant_info:Work.participant_info
      ~proc_add_participant:Work.add_participant
      ~proc_update_participant:Work.update_participant
      ~proc_remove_participant:Work.remove_participant
      ~proc_max_tag_id:Work.max_tag_id
      ~proc_create_tag:Work.create_tag
      ~proc_tag_info:Work.tag_info
      ~proc_update_tag:Work.update_tag
      ~proc_tag_question:Work.tag_question
      ~proc_untag_question:Work.untag_question
      ~proc_get_question_tags:Work.get_question_tags
      ~proc_tag_set_of_question_group:Work.tag_set_of_question_group
      ~proc_stop_server:(Work.stop_server esys)
      ~proc_server_timers:Work.server_timers
      (Rpc_server.Descriptor socket)
      Rpc.Tcp
      Rpc.Socket
      esys in
  Rpc_server.set_session_filter
    server
    (fun _ ->
      (`Accept_limit_length(Config.maximum_message_size, `Deny)));
  Work.server_descriptor := Some server;
  setup_signal_handling esys;

  let client_thread =
    start_dummy_client () in (* for autotests, empty operation otherwise *)

  log "server: ready";
  restartable_run esys;
```

```
dbg "server: stop";
(match client_thread with (* used in autotests *)
| None -> ()
| Some t ->
    log "server: join with client thread";
    Thread.join t);
log "server: exiting"
```

Function `detach_from_caller` is called when we want to run as a daemon, so as to be independent of calling process. We need to:

- fork() in order to be independ of our caller process;
- create our own process group through setsid() in order to avoid receiving shell control signals.

294a ⟨*demexp-server.ml* 289a⟩+≡ ◁293 294b ▷

```
let detach_from_caller () =
  let pid = fork () in
  if pid <> 0 then (
    (* we are the parent, bye bye *)
    exit 0
  ) else (
    (* we are the child *)
    ignore(setsid ())
   )
```

And now, let's the show begin!

294b ⟨*demexp-server.ml* 289a⟩+≡ ◁294a

```
let _ =
  parse_cmdline ();
  if !flag_log_filename <> default_log_filename then
    redirect_logs_to_file ();
  if !flag_daemon && (!flag_log_filename = "") then (
    log "ERROR: you should use --logfile option with --daemon option";
    exit 2
   );
  if !flag_daemon then
    detach_from_caller ();
  log "demexp server (%s)@\n  @[demexp server comes with@ ABSOLUTELY NO WARRANTY.@ This@ prograa
    Config.server_version;
  start_server ()
```