A Functional Approach to Web Publishing (Second Edition)

April 10, 2007

Martin Elsman

Niels Hallenberg

Carsten Varming

SMLserver

A Functional Approach to Web Publishing (Second Edition)

Martin Elsman (mael@itu.dk)
IT University of Copenhagen, Denmark

Niels Hallenberg (nh@itu.dk)
Statistics Denmark, Denmark

Carsten Varming (varming@itu.dk)
IT University of Copenhagen, Denmark

April 10, 2007

Contents

Preface

The ideas behind the SMLserver project came up in 1999 when the first author was attending a talk by Philip Greenspun, the author of the book "Philip and Alex's Guide to Web Publishing" [Gre99]. Philip and his co-workers had been writing an astonishing 250,000 lines of dynamically typed TCL code to implement a community system that they planned to maintain, extend, and even customize for different Web sites. Although Philip and his co-workers were very successful with their community system, the dynamic typing of TCL makes such a large system difficult to maintain and extend, not to mention customize.

Although the cost of using statically typed programming languages for Web applications is a more tedious development cycle, a static type system may cause many bugs to be found before a Web site is launched and help the programmers in the development process.

The SMLserver project was initiated in the end of 2000 by the construction of an embeddable runtime system and a bytecode backend for the MLKit [TBE+01], an Open Source Standard ML compiler. Once the bytecode backend and the embeddable runtime system, also called the Kit Abstract Machine (KAM), was in place, the KAM was embedded in an AOLserver module[1] in such a way that requests for files ending in `.sml` and `.msp` (also called *scripts*) cause the corresponding compiled bytecode files to be loaded and executed. In April 2001, the basic system was running, but more work was necessary to support caching of loaded code, multi-threaded execution, and many of the other interesting AOLserver features, such as database interoperability.

In 2005, SMLserver was ported to Apache 2.0 including the many great features of AOLserver, including script scheduling, Oracle and MySQL database support, caching support, and much more.

SMLserver has been used in practice for building large intranet systems in the form of a course evaluation system at the IT University of Copenhagen and an online course registration system.

In the following, we assume that the reader is familiar with the programming

[1] AOLserver is a multi-threaded Web server provided by America Online (AOL).

language Standard ML and with functional programming in general. There are several good introductory Standard ML text books available, including [Pau96, HR99]. The present book is not meant to be a complete user's manual for SMLserver. Instead, the book is meant to give a broad overview of the possibilities of using SMLserver for Web application development. The choice of content and the examples presented in the book are inspired from more than six years of experience with developing and teaching the course "Web Publishing with Databases" at the IT University of Copenhagen.

We would like to thank Lars Birkedal, Ken Friis Larsen, and Peter Sestoft for their many helpful comments on the project. Peter developed the concept of ML Server Pages and we are happy that much of the code that Peter wrote for his Moscow ML implementation of ML Server Pages is reused in the SMLserver project (in particular, the `Msp` structure is written entirely by Peter). We would also like to thank Mads Tofte for his continued encouragement on working on this project. Mads is a brilliant programmer and has developed several Web applications with SMLserver, including an alumni systemand a course evaluation system for the IT University of Copenhagen.

SMLserver is Open Source and distributed under the GNU General Public License (GPL). More information about the SMLserver project can be found at the SMLserver Web site:

`http://www.smlserver.org`

Martin Elsman
Niels Hallenberg
Carsten Varming

Copenhagen, Denmark
February, 2002 (Revised January 2003 and January 2007 for Second Edition)

Chapter 1

Introduction

SMLserver is a Web server module for Apache 2.0, the most popular Web server on the Internet. SMLserver comes with a compiler for compiling Web applications written in Standard ML [MTHM97] into bytecode to be interpreted by the SMLserver module. SMLserver has an extensive Application Programmer Interface (API). For instance, SMLserver provides efficient interfaces to several Relational Database Management Systems (RDBMSs), including MySQL, Postgresql, and Oracle.

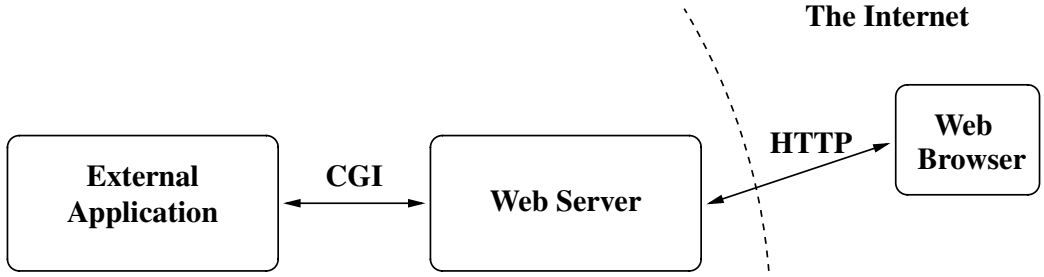
SMLserver extends Apache by providing the possibility of programming dynamic Web pages in Standard ML using its rich language features including datatypes, pattern matching, higher-order functions, parametric polymorphism, and modules. SMLserver provides a Standard ML interface to much of the Apache API, thereby giving the Standard ML programmer access to many great features, including the following:

- Different RDBMSs, including Oracle, Postgresql, and MySQL, may be accessed through a generic database interface.
- SMLserver provides easy access to HTTP header information, including form content and cookie information.
- Efficient caching support makes it possible to decrease the load caused by frequently run database queries, such as the querying of passwords for user authentication.
- SMLserver supports scheduled execution of scripts.
- SMLserver has an advanced interface for Web applications to send emails.
- SMLserver has support for writing type-safe XML-RPC clients and servers in Standard ML.

Besides the above features, SMLserver works well together with other Apache features and other Apache Modules. For instance, SMLserver works well together with Apache's Secure Socket Layer support (SSL), which allows for encrypted communication between the Web server and its clients, and with the Apache Rewrite Module, which allows for trapping (i.e., filtering) of requests for easy access control.

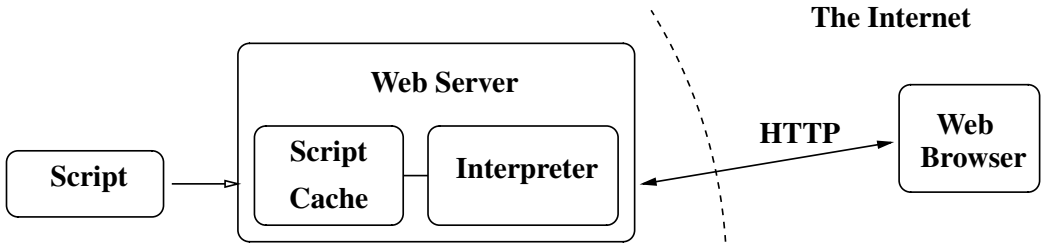
1.1 Web Scripting

The Common Gateway Interface (CGI) is a standard for interfacing external applications with a Web server that communicates with clients using the HyperText Transfer Protocol (HTTP). The situation is pictured as follows:



The external application, which is also called a *CGI program*, may be written in any language that allows the program to be executed on the system. It is even possible to write Standard ML CGI-scripts with your favorite Standard ML compiler using the `Mosmlcgi` library provided with the Moscow ML distribution.

Unfortunately, the traditional CGI approach has a serious drawback: It is slow. Every time a client requests a page from the Web server, the server must fork a new process and load the external application into memory before the application is executed. Moreover, after execution, the operating system must reclaim resources used by the process. One way of increasing availability and to speed up response times is to embed an interpreter within a Web server as the following picture illustrates:



Notice that in this setting, scripts are cached in the Web server, which further increases the efficiency of script execution. This is the approach taken by SMLserver.

1.2 Why Standard ML

Standard ML (SML) is a high-level statically typed functional programming language.

It is a *high-level* programming language in the sense that it uses automatic memory management. In contrast to a low-level programming language, such as C, the programmer need not be concerned with the allocation and deallocation of memory. Standard ML supports many other high-level programming language features as well, including pattern-matching and exceptions. It even has an advanced modules language, which enhances the possibilities of program composition.

In contrast to Web applications built with dynamically typed languages such as TCL, Perl, PHP, or so, systems built with *statically typed* languages are often more reliable and more robust. When a change is made to some part of the program, a static type system enforces (at compile time, that is) the change to integrate well with the entire program; in a dynamically typed setting, no errors are caught this early in the development cycle.

Standard ML is a *functional* language in that it supports *higher-order functions*; that is, functions may take functions as arguments and return functions as a result. Although it is a functional language, Standard ML also has support for imperative features such as mutable data structures like arrays and references.

1.3 Outline

Chapter 2 provides instructions for getting started with SMLserver. Chapter 3 presents two simple examples, which illustrate the basic mechanism for writing dynamic Web pages with SMLserver. Chapter 4 describes how SMLserver Web scripts may use data obtained from users. Chapter 5 describes how state in Web scripts may be emulated using so-called hidden form variables. The concept of regular expressions and the idea of fetching data from foreign Web sites are covered in Chapter 6. The general interface for connecting to a Relational Database Management System (RDBMS) is described in Chapter 7. A mechanism for checking that form variables contain values of the right type is presented in Chapter 8. Chapter 9 presents a user authentication mechanism based on information stored in a database and cookie information provided by the client browser. Finally, Chapter 10 demonstrates SMLserver's support for scheduling of scripts and trapping (i.e., filtering) of requests to for easy access control. A summary is given in Chapter 12.

All concepts are illustrated using a series of examples, which are all included in the SMLserver distribution.

Chapter 2

Getting Started

For installation requirements and instructions, please consult the SMLserver Web site (`http://www.smlserver.org`) or the file `README_SMLSERVER` in the source distribution, which is also available from the SMLserver Web site. In the following we shall assume that SMLserver is installed on your system and that the command

```
$ smlserverc -v
```

salutes you with a greeting from SMLserver.

The remaining sections in this chapter is organised as follows. First, Section 2.1 guides you through compiling your own version of the SMLserver demonstration Web project. Second, Section 2.2 guides you through configuring Apache for use with SMLserver and the compiled Web project.

2.1 Compiling the Demonstration Web Project

Start by creating your own `web` directory and copy the SMLserver demonstration files to this directory:

```
$ cd $HOME
$ mkdir web
$ cp -pa /usr/local/lib/smlserver/{www,web_sys,web_demo_lib} \
   \ web/
```

The `cp` command above assumes that SMLserver is installed in `/usr/local` and not in `/usr`, as it probably would be if SMLserver was installed as a binary package.

Now, change to the `web/www` directory and type `make`:

```
$ cd web/www
$ make
```

Building involves invoking `smlserverc` and `mspcomp` for compiling libraries and script files mentioned in the `mlb`-file `web.mlb`.

2.2 Configuring and Starting Apache

You are now ready to configure the Apache Web server. Before starting, we note that different distributions of Apache (for different OS distributions) use different strategies for configuration. Here we show the details of configuring Apache for SMLserver on a Linux Debian System.

Follow the following steps:

1. Edit the file `/etc/apache2/apache2.conf` by adding to `DirectoryIndex` the list `index.sml`.

2. Edit the file `/etc/apache2/sites-available/default` by (A) changing `DocumentRoot` to `/home/`*user*`/web/www` and (B) changing `Directory path` to `/home/`*user*`/web/www`, where *user* is your login name.

3. Create the file `/etc/apache2/mods-available/sml.conf` with the following content:

   ```
   <IfModule mod_sml.c>
     AddHandler sml-module .sml
     SmlPrjId "web"
     SmlPath "/home/user/web/www/"
     SmlInitScript "/home/user/web/www/../web_sys/init.sml"
   </IfModule>
   ```

   Again, substitute your own login name for `user` in the content.

4. Execute the following commands:

   ```
   $ cd /etc/apache2/mods-enable/
   $ sudo ln -s /etc/apache2/mods-available/sml.conf .
   $ sudo ln -s /etc/apache2/mods-available/sml.load .
   ```

You are now ready to (re)launch Apache with the following command:

```
$ sudo apache2ctl restart
```

Point your browser to `http://localhost/web/` and enjoy. By executing the command

```
$ ps --cols=200 guax | grep nsd
```

you should see that Apache is running five or six processes. Apache writes information into the file `/var/log/apache2/error.log`. By looking at the log, you should see a notice that Apache has loaded the SMLserver module.

2.3 Interfacing to an RDBMS

To get access to an RDBMS from within your SMLserver scripts, an RDBMS supported by SMLserver must be installed on your system. Supported RDBMS's include MySQL, Oracle, and Postgresql (`http://www.postgresql.org`). The MySQL and Postgresql databases are supported through unixODBC.[1].

Information on how to use SMLserver with Oracle and MySQL is available from the SMLserver home page. The next section describes how to interface to the Open Source RDBMS Postgresql.

2.4 Interfacing to Postgresql

This section describes how to set up a database with Postgresql for the purpose of using it with SMLserver. We assume that Postgresql ($\geq$ 8.1) is already installed on the system. We also assume that SMLserver is configured with `unixODBC` support (`--enable-odbc`) before compilation and installation; see the file `README_SMLSERVER` for details.

Follow the steps below:

1. (Re)start the Postgresql daemon process by executing the following command:

   ```
   $ sudo /etc/init.d/postgresql-8.1 restart
   ```

2. Create a database user with the same name as your user name on the Linux box:

   ```
   $ sudo su - postgres
   $ createuser -P user
   ```

   Invent a new password for the database user. Answer yes to the questions asked by `createuser`.

[1] `http://www.unixodbc.org`

3. As `user`, create a database (also called `user`) as follows:

   ```
   $ createdb $USER
   ```

   You can now use the command `psql` to control your database and submit SQL queries and commands to your database. Install the data models for the demonstration programs by executing the commands

   ```
   $ cd $HOME/web/web_demo_lib/pgsql
   $ psql -c "\i all.sql"
   ```

   Verify that the database is available using `isql`:

   ```
   $ isql psql
   > select * from guest;
   ```

   (You should see a message from Homer Simpson...)

4. Restart Apache by executing the command

   ```
   $ sudo /etc/init.d/apache2 restart
   ```

5. After copying the sample Web directory to `$HOME`, as described in Section 2.1, edit the file `$HOME/web/web_demo_lib/Db.sml`. Make sure that the structure `DbBackend` passed to the `DbFunctor` is the structure `Web.DbPgBackend`. The lines defining the Oracle structure and the MySQL structure should be commented out.

6. Edit the file `$HOME/web/web_sys/init.sml`. Enable the Postgresql configuration lines.

7. Compile the sample Web project as described in Section 2.1.

8. Go start your Web browser and visit the database examples available from `http://localhost/web/index.sml`.

2.5 So You Want to Write Your Own Project

To write your own project, create a new `mlb`-file and make this project the project served by SMLserver by updating the Apache configuration file. You can have only one project associated with each Web server that you run. Use the compiler `smlserverc` to compile the library and script-files mentioned in the `mlb`-file into

bytecode. Once your project is compiled, the Web server answers requests of the files listed in the `scripts...end` parts of your `mlb`-file (see Section 3.5).

Library code to be shared between scripts may be stored anywhere on the system and mentioned, for example, in the local part in the project file—look in the sample `mlb`-file `web.mlb` for examples.

Chapter 3

Presenting Pages to Users

In this chapter we show two examples of dynamic Web pages (also called Web scripts) written with SMLserver. The first example, which shows the time of day, takes the form of a regular Standard ML program. It uses the function `Web.Conn.return` to return the appropriate HTML code to the user requesting the page.

The second example, which shows a simple multiplication table, uses the possibility of writing ML Server Pages (MSP) with SMLserver.

3.1 The HyperText Transfer Protocol

Before we dive into the details of particular dynamic Web pages, we briefly describe the protocol that is the basis for the World Wide Web, namely the HyperText Transfer Protocol (HTTP). It is this protocol, which dictates how Web browsers (such as Microsoft's Internet Explorer or Netscape Navigator) make requests to Web servers and how a Web server communicates a response back to the particular browser.

HTTP is a text-based protocol. When a Uniform Resource Locator (URL), such as `http://www.amazon.com`, is entered into a Web browser's location field, the browser converts the user's request into a HTTP `GET` *request*. Web browsers usually request Web pages with *method* `GET`. When a user follows a link from a Web page or when a user submits a form with no method specified, the request is a `GET` request. Another often used request method is `POST`, which supports an unlimited number of form variables with form data of non-restricted size. Other possible methods include `DELETE` and `PUT`. When writing SMLserver applications, however, you need not know about methods other than `GET` and `POST`.

As an example of HTTP in action, consider the case where a user enters the URL `http://www.google.com/search?q=SMLserver` into the location field of a

Web browser. The URL specifies a form variable `q` (read: query) with associated form data `SMLserver`. As a result, the Web browser sends the following `GET` request to port 80 on the machine `www.google.com`:

```
GET /search?q=SMLserver HTTP/1.1
```

The machine `www.google.com` may answer the request by sending the following HTTP *response* back to the client—the HTML content between `<html>` and `</html>` is left out:

```
HTTP/1.1 200 OK
Date: Mon, 23 Jul 2001 11:43:32 GMT
Server: GWS/1.11
Set-Cookie: PREF=ID=49cdd72654784880:TM=995888612:LM=995888612;
            domain=.google.com;
            path=/;
            expires=Sun, 17-Jan-2038 19:14:07 GMT
Content-Type: text/html
Transfer-Encoding: chunked

54d
<html>
  ...
</html>
```

The HTTP response is divided into a status line followed by a series of response header lines and some content. Each *response header* takes the form *key*: *value*, where *key* is a response header key and *value* is the associated response header value. The *status line* specifies that the HTTP protocol in use is version 1.1 and that the *status code* for the request is 200, which says that some content follows after the response headers. Figure 3.1 lists the most commonly used status codes and Figure 3.2 lists some commonly used response headers.[1]

We have more to say about HTTP requests in Chapter 4 where we show how information typed into HTML forms turns into form data submitted with the HTTP request.

3.2 Time of day

We shall now see how to create a small Web service for presenting the time-of-day to a user. The example uses the `Time.now` function from the Standard ML Basis

[1] HTTP 1.1 supported status codes and response headers are listed in RFC 2616. See `http://www.ietf.org`.

Status Code	Description
200 (OK)	Indicates that everything is fine. The document follows the response headers.
301 (Moved Permanently)	The requested document has moved and the URL for the new location is in the `Location` response header. Because the document is moved permanently, the browser may update bookmarks accordingly.
302 (Found)	The requested document has moved temporarily. This status code is very useful because it makes a client request the URL in the `Location` header automatically.
400 (Bad Request)	Bad syntax in the client request.
401 (Unauthorized)	The client tries to access a password protected page without specifying proper information in the `Authorization` header.
404 (Not Found)	The "no such page" response.
405 (Method Not Allowed)	Request method is not allowed.
500 (Internal Server Error)	The "server is buggy" response.
503 (Service Unavailable)	Server is being maintained or is overloaded.

Figure 3.1: The most commonly used HTTP status codes

Header	Description
`Allow`	Specifies the request methods (`GET`, `POST`, etc.) that a server allows. Required for responses with status code 405 (`Method Not Allowed`).
`Cache-Control`	Tells client what caching strategy may be used. Usable values include: `public`: document may be cached `private`: document may be cached by user `no-cache`: document should not be cached `no-store`: document should not be cached and not stored on disk
`Content-Encoding`	May be used for compressing documents (e.g., with `gzip`).
`Content-Language`	Specifies the document language such as `en-us` and `da`. See RFC 1766 for details.[2]
`Content-Length`	Specifies the number of bytes in the document. A persistent HTTP connection is used only if this header is present.
`Content-Type`	Specifies the MIME (Multipurpose Internet Mail Extension) type for the document. Examples include `text/html` and `image/png`.
`Date`	Specifies the current date (Greenwich Mean Time).
`Expires`	Specifies when content should be considered out-of-date.
`Last-Modified`	Indicates the last change of the document.
`Location`	All responses with a status code in the range 300–399 should contain this header.
`Refresh`	Indicates an interval (in seconds) at end of which the browser should automatically request the page again.
`Set-Cookie`	Specifies a cookie associated with the page. Multiple `Set-Cookie` headers may appear.

Figure 3.2: Some commonly used response headers

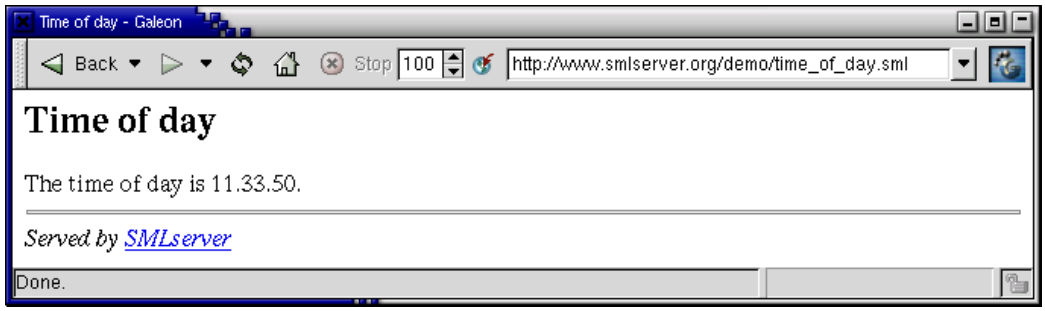


Figure 3.3: The result of requesting the file `time_of_day.sml` using the Netscape browser. The HTTP request causes the compiled `time_of_day.sml` program to be executed on the Web server and the response is sent (via the HTTP protocol) to the Web browser.

Library to obtain the present time of day. HTML code to send to the users browser is constructed using Standard ML string primitives. If you are new to HTML, a short reference is provided in Appendix A on page 115.

```
val time_of_day =
  Date.fmt "%H.%M.%S" (Date.fromTimeLocal(Time.now()))

val _ = Web.Conn.return
  ("<html> \
   \  <head><title>Time of day</title></head> \
   \  <body bgcolor=white> \
   \    <h2>Time of day</h2> \
   \       The time of day is " ^ time_of_day ^ ". \
   \    <hr> <i>Served by \
   \    <a href=http://www.smlserver.org>SMLserver</a> \
   \    </i> \
   \  </body> \
   \</html>")
```

Figure 3.3 shows the result of a user requesting the file `time_of_day.sml` from the Web server.

The example uses the `Web` structure, which gives access to the Web server API; to get an overview of what functions are available in the `Web` structure, consult Appendix B, which lists the Standard ML signature for the structure. The function `Web.Conn.return` takes a string as argument and sends an HTTP response with

status code 200 (Found) and content-type `text/html` to the browser along with HTML code passed in the argument string.

In Section 4.2 on page 25 we show how support for quotations may be used to embed HTML code in Standard ML Web applications somewhat more elegantly than using Standard ML string literals.

In the next section we explore SMLserver's support for ML Server Pages (MSP).

3.3 A Multiplication Table

SMLserver supports the execution of dynamic Web pages written using ML Server Pages (MSP). In this section we show how a dynamic Web page for displaying a multiplication table is written as an ML Server Page. ML Server Pages are stored in files with extension `.msp`. The way in which SMLserver supports `msp`-files is by providing a tool `mspcomp` for compiling an `msp`-file into an `sml`-file, which can then be compiled using `smlserverc` and served by SMLserver. Section 3.5 on page 18 has more to say about projects and how `msp`-files are compiled.

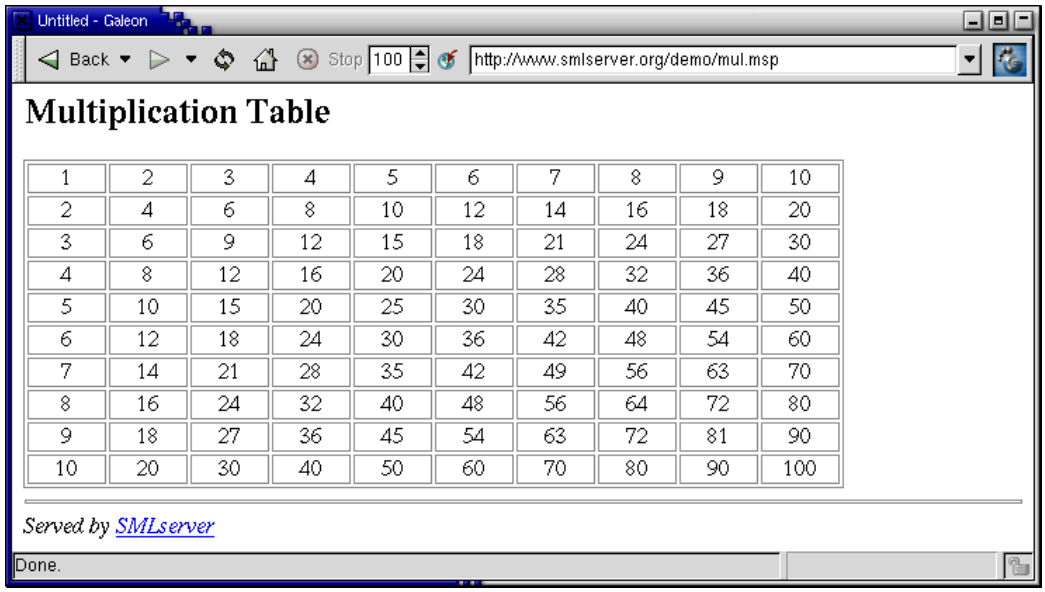
Here is how the ML Server Page for displaying a multiplication table looks like:[3]

```
<?MSP
 local open Msp  infix &&

       fun iter f n = if n = 0 then $""
                      else iter f (n-1) && f n
       fun col r c =
         $"<td width=5% align=center>"
         && $(Int.toString (r * c))
         && $"</td>"
       fun row sz r = $"<tr>" && iter (col r) sz && $"</tr>"
 in
       fun tab sz = iter (row sz) sz
 end
?>

<html>
  <body bgcolor=white>
  <h2>Multiplication Table</h2>
  <table border=1> <?MSP$ tab 10 ?> </table>
```

[3] File `smlserver_demo/www/web/mul.msp`.



1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Figure 3.4: The result of requesting the file `mul.msp` using the Netscape browser. The HTTP request causes the compiled `mul.msp` program to be executed on the Web server and the response is sent (via the HTTP protocol) to the Web browser.

```
    <hr><i>Served by <a
      href=http://www.smlserver.org>SMLserver</a></i>
    </body>
  </html>
```

Figure 3.4 shows the result of a user requesting the file `mul.msp` from the Web server. An `msp`-file contains HTML code with the possibility of embedding Standard ML code into the file, using tags `<?MSP ... ?>` and `<?MSP$ ... ?>`. The former type of tag makes it possible to embed Standard ML declarations into the HTML code whereas the latter type of tag makes it possible to embed Standard ML expressions into the HTML code. The `Msp` structure, which the `msp`-file makes use of, provides functionality for constructing and concatenating HTML code efficiently, by means of constructors `$` and `&&`, respectively. The functions `col`, `row`, and `tab` construct the HTML multiplication table. The functions use the function `iter`, which constructs HTML code by concatenating the results of repeatedly applying the anonymous function given as the first argument; the second argument controls the number of times the anonymous function is called.

3.4 How SMLserver Serves Pages

Before we proceed with more examples of SMLserver Web applications, we describe how SMLserver Web applications are compiled and loaded and, finally, how SMLserver *scripts* (i.e., `sml`-files) are executed when requested by a client.

Apache supports dynamic loading of modules when the server is started. Modules that may be loaded in this way include drivers for a variety of database vendors, a module that enables support for CGI scripts, and a module that enables encryption support, using Secure Socket Layer (SSL). Which modules are loaded when Apache starts is configurable.

SMLserver is implemented as a module `mod_sml.so`, which is loaded into Apache—along with other modules—when Apache starts. When the `mod_sml.so` module is loaded into Apache, future requests for files with extension `.sml` are served by interpreting the bytecode file that is the result of compiling the requested `sml`-file. Compilation of `sml`-files into bytecode files is done by explicitly invoking the SMLserver compiler `smlserverc`.

ML Server Pages, (i.e., scripts with extension `.msp`) may be compiled into scripts with extension `.sml` using the `mspcomp` program that comes with SMLserver.

3.5 ML Basis Files

The SMLserver compiler `smlserverc` takes as argument an `mlb`-file, which lists the `sml`-scripts that a client may request along with Standard ML library code to be used by the client-accessible `sml`-scripts. The result of the compilation is stored within `MLB` directories relative to the individual source files. The `MLB` directories contain information that allows efficient recompilation of `mlb`-files upon change of source code.

Be aware that the project file name must correspond to the string associated with the entry `prjid` in the Apache configuration file, which by default is `sources.mlb`.

An example ML Basis file is listed in Figure 3.5. The `mlb`-file specifies that the two scripts `time_of_day.sml` and `mul.msp.sml` are made available for clients by SMLserver. Assuming the project file name corresponds to the file name mentioned in the Apache configuration file, upon successful compilation of the project, a user may request the files `time_of_day.sml` and `mul.msp.sml`.

The two example scripts `time_of_day.sml` and `mul.msp.sml` (i.e, `mul.msp` compiled with `mspcomp`) may refer to identifiers declared in the files mentioned in the *local-part* of the project file (i.e., between the keywords `local` and `in`), includ-

```
local
  $(SML_LIB)/basis/basis.mlb
  $(SML_LIB)/basis/web/lib.mlb
  ../web_demo_lib/Page.sml
  ../web_demo_lib/FormVar.sml
in
  (* Script files; may refer to identifiers declared in
   * library files, but cannot refer to identifiers
   * in other script files. *)
  scripts
    time_of_day.sml
    mul.msp.sml
  end
end
```

Figure 3.5: An ML Basis file for the two examples in this chapter.

ing identifiers declared by the Standard ML Basis Library[4] and the Web library.[5] In the local-part of the `mlb`-file, it is allowed for an `sml`-file to refer to identifiers declared by previously mentioned `sml`-files or `mlb`-files. However, an `sml`-file mentioned in the `scripts...end` part of an `mlb`-file may not refer to identifiers declared by other files in `scripts...end` parts of the `mlb`-file. Thus, in the example `mlb`-file, `mul.msp.sml` may not refer to identifiers declared in `time_of_day.sml`.

3.6 Compilation

As mentioned, a project is compiled with the SMLserver compiler `smlserverc` with the name of the project file (`sources.mlb` is the default name to use) given as argument:

```
smlserverc sources.mlb
```

The bytecode files (and other information) resulting from compilation of a source file are stored in a subdirectory named `MLB`, located in the same directory as the source file. To work efficiently with SMLserver, you need not know anything about the content of the `MLB` directories. In particular, you should not alter the content

[4] To see what parts of the Standard ML Basis Library that SMLserver supports, consult the file `/usr/local/lib/smlserver/basis/basis.mlb` on your system.

[5] See the file `/usr/local/lib/smlserver/basis/web/lib.mlb`.

of these directories, although it is safe to remove `MLB` directories, if you want to force a recompile of an entire ML Basis File.

To compile an `msp`-file `file.msp` into a file `file.msp.sml`, simply execute the command:

```
$ mspcomp file.msp
```

3.7 Loading and Serving Pages

The first time SMLserver serves an `sml`-file, SMLserver loads the bytecode for the Standard ML Basis Library along with user libraries mentioned in the `mlb`-file before the bytecode for the `sml`-file is loaded. Upon subsequent requests for an `sml`-file, SMLserver reuses the bytecode already loaded.

After bytecode for a request is loaded, SMLserver executes initialization code for each library file before the bytecode associated with the request is executed. Because SMLserver initiates execution in an empty heap each time a request is served, it is not possible to maintain state implicitly in Web applications using Standard ML references or arrays. Instead, state must be maintained explicitly using a Relational Database Management System (RDBMS) or the cache primitives supported by SMLserver (see the `WEB_CACHE` signature in Appendix B). Another possibility is to emulate state behavior by capturing state in form variables or cookies.

At first, this limitation may seem like a major drawback. However, the limitation has several important advantages:

- Good memory reuse. When a request has been served, memory used for serving the request may be reused for serving other requests.

- Support for a threaded execution model. Requests may be served simultaneously by interpreters running in different threads without the need for maintaining complex locks.

- Good scalability properties. For high volume Web sites, the serving of requests may be distributed to several different machines that communicate with a single database server. Making the RDBMS deal with the many simultaneous requests from multiple clients is exactly what an RDBMS is good at.

- Good durability properties. Upon Web server and hardware failures, data stored in Web server memory is lost, whereas, data stored in an RDBMS may be restored using the durability features of the RDBMS.

We have more to say about emulating state using form variables in Chapter 5. Programming with cookies is covered in Chapter 9.

3.8 Logging Messages, Warnings, and Errors

When Apache starts (see Chapter 2), initialization information is written to a *server log file*. The location and name of the server log file is configurable but for many setups, the log file is `/var/log/apache2/error.log`.

In addition to initialization information being written to the server log file, the SMLserver module and other Apache modules may also write information to the server log file when Apache is running. It is also possible for your SMLserver scripts to write messages to the server log file using the function `Web.log`. The function `Web.log` has type `Web.LogSeverity * string -> unit`. The structure `Web` declares the following values of the type `Web.LogSeverity`:

Value	Description (intended use)
`Notice`	Something interesting occurred.
`Info`	Something interesting occurred.
`Warning`	Maybe something bad occurred.
`Error`	Something bad occurred.
`Emergency`	Something extremely bad occurred. The server will shut down after logging this message.
`Critical`	Something extremely bad occurred. The server will shut down after logging this message.
`Alert`	Something extremely bad occurred. The server will shut down after logging this message.
`Debug`	If the server is in *Debug mode*, specified by a flag in the configuration file, the message is printed. If the server is not in debug mode, the message is not printed.

Allowing SMLserver scripts to write messages to the server log file turns out to be handy for debugging scripts.

3.9 Uncaught Exceptions and Aborting Execution

We still have to explain what happens when a script raises an exception that is not handled (i.e., caught) by the script itself. SMLserver deals with such uncaught exceptions by writing a warning in the server log file explaining what exception is raised by what file:

```
[Fri Dec 02 17:00:32 2005] [warn]
```

```
/home/varming/apache2worker/htdocs/web/www/web/
upload/MLB/SMLserver/upload_form.sml.uo
raised Fail
```

There is one exception to this scheme. If the exception raised is the predefined top-level exception `Interrupt`, no warning is written to the server log file. In this way, raising the `Interrupt` exception may be used to silently terminate the execution of a script, perhaps after serving the client an error page. The function `Web.exit`, which has type `unit -> ty`, for any type `ty`, exits by raising the exception `Interrupt`.

An important aspect of using the function `Web.exit` to abort execution of a script is that, with the use of exception handlers, resources such as database connections (see Chapter 7) may be freed appropriately upon exiting.

It is important that SMLserver scripts do not abort execution by calling the function `OS.Process.exit` provided in the Standard ML Basis Library. The reason is that the function `OS.Process.exit` has the unfortunate effect of terminating the Web server main process.[6]

3.10 Accessing Setup Information

The structure `Web.Info` provides an interface to accessing information about the Apache setup, including the possibility of accessing the Web server configuration file settings. Consult Appendix B to see the signature of the `Web.Info` structure.

[6] Recall that each script executes in a separate thread.

Chapter 4

Obtaining Data from Users

One of the fundamental reasons for the success of dynamic Web applications is that Web applications can depend on user input. In this chapter we present two small examples of SMLserver applications that query data from users.

The two examples that we present are both based on two files, an HTML file for presenting a form to the user and an `sml`-file that accesses the submitted data and computes—and returns to the user—HTML code based on the user input. HTML forms provide for many different input types, including text fields, selection boxes, radio buttons, and drop-down menus. If you are new to HTML forms, a quick reference is provided in Appendix A.2 on page 118.

4.1 Temperature Conversion

This section presents a Web application for converting temperatures in degrees Celsius to temperatures in degrees Fahrenheit. The Web application is made up of one file `temp.html` containing an HTML form for querying a temperature from the user and a script `temp.sml` for calculating the temperature in degrees Fahrenheit based on the temperature in degrees Celsius.

The Temperature Form

The file `temp.html` reads as follows:[1]

```
<html>
 <body bgcolor=white>
 <h2>Temperature Conversion</h2>
 Enter a temperature in degrees Celcius:
```

[1] File `smlserver_demo/www/web/temp.html`.

Figure 4.1: The result of displaying the file `temp.html` using the Netscape browser.

```
 <form method=get action=temp.sml>
  <input type=text name=temp_c>
  <input type=submit value="Compute Fahrenheit Temperature">
 </form> <hr><i>Served by <a
     href=http://www.smlserver.org>SMLserver</a>
 </i></body>
</html>
```

The result of displaying the above HTML code in a Web browser is shown in Figure 4.1. The action of the HTML form is the script `temp.sml`. When the user of the HTML form enters a temperature in the text field (20 say) and hits the "Compute Temperature in Fahrenheit" button, the script `temp.sml` is requested from the Web server with the form data `temp_c = 20`.

Calculating the Temperature in Degrees Fahrenheit

Here is the script `temp.sml`:[2]

```
fun calculate c = concat
  ["<html> <body bgcolor=white> ",
   "<h2>Temperature Conversion</h2> ",
   Int.toString c, " degrees Celcius equals ",
   Int.toString (9 * c div 5 + 32),
   " degrees Fahrenheit. <p> Go ",
   "<a href=temp.html>calculate a new temperature</a>.",
```

[2]File `smlserver_demo/www/web/temp.sml`.

```
    "<hr> <i>Served by <a href=http://www.smlserver.org>",
    "SMLserver</a></i> </body></html>"]

val _ = Web.Conn.return
  (case FormVar.wrapOpt FormVar.getIntErr "temp_c"
     of NONE => "Go back and enter an integer!"
      | SOME i => calculate i)
```

The structure `FormVar` provides an interface for accessing form variables of different types.[3]

The expression `FormVar.wrapOpt FormVar.getIntErr` results in a function, which has type `string -> int option`. The function takes the name of a form variable as argument and returns `SOME(`i`)`, where i is an integer obtained from the string value associated with the form variable. If the form variable does not occur in the query data, is not a well-formed integer, or its value does not fit in 32 bits, the function returns `NONE`. We have more to say about the `FormVar` structure in Chapter 8.

In the case that the form variable `temp_c` is associated with a well-formed integer that fits in 32 bits, an HTML page is constructed, which presents the submitted temperature in degrees Celsius, a calculated temperature in degrees Fahrenheit, and a link back to the `temp.html` form. The result of a user converting a temperature in degrees Celsius to a temperature in degrees Fahrenheit is shown in Figure 4.2.

4.2 Quotations for HTML Embedding

As we have seen in the previous example, embedding HTML code in Standard ML programs using strings does not look nice; many characters must be escaped and splitting of a string across lines takes several additional characters per line. This limitation of Standard ML strings makes it difficult to read and maintain HTML code embedded in Standard ML Web applications.

Fortunately, many Standard ML implementations support *quotations*, which makes for an elegant way of embedding another language within a Standard ML program. Here is a small quotation example that demonstrates the basics of quotations:

```
val text = "love"
val ulist : string frag list =
  `<ul>
```

[3] File `smlserver_demo/web_demo_lib/FormVar.sml`.

Figure 4.2: The result of a user converting a temperature in degrees Celsius to a temperature in degrees Fahrenheit.

```
    <li> I ^text Web programming
  </ul>‘
```

The program declares a variable `text` of type `string`, a variable `ulist` of type `string frag list`, and indirectly makes use of the constructors of this predeclared datatype:

```
datatype 'a frag = QUOTE of string
                 | ANTIQUOTE of 'a
```

What happens is that the quotation bound to `ulist` evaluates to the list:

```
[QUOTE "<ul>\n  <li> I ",
 ANTIQUOTE "love",
 QUOTE " Web programming\n</ul>"]
```

Using the `Quot.flatten` function, which has type `string frag list -> string`, the value bound to `ulist` may be turned into a string (which can then be sent to a browser.)

To be precise, a quotation is a particular kind of expression that consists of a non-empty sequence of (possibly empty) fragments surrounded by back-quotes:

exp	::=	‘*frags*‘	quotation
frags	::=	*charseq*	character sequence
	|	*charseq* `^`*id frags*	anti-quotation variable
	|	*charseq* `^(`*exp*`)` *frags*	anti-quotation expression

A *character sequence*, written *charseq*, is a possibly empty sequence of printable characters or spaces or tabs or newlines, with the exception that the characters `^` and ` must be escaped using the notation `^^` and `^``, respectively.

A quotation evaluates to a value of type `ty frag list`, where `ty` is the type of all anti-quotation variables and anti-quotation expressions in the quotation. A character sequence fragment *charseq* evaluates to `QUOTE "`*charseq*`"`. An anti-quotation fragment `^`*id* or `^(`*exp*`)` evaluates to `ANTIQUOTE` *value*, where *value* is the value of the variable *id* or the expression *exp*, respectively.

Quotations are used extensively in the sections and chapters that follow. In fact, to ease programming with quotations, the type constructor `quot` is declared at top-level as an abbreviation for the type `string frag list`. Moreover, the symbolic identifier `^^` is declared as an infix identifier with type `quot * quot -> quot` and associativity similar to `@`. More operations on quotations are available in the `Quot` structure.[4]

4.3 A Dynamic Recipe

This section provides another example of using quotations to embed HTML code in your Standard ML Web applications. Similarly to the temperature conversion example, this example is made up by two files, a file `recipe.html` that provides the user with a form for entering the number of persons to serve apple pie and a script `recipe.sml` that computes the ingredients and serves a recipe to the user.

The Recipe Form

The file `recipe.html` contains the following HTML code:[5]

```
<html>
 <body bgcolor=white>
  <h2>Dynamic Recipe: Apple Pie</h2>
  Enter the number of people you're inviting for apple pie:
  <form method=post action=recipe.sml>
  <input type=text name=persons>
  <input type=submit value="Compute Recipe">
  </form> <hr> <i>Served by
      <a href=http://www.smlserver.org>SMLserver</a></i>
 </body>
</html>
```

[4] File `/usr/share/smlserver/basis/web/Quot.sml` lists the signature for the `Quot` structure.
[5] File `smlserver_demo/www/web/recipe.html`.

Figure 4.3: The result of requesting the file `recipe.html` using the Netscape browser.

The result of requesting the page `recipe.html` using Netscape Navigator is shown in Figure 4.3.

Computing the Recipe

The script `recipe.sml`, which computes the apple pie recipe and returns a page to the user reads as follows:[6]

```
fun error s =
  (Page.return ("Error: " ^ s)
   ‘An error occurred while generating a recipe for
    you; use your browser’s back-button to backup
    and enter a number in the form.‘
   ; Web.exit())

val persons =
  case FormVar.wrapOpt FormVar.getNatErr "persons"
    of SOME n => real n
     | NONE => error "You must type a number"

fun pr_num s r =
  if Real.== (r,1.0) then "one " ^ s
  else
    if Real.==(real(round r),r) then
```

[6]File `smlserver_demo/www/web/recipe.sml`.

```
      Int.toString (round r) ^ " " ^ s ^ "s"
    else Real.toString r ^ " " ^ s ^ "s"

val _ = Page.return "Apple Pie Recipe"
  `To make an Apple pie for ^(pr_num "person" persons), you
   need the following ingredients:
   <ul>
    <img align=right src=applepie.jpg>
    <li> ^(pr_num "cup" (persons / 16.0)) butter
    <li> ^(pr_num "cup" (persons / 4.0)) sugar
    <li> ^(pr_num "egg" (persons / 4.0))
    <li> ^(pr_num "teaspoon" (persons / 16.0)) salt
    <li> ^(pr_num "teaspoon" (persons / 4.0)) cinnamon
    <li> ^(pr_num "teaspoon" (persons / 4.0)) baking soda
    <li> ^(pr_num "cup" (persons / 4.0)) flour
    <li> ^(pr_num "cup" (2.5 * persons / 4.0)) diced apples
    <li> ^(pr_num "teaspoon" (persons / 4.0)) vanilla
    <li> ^(pr_num "tablespoon" (persons / 2.0)) hot water
   </ul>

   Combine ingredients in order given. Bake in greased 9-inch
   pie pans for 45 minutes at 350F. Serve warm with whipped
   cream or ice cream. <p>

   Make <a href=recipe.html>another recipe</a>.`
```

When a user enters a number (say 4) in the form shown in Figure 4.3 and hits the button "Compute Recipe", a recipe is computed by the `recipe.sml` program and HTML code is sent to the user's browser, which layouts the HTML code as shown in Figure 4.4. The expression `FormVar.wrapOpt FormVar.getNatErr` results in a function with type `string -> int option`. This function takes the name of a form variable as argument and returns `SOME(`n`)`, if a representable natural number n is associated with the form variable. If on the other hand the form variable does not occur in the query data or the value associated with the form variable is not a well-formed integer greater than or equal to zero or the integer does not fit in 32 bits, the function returns `NONE`.

Besides the `FormVar` structure, the recipe program also makes use of a library function `Page.return`, which takes a heading and a page body as argument and returns a page to the client:[7]

[7] File `smlserver_demo/web_demo_lib/Page.sml`.

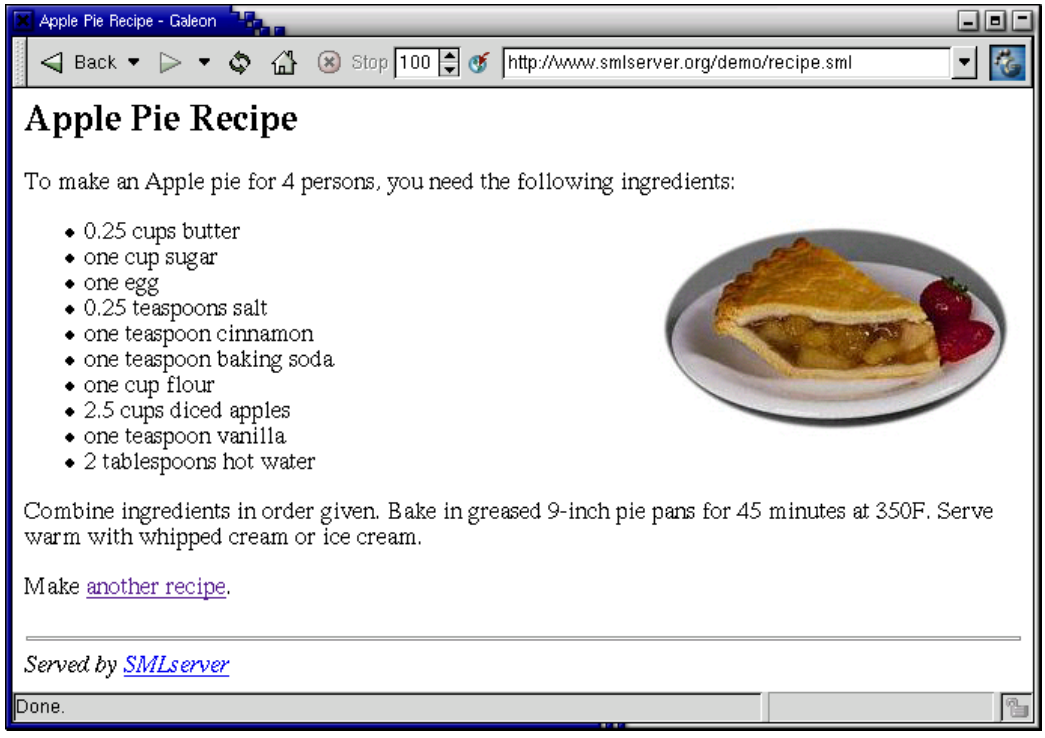


Figure 4.4: The result of computing a recipe for a four-person apple pie.

```
fun return head body = Web.return
  (`<html>
    <head><title>^head</title>
    </head>
    <body bgcolor=white>
     <h2>^head</h2> ` ^^
       body ^^
       `<hr><i>Served by
        <a href=http://www.smlserver.org>SMLserver</a></i>
    </body>
    </html>`)
```

Chapter 5

Emulating State Using Hidden Form Variables

We have mentioned earlier how state in SMLserver Web applications may be implemented using a Relational Database Management System. In Chapter 7, we shall follow this idea thoroughly. In this chapter, on the other hand, we present some examples that show how state in Web applications may be emulated using so called "hidden form variables". The main idea is that no state is maintained by the Web server itself; instead, all the state information is sent back and forth between the client and the Web server for each request and response.

The first example we present implements a simple counter with buttons for counting up and down. The second example implements the "Guess a Number" game.

5.1 Counting Up and Down

The implementation of the counter consists of one `sml`-file named `counter.sml`, which uses the `FormVar` functionality (described on page 25 in Section 4.1) to get access to the form variable `counter`, if present. If the form variable `counter` is not present, a value of 0 (zero) is used for the value of `counter`. The implementation also makes use of the function `Web.Conn.formvar` on which the `FormVar` structure is built (see Section 8.3 on page 86). The script `counter.sml` takes the following form:[1]

```
val counter = Int.toString
  (case FormVar.wrapOpt FormVar.getIntErr "counter"
```

[1] File `smlserver_demo/www/web/counter.sml`.

Figure 5.1: The counter rendered by Netscape Navigator after a few clicks on the "Up" button.

```
      of SOME c => (case Web.Conn.formvar "button"
                     of SOME "Up" => c + 1
                      | SOME "Down" => c - 1
                      | _ => c)
       | NONE => 0)

val _ = Page.return ("Count: " ^ counter)
  `<form action=counter.sml>
     <input type=hidden name=counter value=^counter>
     <input type=submit name=button value=Up>
     <input type=submit name=button value=Down>
   </form>`
```

Figure 5.1 presents the counter as it is rendered by Netscape Navigator. Notice that because a request method is not specified, the request method `GET` is used for the form, which shows in the location field where the form variable key-value pairs are appended to the URL for the file `counter.sml`. In the next example, we shall see that by using the request method `POST`, the key-value pairs of form variables do not turn up in the location field.

5.2 Guess a Number

We now demonstrate how to write a small game using SMLserver. As for the previous example, the "Guess a Number" Web game is made up of one `sml`-file `guess.sml`. The Web game uses the `FormVar` functionality explained on page 29 in Section 4.3 to get access to the form variables `n` and `guess`, if present. Here is

the script `guess.sml`:[2]

```
fun returnPage title pic body = Web.return
  `<html>
    <head><title>^title</title></head>
    <body bgcolor=white> <center>
    <h2>^title</h2> <img src=^pic> <p>
     ^(Quot.toString body) <p> <i>Served by <a
        href=http://www.smlserver.org>SMLserver</a>
    </i> </center> </body>
   </html>`

fun mk_form (n:int) =
  `<form action=guess.sml method=post>
     <input type=hidden name=n value=^(Int.toString n)>
     <input type=text name=guess>
     <input type=submit value=Guess>
   </form>`

val _ =
  case FormVar.wrapOpt FormVar.getNatErr "n"
    of NONE =>
       returnPage "Guess a number between 0 and 100"
         "bill_guess.jpg"
         (mk_form (Random.range(0,100) (Random.newgen())))

     | SOME n =>
       case FormVar.wrapOpt FormVar.getNatErr "guess"
         of NONE =>
           returnPage "You must type a number - try again"
             "bill_guess.jpg" (mk_form n)
          | SOME g =>
           if g > n then
             returnPage "Your guess is too big - try again"
               "bill_large.jpg" (mk_form n)
           else if g < n then
             returnPage "Your guess is too small - try again"
               "bill_small.jpg" (mk_form n)
           else
```

[2]File `smlserver_demo/www/web/guess.sml`.

```
returnPage "Congratulations!" "bill_yes.jpg"
  `You guessed the number ^(Int.toString n) <p>
   <a href=guess.sml>Play again?</a>`
```

In the case that no form variable `n` exists, a new random number is generated and the game is started by presenting an introduction line to the player along with a form for entering the first guess. The Web game then proceeds by returning different pages to the user dependent on whether the user's `guess` is greater than, smaller than, or equal to the random number `n`.

Notice that the game uses the `POST` request method, so that the random number that the user is to guess is not shown in the browser's location field. Although in theory, it may take up to 7 guesses for a user to guess the random number, in practice—with some help from the Web browser—it is possible to "guess" the random number using only one guess; it is left as an exercise to the reader to find out how!

Figure 5.2 shows four different pages served by the "Guess a Number" game.

Figure 5.2: Four different pages served by the "Guess a Number" game.

Chapter 6

Extracting Data from Foreign Web Sites

The Internet hosts a large set of Web services, readily available for use by your Web site! Examples of such available services include real-time population clocks (e.g., `http://www.census.gov/cgi-bin/popclock`), currency rate services (e.g., `http://se.finance.yahoo.com`), and a large number of stock quote services (e.g., `http://quotes.nasdaq.com`). In this chapter, we shall see how to extract data from another Web site and how to use the data for content on your own Web site, using what are called regular expressions.

6.1 Grabbing a Page

The SMLserver API has a built-in function `Web.fetchUrl`, with type `string -> string option`, for fetching a page from the Internet and return the page as a string. Upon calling `Web.fetchUrl`, SMLserver connects to the HTTP Web server, specified by the argument URL, which must be fully qualified. The function does not handle redirects or requests for protocols other than HTTP. If the function fails, for instance by trying to fetch a page from a server that is not reachable, the function returns `NONE`. `Web.fetchUrl` has a default timeout of 60 seconds. This can be changed by configuring "FetchUrlTimeOut" with an integer. The function `fetchUrlTime` behaves as `fetchUrl`, but take a timeout as an argument.

Say we want to build a simple currency service that allows a user to type in an amount in some currency and request the value of this amount in some other currency.

First we must find a site that provides currency rates; one such site is Yahoo Finance: `http://se.finance.yahoo.com`. By browsing the site we see how to

obtain currency rates. For instance, if we want to exchange one American Dollar into Danish Kroner then we use the URL

`http://uk.finance.yahoo.com/q?s=USDDKK=X`.

This URL specifies a form variable `s` as the string `=USDDKK=X`, which means that we are interested in the eXchange rate from USD (US Dollars) to DKK (Danish kroner). The currencies that we shall use in our service are abbreviated according to the following table:

Currency	**Abbreviation**
American Dollar	USD
Australian Dollar	AUD
Bermuda Dollar	BMD
Danish Kroner	DKK
EURO	EUR
Norwegian Kroner	NOK
Swedish Kroner	SEK

The service that we shall build is based on two files, a simple HTML file `currency_form.html` that queries the user for the amount and currencies involved (see Figure 6.1). The other file, the script `currency.sml`, is the target of the HTML form; the first part of the script `currency.sml` takes the following form:

```
val getReal = FormVar.wrapFail FormVar.getRealErr
val getString = FormVar.wrapFail FormVar.getStringErr

val a = getReal ("a", "amount")
val s = getString ("s", "source currency")
val t = getString ("t", "target currency")

val url =
  "http://se.finance.yahoo.com/q?s=" ^
  Web.encodeUrl s ^ Web.encodeUrl t ^ "=X"

fun errPage () =
  (Page.return "Currency Service Error"
   `The service is currently not available, probably
    because we have trouble getting information from
    the data source: <a href="^url">^url</a>.`
    ; Web.exit())
```

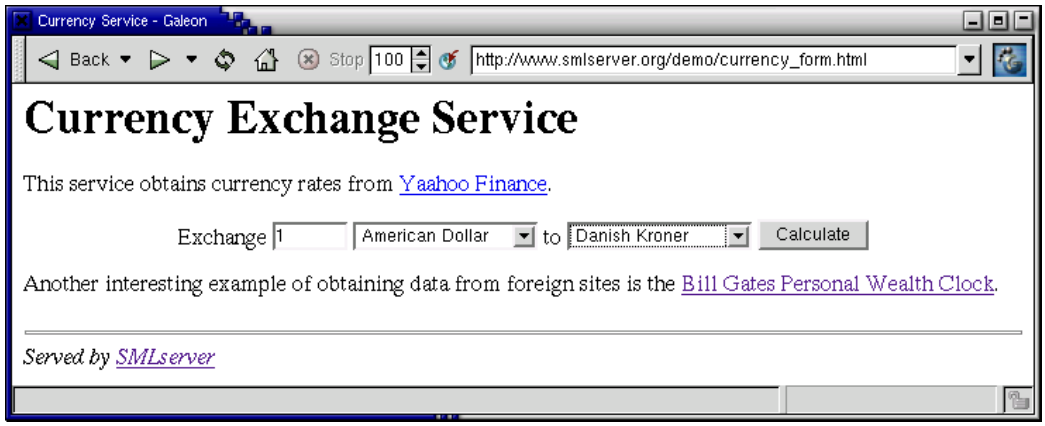


Figure 6.1: The Currency Service entry form, `currency_form.html`.

```
val pg = case Web.fetchUrl url
           of NONE => errPage()
            | SOME pg => pg

(* code that extracts the currency rate from 'pg'
 * and presents calculations for the user ... *)
```

The code constructs the URL by use of the form variables provided by the user. Notice the use of the function `Web.encodeUrl` for building the URL; the function `Web.encodeUrl` encodes characters, such as `&` and `?`, that otherwise are invalid or have special meaning in URLs. The returned page `pg` contains HTML code with the currency information that we are interested in.

Before we continue the description of the currency example, we shall spend the next section on the concept of regular expressions. Later, regular expressions are used to extract the interesting currency information from the page obtained from Yahoo Finance.

6.2 Regular Expressions

In this section we introduce a language of *regular expressions* for classifying strings. A relation called *matching* defines the class of strings specified by a particular regular expression (also called a pattern). By means of the definition of matching, one may ask if a pattern p matches a string s. In the context of building Web sites, there are at least two important uses of regular expressions:

1. Checking form data by ensuring that data entered in forms follow the expected syntax. If a number is expected in an HTML form, the server program must check that it is actually a number that has been entered. This particular use of regular expressions is covered in Chapter 8. Regular expressions can only check syntax; that is, given a date, a regular expression cannot easily be used to check the validity of the date (e.g., that the date is not February 30). However, a regular expression may be used to check that the date has the ISO-format `YYYY-MM-DD`.

2. Extracting data from foreign Web sites, as in the Currency Service above.

In the following we shall often use the term "pattern" instead of the longer "regular expression". The syntax of regular expressions is defined according to the description in Figure 6.2.

A character class *class* is a set of ASCII characters defined according to Figure 6.3.

Potential use of regular expressions is best illustrated with a series of examples:

- `[A-Za-z]` : matches all characters in the english alphabet.
- `[0-9][0-9]` : matches numbers containing two digits, where both digits may be zero.
- `(cow|pig)s?` : matches the four strings `cow`, `cows`, `pig`, and `pigs`.
- `((a|b)a)*` : matches `aa`, `ba`, `aaaa`, `baaa`, ...
- `(0|1)+` : matches the binary numbers (i.e., `0`, `1`, `01`, `11`, `011101010`,...).
- `..` : matches two arbitrary characters.
- `([1-9][0-9]+)/([1-9][0-9]+)` : matches positive fractions of whole numbers (e.g., 1/8, 32/5645, and 45/6). Notice that the pattern does not match the fraction 012/54, nor 1/0.
- `<html>.*</html>` : matches HTML pages (and text that is not HTML).
- `www\.(((it-c|itu)\.dk)|(it\.edu))` : matches the three Web addresses `www.itu.dk`, `www.it-c.dk`, and `www.it.edu`.
- `http://hug.it.edu:8034/ps2/(.*)\.sml` : matches all URLs denoting `.sml` files on the machine `hug.it.edu` in directory `ps2` for the service that runs on port number 8034.

In the next section, we turn to see how regular expressions may be used with SMLserver.

p	**Definition**
`.`	matches all characters
c	matches the character c
\c	matches the escaped character c, where c is one of `|`, `*`, `+`, `?`, `(`, `)`, `[`, `]`, `$`, `.`, `\`, `t`, `n`, `v`, `f`, `r`
p_1p_2	matches a string s if p_1 matches a prefix of s and p_2 matches the remainder of s (e.g., the string `abc` is matched by the pattern `a.c`)
p`*`	matches 0, 1, or more instances of the pattern p (e.g., the strings `abbbbbba` and `aa` are matched by the pattern `ab*a`)
`(`p`)`	matches the strings that match p (e.g., the string `cababcc` is matched by the pattern `c(ab)*cc`
p`+`	matches 1 or more instances of the pattern p (e.g., the pattern `ca+b` matches the string `caaab` but not the string `cb`)
p_1`|`p_2	matches strings that match either p_1 or p_2 (e.g., the pattern (`pig|cow`) matches the strings `pig` and `cow`)
`[`*class*`]`	matches a character in *class*; the notion of character class is defined below. The pattern `[abc1-4]*` matches sequences of the characters `a`, `b`, `c`, `1`, `2`, `3`, `4`; the order is insignificant.
`[^`*class*`]`	matches a character not in *class*. The pattern `[^abc1-4]*` matches sequences of all the characters except `a`, `b`, `c`, `1`, `2`, `3`, `4`.
`$`	matches the empty string
p`?`	matches 0 or 1 instances of the pattern p (e.g., the strings `aa` and `aba` matches the pattern `ab?a`, but the string `abba` does not match the pattern `ab?a`).

Figure 6.2: The syntax of regular expressions (patterns). The letter p is used to range over regular expressions. The word *class* is used to range over classes, see Figure 6.3.

class	**Definition**
c	class containing the specific character c
\c	class containing the escaped character c, where c is one of `|`, `*`, `+`, `?`, `(`, `)`, `[`, `]`, `$`, `.`, `\`, `t`, `n`, `v`, `f`, `r`.
c_1-c_2	class containing ASCII characters in the range c_1 to c_2 (defined by the characters' ASCII value)
	the empty class
$class_1\,class_2$	class composed of characters in $class_1$ and $class_2$

Figure 6.3: The syntax of character classes. Character classes are ranged over by *class*.

6.3 The Structure `RegExp`

SMLserver contains a simple interface for the use of regular expressions:

```
structure RegExp :
  sig
    type regexp
    val fromString : string -> regexp
    val match      : regexp -> string -> bool
    val extract    : regexp -> string -> string list option
  end
```

The function `RegExp.fromString` takes a textual representation of a regular expression (pattern) and turns the textual representation into an internal representation of the pattern, which may then be used for matching and extraction. The function `RegExp.fromString` raises the exception `General.Fail(`*msg*`)` in case the argument is not a regular expression according to the syntax presented in the previous section.

The application `RegExp.match` p s returns `true` if the pattern p matches the string s; otherwise `false` is returned. The following table illustrates the use of the `RegExp.match` function:

Expression	**Evaluates to**
`match (fromString "[0-9]+") "99"`	`true`
`match (fromString "[0-9]+") "aa99AA"`	`false`
`match (fromString "[0-9]+.*") "99AA"`	`true`
`match (fromString "[0-9]+") "99AA"`	`false`
`match (fromString "[0-9]+") "aa99"`	`false`

The second expression evaluates to `false` because the pattern `[0-9]+` does not match the strings `aa` and `AA`. A number of additional examples are available in the file `smlserver_demo/www/web/regexp.sml`.

The application `RegExp.extract` r s returns `NONE` if the regular expression r does not match the string s. It returns `SOME(`l`)` if the regular expression r matches the string s; the list l is a list of all substrings in s matched by some regular expression appearing in parentheses in r. Strings in l appear in the same order as they appear in s. Nested parentheses are supported, but empty substrings of s that are matched by a regular expression appearing in a parenthesis in r are not listed in l.

For a group that takes part in the match repeatedly, such as the group `(b+)` in pattern `(a(b+))+` when matched against the string `abbabbb`, all matching substrings are included in the result list: `["bb", "abb", "bbb", "abbb"]`.

For a group that does not take part in the match, such as `(ab)` in the pattern `(ab)|(cd)` when matched against the string `cd`, a list of only one match is returned, a match for `(cd)`: `["cd"]`.

Again, the use of regular expressions for string extraction is best illustrated with a series of examples:

- Name and telephone. The application

```
extract "Name: ([a-zA-Z ]+);Tlf: ([0-9 ]+)"
  "Name: Hans Hansen;Tlf: 55 55 55 55"
```

  evaluates to

```
SOME ["Hans Hansen", "55 55 55 55"]
```

- Email. The application

```
extract "([a-zA-Z][0-9a-zA-Z\._]*)@([0-9a-zA-Z\._]+)"
  "name@company.com"
```

  evaluates to `SOME ["name","company.com"]`. The application

```
extract "([a-zA-Z][0-9a-zA-Z\._]*)@([0-9a-zA-Z\._]+)"
  "name@company@com"
```

  evaluates to `NONE`.

- Login and Email. The application

```
extract "(([a-zA-Z][0-9a-zA-Z._]*)@[0-9a-zA-Z._]+,?)*"
  "joe@it.edu,sue@id.edu,pat@it.edu")
```

evaluates to

```
SOME ["joe", "joe@it.edu,", "sue", "sue@id.edu,",
      "pat", "pat@it.edu"]}.
```

For more examples, consult the file `regexp.sml` in the demonstration directory `smlserver_demo/www/web/`.

6.4 Currency Service—Continued

We are now ready to continue the development of the Currency Service initiated in Section 6.1. Recall that we have arranged for a page containing currency information to be fetched from the Yahoo Finance Web site. What we need to do now is to arrange for the currency information to be extracted from the fetched page, which is available as a string in a variable `pg`. By inspection, we learn that at one time `pg` contains the following HTML code:

```
<table>
  ...
  AUDSEK=X</a></td><td>200.0</td><td>23:18</td>
  <td>5.468220</td><td><b>1,093.64</b></td></tr>
</table>
```

The pattern `.+AUDSEK.+<td>([0-9]+).([0-9]+)</td>.+"` may be used to extract the rate 5.468220. With this pattern, it is not the value 200.0 that is extracted, because with regular expressions, it is always the longest match that is returned.

Here is the remaining part of the script `currency.sml`—continued from page 41:

```
val pattern = RegExp.fromString
  (".+" ^ s ^ t ^ ".+<td>([0-9]+).([0-9]+)</td>.+")

fun getdate() =
  Date.fmt "%Y-%m-%d" (Date.fromTimeLocal (Time.now()))

fun round r =
  Real.fmt (StringCvt.FIX(SOME 2)) r
```

```
val _ =
  case RegExp.extract pattern pg
    of SOME [rate1, rate2] =>
      (let
         val rate = Option.valOf
           (Real.fromString (rate1^"."^rate2))
       in
         Page.return ("Currency Service - " ^ getdate())
         `^(Real.toString a) (^s) gives you
          ^((round (a*rate))) (^t).<p> The rate used
          is ^(round rate) and is obtained from
          <a href="^url">^url</a>.<p>
          New <a href="currency.html">Calculation</a>?`
       end handle _ => errPage())
     | _  => errPage()
```

The function `RegExp.extract` returns the empty string if there is no match, which is likely to happen when Yahoo Finance changes the layout of the page.

6.5 Caching Support

It can happen that small easy-to-write services become tremendously popular. One example of such a Web service is Bill Gates Personal Wealth Clock (available from `http://philip.greenspun.com/WealthClockIntl`), which estimates your personal contribution to Bill Gates' wealth, using stock quotes from either NASDAQ (`http://quotes.nasdaq.com`) or Security APL (`http://qs.secapl.com`), public information about the world population from the U.S. Census Bureau (`http://www.census.gov/cgi-bin/ipc/popclockw`), and the estimated holding of Microsoft shares owned by Bill Gates. The Web site provides a precise description of the math involved. As of January 7, 2003, the Web site estimates that each and every person in the world has contributed $9.87 to Bill Gates.

This service got popular around the summer 1996 with a hit rate of two requests per second. Such a hit rate is extreme for a service that obtains data from two external sites; not only is it bad netiquette to put an extreme load on external sites for querying the same information again and again, but it almost certainly causes the Web site to break down, which of course lowers the popularity of the site.

There is a simple solution; have your Web server cache the results obtained from the foreign services for a limited amount of time. The wealth clock does not depend on having up-to-the-minute information (e.g., updates every 10 minutes

are probably accurate enough). The SMLserver API has a simple caching interface that can be used to cache data so that requests may share the same information. Another use of the cache mechanism is for authentication, which is covered in Chapter 9.

6.6 The Cache Interface

SMLserver has a simple type safe caching interface that can be used to cache data so that information computed by some script invocation can be used by subsequent script invocations. The cache functionality is implemented as a structure `Cache`, which matches the signature `WEB_CACHE` listed below:[1]

```
signature WEB_CACHE =
  sig
    datatype kind =  WhileUsed of Time.time option * int option
                   | TimeOut   of Time.time option * int option
    type ('a,'b) cache
    include WEB_SERIALIZE
    type name = string

    val get : 'a Type * 'b Type * name * kind -> ('a,'b) cache

    val lookup : ('a,'b) cache -> 'a -> 'b option
    val insert : ('a,'b) cache * 'a * 'b * Time.time option -> bool
    val flush  : ('a,'b) cache -> unit

    val memoize  : ('a,'b) cache -> ('a -> 'b) -> 'a -> 'b
    val memoizeTime  : ('a,'b) cache -> ('a -> 'b * Time.time option) ->
                                                          'a -> 'b
    val memoizePartial  : ('a,'b) cache -> ('a -> 'b option) -> 'a -> 'b
    val memoizePartialTime  : ('a,'b) cache ->
                                  ('a -> 'b * Time.time option) option ->
                                                          'a -> 'b

  end
```

A cache of type `(`α`,`β`) cache` maps keys of type α `Type` to values of type β `Type`. The `WEB_SERILIZE` signature defines a set of *base types* (i.e., `Int`, `Real`, `Bool`, `Char` and `String`) and a set of type constructors to build new types (i.e., `Pair`, `Option`, `List` and `Triple`). A cache has a *cache name*, which is represented by a Standard ML string.SMLserver supports two *kinds* of caches:

[1] The structure `Cache` is accessed through the `Web` structure.

- Timeout caches. For caches of kind `TimeOut(SOME(`t`), `s`)`, an entry inserted with timeout `SOME(`t'`)` expires $\min(t, t')$ from insertion. If an entry is inserted with `NONE`, t is used and vice versa. If both are `NONE` no timeout occurs. If s is `SOME(s')` then the cache will not grow larger than s'. Otherwise the cache might grow very large. If supplied with a timeout this kind of cache guarantees that the cache is updated with freshly computed information, even if the cache is accessed constantly.

  This cache strategy is well suited for caching HTML pages obtained from foreign sites. In Section 6.8, we shall see how the Currency Service of Sections 6.1 and 6.4 is extended to cache currency information obtained from a foreign site.

- Keep-while-used caches. An entry inserted with timeout t' in a cache of kind `WhileUsed(SOME(`t`), `s`)` expires when it has not been accessed within $\min(t, t')$. This kind of cache is useful for caching authentication information, such as passwords, so as to lower the pressure on the RDBMS. If s is provided the cache will not grow large than this.

The function `get` obtains a cache given a domain type, a range type, a cache name, and a cache kind. The first time `get` is called with a particular domain type (*aType*), a particular range type (*bType*), a particular cache name (*cn*) and a particular cache kind (*ck*), a new cache is constructed. If a cache c with cache name *cn* and cache kind *ck* already exists, then there are two possibilities to consider:

1. If the cache c is a mapping from *aType* to *bType*, then c is returned.
2. If the cache c is not a mapping from *aType* to *bType*, then a new cache *c'* is created and returned.

It is possible to create two caches with the same name, but only if they describe mappings of different type or they are defined with different cache kinds.

Conceptually one can think of the function `get` as having the constrained (or bounded) polymorphic type [FP02]

$$\forall\, \alpha \leq \texttt{Type},\ \beta \leq \texttt{Type}\ .\ \texttt{name * kind -> (}\alpha, \beta\texttt{) cache}$$

where `Type` denotes the set of types supported by the cache interface. As an example, the following expression constructs a cache named `mycache`, which maps pairs of integers to lists of reals:

```
get (Pair Int Int, List Real, "mycache", WhileUsed (NONE, SOME(9*1024)))
```

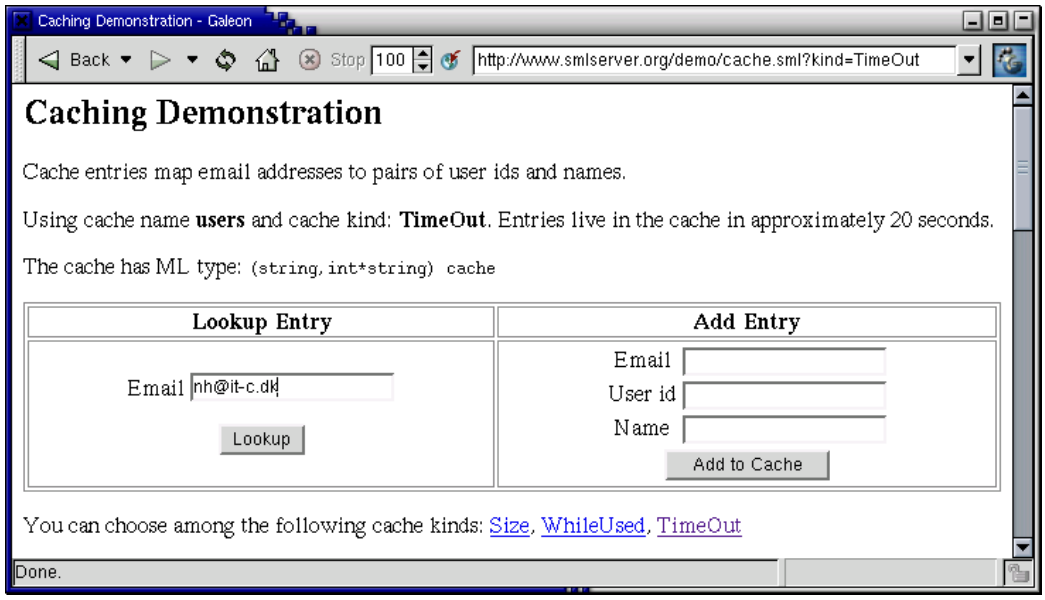


Figure 6.4: The example uses a cache to store a mapping from email addresses to pairs of names and user ids. Cached values are accessible 20 seconds after the last use.

The cache interface also provides functions for adding entries (`insert`), looking up entries (`lookup`) and flushing caches (`flush`). The function `flush` deletes all entries in a cache, but the cache still exists. A cache cannot be deleted.

Caching Demonstration

In the remainder of this Section, we present a small caching demonstration, which implements the mapping from email addresses to pairs of names and user ids.[2] Figure 6.4 shows the entry form.

The entry form shows first the name of cache used, the cache kind and cache type. Below the table, you see three links. Clicking the links will change the cache kind used for caching entries. Notice, that three different caches are created if you click the three links.

The cache kind is passed as a hidden form variable, with name `kind`, to the script `cache_add.sml`:[3]

[2] File `smlserver_demo/www/web/cache.sml`.

[3] File `smlserver_demo/www/web/cache_add.sml`.

```
val kind = Option.valOf (Web.Conn.formvar "kind") handle _ => "Size"

val cache =
  let
    val k =
      case kind of
         "WhileUsed" =>
           Web.Cache.WhileUsed (SOME(Time.fromSeconds 20),SOME(10000))
       | "TimeOut" =>
           Web.Cache.TimeOut (SOME(Time.fromSeconds 20), SOME(10000))
       | "Size" => Web.Cache.WhileUsed (NONE, SOME(10000))
  in
    Web.Cache.get (Web.Cache.String,
      Web.Cache.Pair Web.Cache.Int Web.Cache.String, "users", k)
  end
```

The function `get` is used to obtain a cache with the specified name (`users`), the chosen cache kind (variable `k`) and cache type. We use the type constructor `Pair` to build the range type using the base types `Int` and `String`. A timeout value of 20 seconds is used for cache kinds `WhileUsed` and `TimeOut`. A cache size of 100 bytes is used for cache kind `Size`.

The script `cache_add.sml` also processes the email, user id and name entered in the entry form at the left:

```
val new_p = (* new_p true if new value added *)
  case (Web.Conn.formvar "email", Web.Conn.formvar "name",
        Web.Conn.formvar "uid", Web.Conn.formvar "timeout") of
    (SOME email, SOME name, SOME uid, SOME timeout) =>
      Web.Cache.insert(cache,email,
      (Option.getOpt(Int.fromString uid,0) ,name),
      Option.map Time.fromSeconds (Int.fromString timeout))
  | _ => false

val head = if new_p then "New Value added"
           else "Key already in Cache"

val _ = Page.return ("Caching Demonstration" ^ ": cache_add.sml")
  (`^head <p>

` (*^^  `Pretty printing the cache:
  <pre>
```

```
  ^(Web.Cache.pp_cache cache)
  </pre><p> `*) ^^ `

  Go back to <a href=cache.sml?kind=^kind>
  Cache Demo Home Page</a>.`)
```

We use user id 0, in case the user id given in form variable `uid` is not an integer.

The code to lookup a name in the cache is in the script `cache_lookup.sml`. Again, the cache kind is passed as a hidden form variable and used to obtain the cache. The variable `cache` contains the cache obtained with function `get`; this is similar to `cache_add.sml` and not shown below. The function `lookup` is used to lookup the entry associated with the email stored in the form variable `email`. The function `lookup` returns `NONE` if the email address is not in the cache:[4]

```
val kind = ...
val cache = ...

fun returnPage s = Page.return "Caching Demonstration"
  (`^s <p>

  Using cache kind: ` ^^ (pp_kind kind) ^^ `<p>

  Go back to <a href=cache.sml?kind=^kind>
  Cache Demo Home Page</a>.`)

val _ = (* new_p is true if new value added *)
  case Web.Conn.formvar "email"
      of NONE => Web.returnRedirect "cache.sml"
       | SOME email =>
        returnPage
        (case Web.Cache.lookup cache email
           of SOME(uid,name) => "Name and userid for " ^
                        email ^ " is: (" ^ name ^ "," ^
                        (Int.toString uid) ^ ")"
         | NONE => "No name in cache for " ^ email)
```

6.7 Memoization

The function `memoize` adds caching functionality (i.e., memoization) to a function. Assuming that the function f has type `int -> string * real` and c is an appro-

[4] File `smlserver_demo/www/web/cache_lookup.sml`.

priately typed cache, the expression `memoize` c f returns a new function f', which caches the results of evaluating the function f. Subsequent calls to f' with the same argument results in cached pairs of strings and reals, except when a result no longer lives in the cache, in which case f is evaluated again.

Consider the Fibonacci function with type `int -> int`:[5]

```
fun fib 0 = 1
  | fib 1 = 1
  | fib n = fib (n-1) + fib (n-2)
```

To implement a cached version of `fib` we create a cache (named `fib`) with cache type (`int`,`int`) `cache`:

```
val cache =
  Web.Cache.get(Web.Cache.Int, Web.Cache.Int, "fib",
                Web.Cache.WhileUsed(SOME(Time.fromSeconds 20),
                                    SOME(10000)))
```

To implement a memorized version of `fib` we can do the following:

```
val fib_m n = (Web.Cache.memoize cache fib) n
```

However, the function `fib_m` only caches the result, not the intermediate results. First evaluating `fib_m` 1000 and then `fib_m` 999 will not be different from evaluating `fib` 1000 and then `fib 999`. After evaluating `fib_m` 1000 the cache will only contain one entry, which cannot be used when evaluating `fib_m` 999.

The following implementation does store intermediate results in the cache, which is also the implementation you find in file `cache_fib.sml`:

```
fun fib_m 0 = 1
  | fib_m 1 = 1
  | fib_m n = fib' (n-1) fib' (n-2)
and fib' n = (Web.Cache.memoize cache fib_m) n
```

6.8 Caching Version of Currency Service

In this section we demonstrate the memoization function `memoize` in the context of the Currency Service of Sections 6.1 and 6.4. Similarly to the Bill Gates Personal Wealth Clock, our Currency Service should not access Yahoo Finance on each and

[5] File `smlserver_demo/www/web/cache_fib.sml`.

every access. Instead, the currency rates obtained from Yahoo are cached in 300 seconds (five minutes).

Notice the distinction between the cache strategies `TimeOut` and `WhileUsed`; the service should not make use of the cache strategy `WhileUsed` because rates must be updated every 300 seconds—irrespectively of whether the service is accessed every minute. Here is the script `currency_cache.sml`,[6] which implements the cached version of the Currency Service:

```
structure C = Web.Cache

val getReal = FormVar.wrapFail FormVar.getRealErr
val getString = FormVar.wrapFail FormVar.getStringErr

val a = getReal ("a", "amount")
val s = getString ("s", "source currency")
val t = getString ("t", "target currency")

val url = "http://se.finance.yahoo.com/m5?s=" ^
  Web.encodeUrl s ^ "&t=" ^ Web.encodeUrl t

fun errPage () =
  (Page.return "Currency Service Error"
   `The service is currently not available, probably
    because we have trouble getting information from
    the data source: <a href="^url">^url</a>.`
    ; Web.exit())

fun getdate () =
  Date.fmt "%Y-%m-%d" (Date.fromTimeLocal (Time.now()))

fun round r = Real.fmt (StringCvt.FIX(SOME 2)) r

val pattern = RegExp.fromString
  (".+" ^ s ^ t ^ ".+<td>([0-9]+).([0-9]+)</td>.+")

val cache = C.get (C.String,C.Option C.Real,"currency",
              C.TimeOut (SOME(Time.fromSeconds(5*60)),
                         SOME(10000)))
```

[6] File `smlserver_demo/www/web/currency_cache.sml`.

```
val fetch = C.memoize cache
  (fn url => case Web.fetchUrl url
               of NONE => NONE
                | SOME pg =>
                  (case RegExp.extract pattern pg
                     of SOME [r1,r2] =>
                          Real.fromString (r1 ^ "." ^ r2)
                      | _ => NONE))

val _ =
  case fetch url of
    NONE => errPage ()
  | SOME rate =>
      Page.return
        ("Currency Exchange Service, " ^ getdate())
        `^(Real.toString a) ^s gives ^(round (a*rate)) ^t.<p>
        The exchange rate is obtained by fetching<p>
        <a href="^url">^url</a><p>
        New <a href="currency_cache.html">Calculation</a>`
```

The program creates a cache `cache` that maps strings (base type `String`) to optional reals (constructed type `Option Real`).

The anonymous function passed to the function `memoize` tries to fetch a page from Yahoo Finance and extract the currency rate for the currencies encoded in the argument URL. Now, when passed to the function `memoize`, the fetching function is executed only if no currency rate is associated with the argument URL in the cache named `currency`. Notice that only currency rates are stored in the cache, not the entire fetched pages.

Chapter 7

Connecting to a Relational Database Management System

Until now, the Web applications that we have looked at have been in the category of "Web sites that are programs." In this chapter, we exploit the possibility of implementing Web applications that fall into the category "Web sites that are databases." The ability of a Web application accessing and manipulating information stored in some sort of database drastically widens the kind of Web applications that one can build.

There are many possible ways in which a Web application may keep track of data between sessions. One possibility is to use the file system on the machine on which the Web server runs for storing and reading data. Another possibility is to use some sort of Web server support for maintaining state between sessions to create and manipulate task-specific data structures. Yet another possibility is to use some proprietary relational database management system for storing and accessing data.

What we argue in the following is that, unless you have some very good reasons, you want data on the server to be maintained exclusively by a Relational Database Management System (RDBMS), perhaps with the addition of some simple caching support.

Let us assume for a moment that you have constructed a Web based system that uses the local file system for storing and accessing data. By request from the management department, you have constructed a Web based system for managing employee data such as office location, home addresses, and so on. The system that you came up with even has a feature that allows an employee to maintain a "What am I doing now" field. You have spent weeks developing the system. Much of the time was spent designing the layout of the data file and for writing functions for parsing and writing employee data. You have tested the system with

a few employees added to the data file and you have even been careful using locks to prevent one Web script from writing into the data file while some other Web script is reading it, and vice versa. The system is launched and the employees are asked to update the "What am I doing now" field whenever they go to a meeting or such. For the three managers and the 20 employees in the management department, the system works great; after two weeks, the success of your Web based employee system has spread to other departments in the organization. Gradually, more departments start using your system, but at some point people start complaining about slow response times, especially around lunch-time where everyone of the 300 employees that now use the system wants to update the "What am I doing now" field.

After a few days of complaints, you get the idea that you can read the data file into an efficient data structure in the Web server's memory, thereby getting quicker response and update times, as long as you write log files to disk that say how the data file should be updated so as to create a valid data file. After a few more weeks of development—and only a little sleep—the system finally performs well. You know that there are issues that you have not dealt with. For example, what happens if somebody shuts down the machine while a log file is written to disk? Is the system then left in an inconsistent state?

You start realizing that what you have been doing the last month is what some companies have been doing successfully for decades; you have developed a small database management system, although tailored specifically to your problem at hand and very fragile to changes in your program. You decide to modify your Web application to use a database management system instead of your home-tailored file-based system. But there are many database management systems to choose from! The next sections tell you something about what properties you want from a database management system.

7.1 What to Expect from an RDBMS

Decades of research and development in the area of database management systems have resulted in easily adaptable systems, which efficiently solve the problem of serving more than one user at the same time. In some systems, such as the Oracle RDBMS, readers need not even wait for writers to finish! Here is a list of some of the features that an RDBMS may provide:

- Methods for query optimizations. An RDBMS supports known methods for optimizing queries, such as index creation for improving query performance.
- Data abstraction. Through the use of SQL, an RDBMS may help programmers abstract from details of data layout.

- Support for simultaneous users. RDBMS vendors have solved the problems of serving simultaneous users, which make RDBMSs ideal for Web purposes.
- System integration. The use of standardized SQL eases system integration and inter-system communication.
- Failure recovering. A good RDBMS comes with support for recovering from system failures and provides methods for backing up data while the system is running.

7.2 The ACID Test

If you want to sleep well at night while your Web site is serving user requests, you want your RDBMS of choice to support *transactions*. Basically, what this means is that you want your RDBMS to pass the *ACID test* [Gre99]:

- Atomicity. A transaction is either fully performed or not performed. Example: When money is transferred from one bank account to another, then either both accounts are updated or none of the accounts is updated.
- Consistency. A transaction sends a database from one consistent state to another consistent state. Transactions that would send the database into an inconsistent state are not performed. Example: A bank may specify, using consistency constraints, that for some kinds of bank accounts, the account balance must be positive. Transaction specifying a transfer or a withdrawal causing the balance on such an account to be negative are not performed.
- Isolation. A transaction is invisible to other transactions until the transaction is fully performed. Example: If a bank transaction transfers an amount of money from one account to another account while at the same time another transaction computes the total bank balance, the amount transferred is counted only once in the bank balance.
- Durability. A complete transaction survives future crashes. Example: When a customer in a bank has successfully transferred money from one account to another, a future system crash (such as power failure) has no influence on the effect of the transaction.

Two RDBMSs that pass the ACID test are the proprietary Oracle RDBMS and the Open Source RDBMS Postgresql, both of which are supported by SMLserver.

The language used to communicate with the RDBMS is the standardized Structured Query Language (SQL), although each RDBMS has its own extensions to

the language. SQL is divided into two parts, a Data Definition Language (DDL) and a Data Manipulation Language (DML).

Although this book is not meant to be an SQL reference, in the next two sections, we discuss the two parts of the SQL language in turns.

7.3 Data Modeling

The term "data modeling" covers the task of defining data entities (tables) and relations between entities. The SQL data definition language contains three commands for creating, dropping and altering tables, namely `create table`, `drop table`, and `alter table`.

`create table`

The SQL command `create table` takes as argument a name for the table to create and information about the table columns in terms of a name and a data type for each column. The following `create table` command specifies that the table `employee` be created with five columns `email`, `name`, `passwd`, `note`, and `last_modified`.

```
create table employee (
  email          varchar(200) primary key not null,
  name           varchar(200) not null,
  passwd         varchar(200) not null,
  note           varchar(2000),
  last_modified  date
);
```

There are a variety of column data types to choose from and each RDBMS has its own extensions to SQL, also in this respect. The column data type `varchar(200)` specifies that the column field can contain at most 200 characters, but that shorter strings use less memory. The column data type `date` is used for storing dates.

The command also specifies some *consistency constraints* on the data, namely that the columns `email`, `name`, and `passwd` must be non-empty—specified using the `not null` constraint. The `primary key` constraint on the `email` column has two purposes. First, it specifies that no two rows in the table may have the same email address. Second, the constraint specifies that the RDBMS should maintain an index on the email addresses in the table, so as to make lookup of email addresses in the table efficient.

alter table

The `alter table` command is used to modify already existing tables, even when data appears in the table. The `alter table` command takes several forms. The simplest form makes it possible to drop a column from a table:[1]

```
alter table employee drop last_modified;
```

Here the column `last_modified` is eliminated from the table. A second form makes it possible to add a column to a table:

```
alter table employee add salary integer;
```

In this example, a column named `salary` of type `integer` is added to the `employee` table. The `update` command may be used to initialize the new column as follows:

```
update employee set salary = 0 where salary = NULL;
```

drop table

The `drop table` command is used to remove a table from a database. As an example, the command

```
drop table employee;
```

removes the table `employee` from the database.

7.4 Data Manipulation

The four most useful SQL data manipulation commands are `insert`, `select`, `delete`, and `update`. In this section, we give a brief overview of these commands.

insert

Each `insert` command corresponds to inserting one row in a table. An example `insert` command takes the following form:

```
insert into employee (name, email, passwd)
values ('Martin Elsman', 'mael@it.edu', 'don''tforget');
```

[1] This form is not supported by the Postgresql 7.2 RDBMS.

There are several things to notice from this `insert` command. First, values to insert in the table appears in the order column names are specified in the command. In this way, the order in which column names appeared when the table was created has no significance for the `insert` command. Second, not all columns need be specified; only those columns for which a `not null` constraint is specified in the `create table` command must be mentioned in the `insert` command—for the remaining columns, `null` values are inserted. Third, string values are written in quotes (`'...'`). For a quote to appear within a string, the quote is escaped by using two quotes (`''`). Here is another example `insert` command:

```
insert into employee (email, name, passwd, note)
values ('nh@it.edu', 'Niels Hallenberg', 'hi', 'meeting');
```

`select`

The `select` command is used for querying data from tables. Here is an example querying all data from the `employee` table:

```
select * from employee;
```

The result includes the two rows in the `employee` table:

email	**name**	**passwd**	**note**
`mael@it.edu`	`Martin Elsman`	`don'tforget`	**null**
`nh@it.edu`	`Niels Hallenberg`	`hi`	`meeting`

Notice that only one quote appears in the `passwd` string "`don'tforget`".

The `select` command allows us to narrow the result both horizontally and vertically. By explicitly mentioning the columns of interest, only the mentioned columns appear in the result. Similarly, the `select` command may be combined with `where` clauses, which narrows what rows are included in the result. Consider the following `select` command:

```
select name, passwd
from employee
where email = 'mael@it.edu';
```

The result of this query contains only one row with two columns:

name	**passwd**
`Martin Elsman`	`don'tforget`

Because the column `email` is primary key in the `employee` table, the RDBMS maintains an index that makes lookup based on email addresses in the table efficient; thus, the data model we have chosen for employees scales to work well even for millions of employees.

The `select` command may be used in many other ways than shown here; in the sections to follow, we shall see how the `select` command can be used to select data from more than one table simultaneously, through what is called a *join*, and how the `group by` clause may be used to compute a summary of the content of a table.

`update`

As the name suggests, the `update` command may be used to update a number of rows in a table. The following example `update` command uses a `where` clause to update the content of the `note` column for any employee with email-address `nh@it.edu`—of which there can be at most one, because `email` is a key:

```
update employee
set note = 'back in office'
where email = 'nh@it.edu';
```

Here is an example that updates more than one column at the same time:

```
update employee
set note = 'going to lunch',
    passwd = 'back'
where email = 'mael@it.edu';
```

After the two update commands, the `employee` table looks as follows:

email	name	passwd	note
mael@it.edu	Martin Elsman	back	going to lunch
nh@it.edu	Niels Hallenberg	hi	back in office

`delete`

The `delete` command is used to delete rows from a table. As for the `select` and `update` command, one must be careful to constrain the rows that are effected using `where` clauses. An example `delete` command that deletes one row in the `employee` table looks as follows:

```
delete from table employee
where email = 'mael@it.edu';
```

7.5 Three Steps to Success

When developing Web sites backed by a database, we shall often commit to the following three steps:

1. Development of a data model that supports all necessary transactions. This is the hard part.

2. Design of a Web site diagram that specifies names of scripts and how scripts link to each other. Do not underestimate the importance of this part.

3. Implementation of scripts, including the implementation of database transactions using the SQL data manipulation language. This is the easy part!

We emphasize that the easy part of developing a Web site backed by a database is the third part, the implementation of scripts for supporting the appropriate transactions. Not surprisingly, the more time spent on the first two parts, the better are the chances for a satisfactory result.

In general, the construction of a data model results in the creation of a file containing SQL data definition language commands for defining tables and perhaps data manipulation commands for inserting initial data in the tables.

The construction of a data model for the employee example results in a file `employee.sql`[2] containing only a few data definition language commands and two `insert` commands for inserting example data in the table:

```
drop table employee;
create table employee (
  email          varchar(200) primary key not null,
  name           varchar(200) not null,
  passwd         varchar(200) not null,
  note           varchar(2000),
  last_modified  date
);
insert into employee (name, email, passwd)
  values ('Martin Elsman', 'mael@it.edu', 'don''tforget');
insert into employee (email, name, passwd, note)
  values ('nh@it.edu', 'Niels Hallenberg', 'hi', 'meeting');
```

Notice that the `employee.sql` file contains a `drop table` command; this command turns out to be useful when the `employee.sql` file is reloaded upon changes in the data model.

[2] File `smlserver_demo/web_demo_lib/pgsql/employee.sql`.

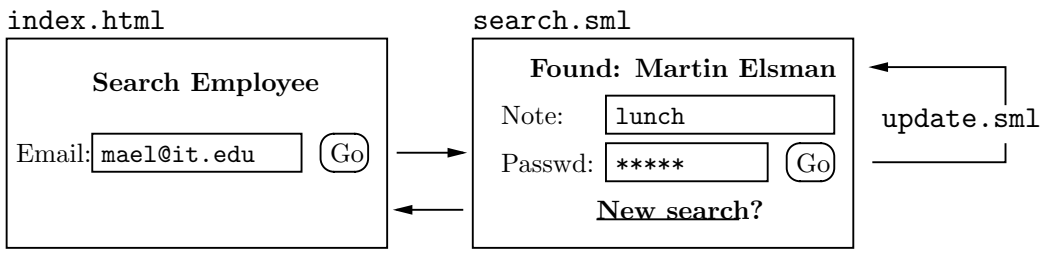


Figure 7.1: Web site diagram for the employee example. Administrator pages for adding and deleting employees are not shown.

To load the data model in a running Postgresql RDBMS, run the program `psql` with the file `employee.sql` as argument:

```
% psql -f employee.sql
DROP
psql:employee.sql:9: \
    NOTICE:  CREATE TABLE/PRIMARY KEY will create implicit \
             index 'employee_pkey' for table 'employee'
CREATE
INSERT 167792 1
INSERT 167793 1
```

For larger data models, it is important to give the data model more thought, perhaps by constructing an Entity-Relation diagram (E-R diagram) for the model; we shall see an example of such an E-R diagram in Section 7.7.

A simple Web site diagram for the employee example is shown in Figure 7.1. The boxes in the diagram represents the different HTML pages that the employee Web application may send to the user. An edge in the diagram represents either a link or a form action. A labeled edge represents an update transaction on the database.

The entry page to the employee example may be implemented as a simple HTML form with action `search.sml`:[3]

```
<html>
  <head><title>Search the Employee Database</title></head>
  <body bgcolor=white>
     <center> <h2>Search the Employee Database</h2> <p>
```

[3]File `smlserver_demo/www/web/employee/index.sml`.

```
        <form action=search.sml method=post>
          Email: <input type=text name=email>
          <input type=submit value=Search>
        </form>
      </center>
    </body>
  </html>
```

Because the result of submitting the form is dependant on the content of the `employee` table, HTML code for the result page must be computed dynamically, which is what the file `search.sml` does (see the next section). Moreover, if a user with a valid password chooses to update the note for a given user, we arrange for the `employee` table to be updated by executing an SQL `update` command from within the `update.sml` script. When the transaction is finished executing, the script sends an HTTP redirect to the client, saying that the client browser should request the file `search.sml`.

7.6 Transactions as Web Scripts

SMLserver scripts may access and manipulate data in an RDBMS through the use of a structure that matches the `WEB_DB` signature.[4] Because SMLserver supports the Oracle RDBMS, the Postgresql RDBMS, and MySQL, there are three structures in the `Web` structure that matches the `WEB_DB` signature, namely `Web.DbOra`, `Web.DbPg`, and `Web.DbMySQL`. The example Web server project file includes a file `Db.sml`, which binds a top-level structure `Db` to the structure `Web.DbPg`; thus, in what follows, we shall use the structure `Db` to access the Postgresql RDBMS. Figure 7.2 lists the part of the RDBMS interface that we use in the following.

To access or manipulate data in an RDBMS, SMLserver scripts need not explicitly open a connection to the RDBMS. Instead, the opening of connections is done at the time the Web server (i.e., Apache) is started, which avoids the overhead of opening connections every time a script is executed.

A *database handle* identifies a connection to an RDBMS and a *pool* is a set of database handles. When the Web server is started, one or more pools are created. At any particular time, a database handle is owned by at most one script. Moreover, the database handles owned by a script at any one time belong to different pools. The functions shown in Figure 7.2 request database handles from the initialized pools and release the database handles again in such a way that deadlocks are avoided; a simple form of deadlock is caused by one thread holding on to a resource A when attempting to gain access to a resource B, while

[4] See the file `smlserver_demo/web_demo_lib/WEB_DB.sml`.

```
signature WEB_DB =
  sig
    val dml  : quot -> unit
    val fold : ((string->string)*'a->'a) -> 'a -> quot -> 'a

    val oneField     : quot -> string
    val oneRow       : quot -> string list
    val zeroOrOneRow : quot -> string list option

    val seqNextvalExp : string -> string
    val qq            : string -> string
    val qqq           : string -> string
    ...
  end
```

Figure 7.2: Parts of the `WEB_DB` signature.

another thread holds on to resource B when attempting to gain access to resource A.

The `WEB_DB` function `dml` with type `quot->unit` is used to execute a data manipulation language command, specified with the argument string, in the RDBMS. On error, the function raises the exception `General.Fail(`*msg*`)`, where *msg* holds an error message. Data manipulation language commands that may be invoked using the `dml` function include the `insert` and `update` statements.

The four functions `fold`, `oneField`, `oneRow`, and `zeroOrOneRow` may be used to access data in the database. In all cases a `select` statement is passed as an argument to the function. The function `fold` requires some explanation. An application `fold` *f b sql* executes the SQL statement given by the quotation *sql* and folds over the result set. The function *f* is the function used in the folding with base *b*. The first argument to *f* is a function that maps column names into values for the row. The function raises the exception `General.Fail(`*msg*`)`, where *msg* is an error message, on error. See the script `wine.sml` listed on page 75 for an example that uses the `fold` function.

Because the number of database handles owned by a script at any one time is limited to the number of initialized pools, nesting of other database access functions with the `fold` function is limited by the number of initialized pools.

The function `qq`, which has type `string->string`, returns the argument string in which every occurrence of a quote (`'`) is replaced with a double occurrence (`''`).

Thus, the result of evaluating `qq("don'tforget")` is the string `"don''tforget"`. The function `qqq` is similar to the `qq` function with the extra functionality that the result is encapsulated in quotes (`'...'`).

The script `search.sml`, which implements the employee search functionality, looks as follows:[5]

```
fun returnPage title body = Web.return
  (`<html>
    <head><title>^title</title></head>
    <body bgcolor=white>
      <center><h2>^title</h2><p>` ^^ body ^^
      `</center>
    </body>
    </html>`)

val email = FormVar.wrapFail
  FormVar.getStringErr ("email","email")

val sql = `select name, note
           from employee
           where email = '^(Db.qqq email)'`

val _ =
  case Db.zeroOrOneRow sql of
    SOME [name, note] =>
      returnPage "Employee Search Success"
      `<form action=update.sml method=post>
         <input type=hidden name=email value="^email">
         <table align=center border=2>
            <tr><th>Name:</th>
                <td>^name</td></tr>
            <tr><th>Email:</th>
                <td>^email</td></tr>
            <tr><th>Note:</th>
                <td><input name=note type=text value="^note">
                </td></tr>
            <tr><th>Password:</th>
                <td><input name=passwd type=password>
                    <input type=submit value="Change Note">
```

[5] File `smlserver_demo/www/web/employee/search.sml`.

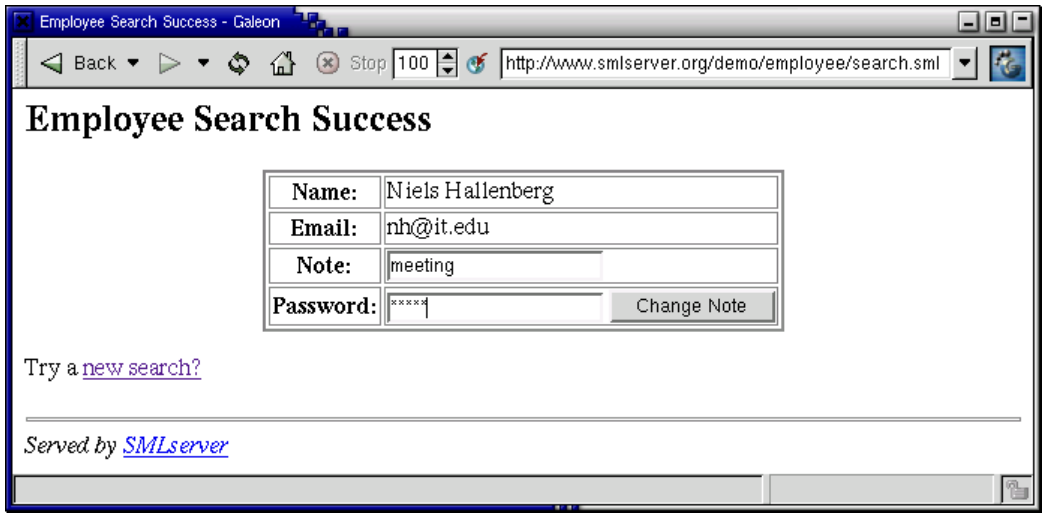


Figure 7.3: The result of searching for an employee with email `nh@it.edu`

```
                </td></tr>
          </table>
        </form><p>
        Try a <a href=index.html>new search?</a>‘
  | _ =>
        returnPage "Employee Search Failure"
        ‘Use the back-button in your Web browser
        to go back and enter another email address‘
```

The expression `FormVar.wrapFail FormVar.getStringErr (var,name)` returns an error page to the user in case form variable `var` is not available or in case it contains the empty string. The argument `name` is used for error reporting. Searching for an employee with email `nh@it.edu` results in the Web page shown in Figure 7.3. The script `update.sml` looks as follows:[6]

```
val getString = FormVar.wrapFail FormVar.getStringErr

val email  = getString ("email","email")
val passwd = getString ("passwd","passwd")
val note   = getString ("note", "note")
```

[6]File `smlserver_demo/www/web/employee/update.sml`.

```
val update = `update employee
             set note = ^(Db.qqq note)
             where email = ^(Db.qqq email)
               and passwd = ^(Db.qqq passwd)`

val _ =
  (Db.dml update;
   Web.returnRedirect ("search.sml?email="
                       ^ Web.encodeUrl email))
  handle _ =>
    Page.return "Employee Database" `Update failed`
```

The function `Web.returnRedirect` returns a redirect, which causes the browser to request the script `search.sml` from the server. The `email` address is sent along to the `search.sml` script as a form variable. The value is URL encoded to support characters other than letters and digits in the `email` address.

7.7 Best Wines Web Site

We now present a somewhat larger example. The example constitutes a wine rating Web site, which we call *Best Wines*. The Best Wines Web site allows users to rate and comment on wines and to see the average rating for a wine in addition to other user's comments.

Recall the three steps to the successful construction of a Web site backed by a database:

1. Development of a data model that supports all necessary transactions
2. Design of a Web site diagram that specifies names of scripts and how scripts link
3. Implementation of scripts, including the implementation of database transactions using the SQL data manipulation language

The next three sections cover these steps for the Best Wines Web site.

Data Model and Transactions

The data modeling process attempts to answer questions that focus on application data. What are the primary data objects that are processed by the system? What

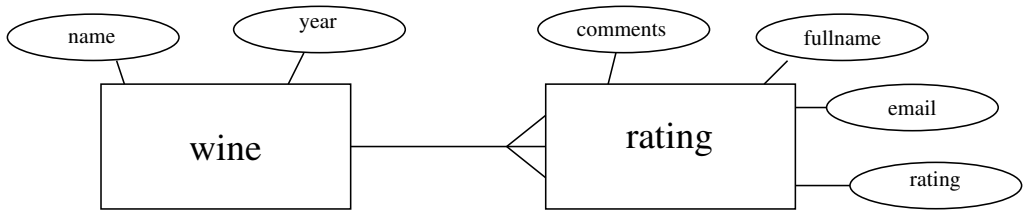


Figure 7.4: E-R diagram for the Best Wine Web site. The fork in the diagram specifies that the relation between the wine-entity and the rating-entity is a one-to-many relation; to every one wine there may be many ratings.

attributes describe each object? What are the relationships between objects? What are the processes that access and manipulate objects?

As the first part of developing a data model for the Best Wines Web site, we construct an Entity-Relationship diagram (E-R diagram) for the Web site, which leads to the construction of SQL data modeling commands for the data model. The second part of the data modeling process focuses on developing the appropriate transactions for accessing and manipulate data.

An *entity-relationship diagram* is composed of three types of components:

1. Entities, which are drawn as rectangular boxes

2. Attributes, which are drawn as ellipses

3. Relationships, which connects entities

When an E-R diagram is constructed for a Web site, it is a straightforward task to develop the corresponding SQL data modeling commands. In fact, entities in the E-R diagram map directly to table names and attributes map to column names in the associated tables. Before we say what relationships map to, consider the E-R diagram for the Best Wine Web site in Figure 7.4.

The E-R diagram contains two entities, `wine` and `rating`. Attributes associated with the `wine` entity include a `name` and a `year` (vintage) for the wine. Attributes associated with the `rating` entity include a user's `comments`, the user's `fullname` and `email`, and a `rating`. Notice that the diagram does not say anything about the data types for the attributes.

The relationship between the entities `wine` and `rating` is a *one-to-many* relationship, that is, to every one wine there may be many ratings. This type of relationship is pictured in the diagram as a fork. In general, there are other types

of relationships besides one-to-many relationships, including one-to-one relationships and many-to-many relationships. Before an E-R diagram can be mapped to SQL data modeling commands, many-to-many relationships are broken up by introducing intermediate entities.

SQL data modeling commands corresponding to the E-R diagram in Figure 7.4 look as follows:[7]

```
create sequence wid_sequence;
create table wine (
       wid       integer primary key,
       name      varchar(100) not null,
       year      integer,
       check     ( 1 <= year and year <= 3000 ),
       unique    ( name, year )
);
create table rating (
       wid       integer references wine,
       comments  varchar(1000),
       fullname  varchar(100),
       email     varchar(100),
       rating    integer,
       check     ( 0 <= rating and rating <= 6 )
);
```

The first command creates an SQL *sequence*, with name `wid_sequence`, which we shall use to create fresh identifiers for identifying wines.

The two entities `wine` and `rating` are transformed into `create table` commands with columns corresponding to attributes in Figure 7.4. Data types for the columns are chosen appropriately. The relationship between the two tables is encoded by introducing an additional column `wid` (with data type `integer`) in each table. Whereas the column `wid` in the `wine` table is declared to be primary (i.e., no two rows have the same `wid` value and an index is constructed for the table, making lookup based on the `wid` value efficient), a referential integrity constraint on the `wid` column in the `rating` table, ensures that a row in the `rating` table is at all times associated with a row in the `wine` table. Additional consistency constraints guarantee the following properties:

- The `year` column is an integer between one and 3000
- No two rows in the `wine` table is associated with the same name and the same year

[7] File `smlserver_demo/web_demo_lib/pgsql/rating.sql`.

- A `rating` in the `rating` table is an integer between zero and six

A list of possible transactions and associated SQL data-manipulation commands are given here:

Wine insertion:

```
insert into wine (wid, name, year)
values (1, 'Margaux - Chateau de Lamouroux', 1988);
```

Rating insertion:

```
insert into rating
   (wid, fullname, email, comments, rating)
values
   (1, 'Martin Elsman', 'mael@it.edu', 'Great wine', 5);
```

Wine comments:

```
select comments, fullname, email, rating
from rating where wid = 1;
```

Wine index:

```
select wine.wid, name, year,
       avg(rating) as average, count(*) as ratings
from wine, rating
where wine.wid = rating.wid
group by wine.wid, name, year
order by average desc, name, year;
```

The difficult transaction is the wine index transaction, which is used in the construction of the main page of the Best Wine Web site (see Figure 7.8). The `select` command computes the average ratings for each wine in the `wine` table. The transaction makes use of the `group by` feature of the `select` command to group rows with the same `wid`, `name`, and `year` columns. For each of the resulting rows, the average rating for the grouped rows is computed as well as the number of rows that are grouped in each group.

Web Site Diagram

A Web site diagram for the Best Wines Web site is shown in Figure 7.5. The Web site is made up of four scripts, three of which construct pages that are returned

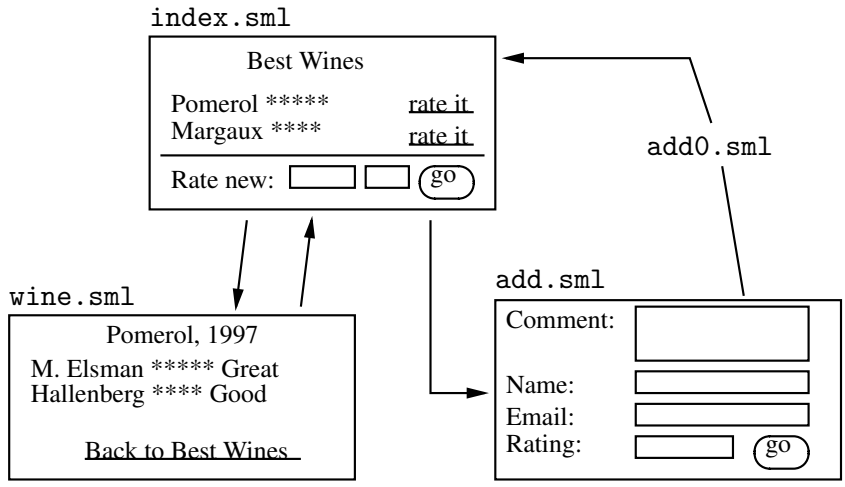


Figure 7.5: Web site diagram for the Best Wine Web site.

to users. The fourth script `add0.sml` implements the rating-insert transaction for inserting a rating in the `rating` table.

The next section describes the implementation of each of the SMLserver scripts.

Implementation of SMLserver Scripts

The scripts `index.sml`, `wine.sml`, `add.sml`, and `add0.sml` make use of functionality provided in a structure `RatingUtil`. We shall not present the structure `RatingUtil` here, but only show its signature:[8]

```
signature RATING_UTIL =
  sig
    (* [returnPage title body] returns page to browser. *)
    val returnPage : string -> string frag list -> unit

    (* [returnPageWithTitle title body] returns page
     * to browser with title as h1-header. *)
    val returnPageWithTitle :
      string -> string frag list -> unit

    (* [bottleImgs n] returns html code for
```

[8] File `smlserver_demo/web_demo_lib/RatingUtil.sml`.

```
     * n bottle images. *)
    val bottleImgs : int -> string

    (* [mailto email name] returns mailto anchor. *)
    val mailto : string -> string -> string
  end
```

The SMLserver scripts also make use of the structure `FormVar` presented in Chapter 8.

The script `wine.sml`

The script `wine.sml` lists user comments for a specific wine. The script assumes a form variable `wid` that denotes the wine. The script uses the `Db.fold` function (see page 67) to construct a page with the comments associated with the specific wine. The page is returned to the user using the `RatingUtil.returnPageWithTitle` function. Here is the listing of the script `wine.sml`:[9]

```
(* Present comments and ratings for a specific wine *)
val wid = FormVar.wrapFail FormVar.getNatErr
  ("wid","internal number")

val query =
  `select comments, fullname, email, rating
   from rating
   where wid = ^(Int.toString wid)`

val lines = Db.fold
  (fn (g,r) =>
   let val rating =
         case Int.fromString (g "rating") of
           SOME i => i
         | NONE => raise Fail "Rating not integer"
   in
     `<tr><th> ^(RatingUtil.bottleImgs rating)
      <td> ^(g "comments")
      <td> ^(RatingUtil.mailto (g "email") (g "fullname"))`
   end ^^ r) `` query

val body =
```

[9] File `smlserver_demo/www/web/rating/wine.sml`.

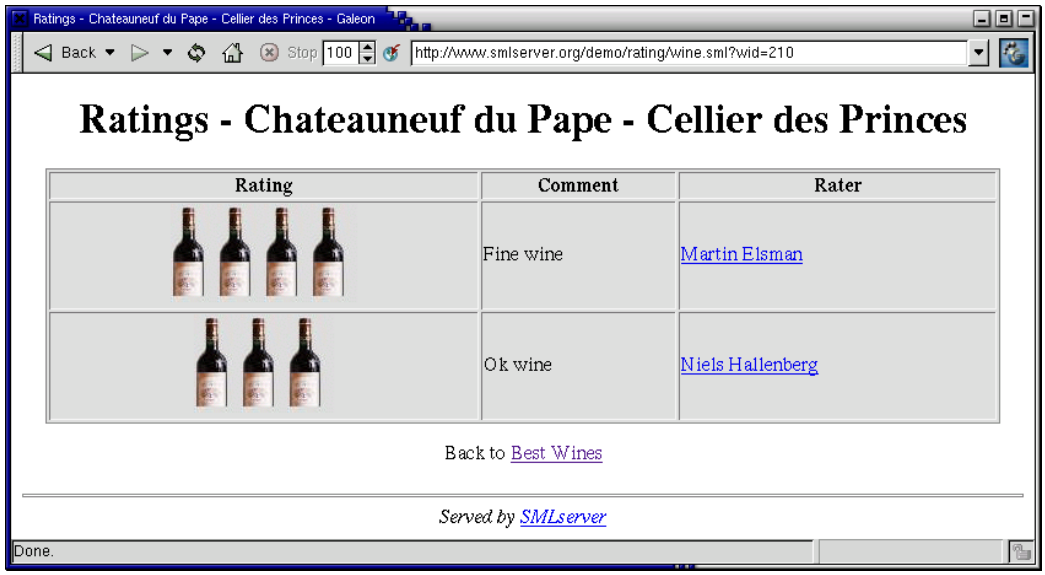


Figure 7.6: The Best Wines comment page.

```
  `<table width=95% bgcolor="#dddddd" border=1>
     <tr><th>Rating<th>Comment<th>Rater` ^^ lines ^^
  `</table>
   <p>Back to <a href=index.sml>Best Wines</a>`

val name = Db.oneField
  `select name from wine
   where wid = ^(Int.toString wid)`

val _ = RatingUtil.returnPageWithTitle
  ("Ratings - " ^ name) body
```

The result of a user requesting the script `wine.sml` with the form variable `wid` set to 1 is shown in Figure 7.6. The function `RatingUtil.mailto` is used to present the name of the raters as mailto-anchors.

The script `add.sml`

The script `add.sml` assumes either (1) the presence of a form variable `wid` or (2) the presence of form variables `name` and `year`. In case of (1), the `name` and `year` of

the wine are obtained using simple `select` commands. In case of (2), it is checked, also using a `select` command, whether a wine with the given `name` and `year` is present in the `wine` table already; if not, a new wine is inserted in the `wine` table. Thus, before a rating form is returned to the user, the wine to be rated will be present in the `wine` table. Here is the listing of the script `add.sml`:[10]

```
structure FV = FormVar
val (wid, name, year) =
  case FV.wrapOpt FV.getNatErr "wid" of
    SOME wid =>   (* get name and year *)
      let val wid = Int.toString wid
          val query =
            `select name, year from wine
             where wid = ^wid`
      in case Db.oneRow query of
        [name,year] => (wid, name, year)
      | _ => raise Fail "add.sml"
      end
  | NONE =>
      let val name = FV.wrapFail
            FV.getStringErr ("name","name of wine")
          val year = FV.wrapFail
            (FV.getIntRangeErr 1 3000)
            ("year", "year of wine")
          val year = Int.toString year
          val query = `select wid from wine
                       where name = ^(Db.qqq name)
                         and year = ^(Db.qqq year)`
      in
        case Db.zeroOrOneRow query of
          SOME [wid] => (wid, name, year)
        | _ => (* get fresh wid from RDBMS *)
            let val wid = Int.toString
                  (Db.seqNextval "wid_sequence")
                val _ = Db.dml
                  `insert into wine (wid, name, year)
                   values (^wid,
                           ^(Db.qqq name),
                           ^(Db.qqq year))`
```

[10] File `smlserver_demo/www/web/rating/add.sml`.

```
            in (wid, name, year)
            end
      end

(* return forms to the user... *)
val _ =
  RatingUtil.returnPageWithTitle
  ("Your comments to ``" ^ name ^ " - year " ^ year ^ "''")
  `<form action=add0.sml>
    <input type=hidden name=wid value=^wid>
    <textarea rows=5 cols=40 name=comment></textarea><br>
    <b>Email:</b> 
    <input type=text name=email size=30><br>
    <b>Name:</b> 
    <input type=text name=fullname size=30><br>
    <b>Rate (between 0 and 6):</b> 
    <input type=text name=rating size=2> 
    <input type=submit value="Rate it">
    <p>Back to <a href=index.sml>Best Wines</a>
   </form>`
```

A rating form for the wine "Margaux - Chateau de Lamouroux" is shown in Figure 7.7.

The script `add0.sml`

The script `add0.sml` implements the rating-insert transaction. Here is the listing of the script:[11]

```
structure FV = FormVar
val comment = FV.wrapFail FV.getStringErr
  ("comment", "comment")
val fullname = FV.wrapFail FV.getStringErr
  ("fullname", "fullname")
val email = FV.wrapFail FV.getStringErr
  ("email", "email")
val wid = Int.toString(FV.wrapFail FV.getNatErr
                       ("wid","internal number"))
val rating =
  Int.toString(FV.wrapFail (FV.getIntRangeErr 0 6)
```

[11] File `smlserver_demo/www/web/rating/add0.sml`.

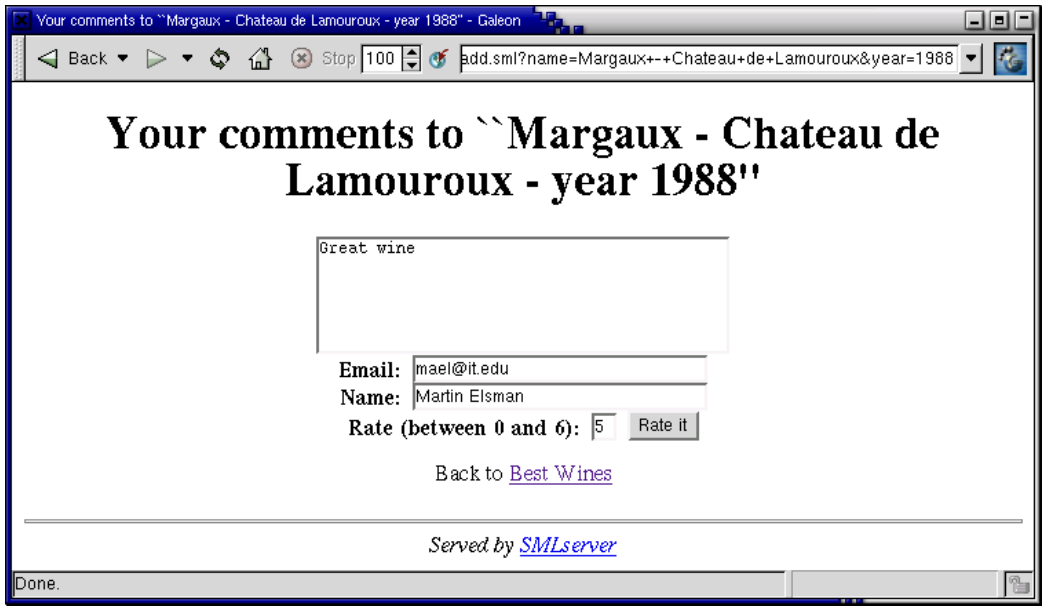


Figure 7.7: The wine rating form. Users are asked to provide ratings between 0 and 6.

```
                ("rating","rating"))

  val _ = Db.dml
    `insert into rating (wid, comments, fullname,
                         email, rating)
     values (^wid, ^(Db.qqq comment), ^(Db.qqq fullname),
             ^(Db.qqq email), ^rating)`

  val _ = Web.returnRedirect "index.sml"
```

The form variable functions provided in the `FormVar` structure are used to return error messages to the user in case a form variable is not present in the request or in case its content is ill-formed.

The function `Web.returnRedirect` is used to redirect the user to the Best Wines main page after the insert transaction is executed.

The script `index.sml`

The script `index.sml` implements the Best Wines main page. It presents the rated wines, listing the wine with the highest average rate first. Here is the script `index.sml`:[12]

```
(* the complex query that calculates the scores *)
val query =
  `select wine.wid, name, year,
          avg(rating) as average,
          count(*) as ratings
   from wine, rating
   where wine.wid = rating.wid
   group by wine.wid, name, year
   order by average desc, name, year`

fun formatRow (g, acc) =
  let val avg = g "average"
      val avgInt =
        case Int.fromString avg of
          SOME i => i
        | NONE => case Real.fromString avg of
            SOME r => floor r
          | NONE => raise Fail "Error in formatRow"
```

[12] File `smlserver_demo/www/web/rating/index.sml`.

```
        val wid = g "wid"
    in acc ^^
      `<tr><td><a href="wine.sml?wid=^wid">^(g "name")</a>
        (year ^(g "year"))
       <th> ^(RatingUtil.bottleImgs avgInt)
       <td align=center>^(g "ratings")
       <td><a href="add.sml?wid=^wid">rate it</a></tr>`
    end

  val _ = RatingUtil.returnPageWithTitle "Best Wines"
   (`<table width=95% bgcolor="#dddddd" border=1>
     <tr><th>Wine<th>Average Score (out of 6)
         <th>Ratings<th> ` ^^
    (Db.fold formatRow `` query) ^^
    `</table>
     <form action=add.sml>
     <h2>Rate new wine - type its name and year</h2>
     <b>Name:</b><input type=text name=name size=30> 
     <b>Year:</b><input type=text name=year size=4> 
     <input type=submit value="Rate it...">
     </form>`)
```

The implementation uses the function `RatingUtil.bottleImgs` to generate HTML code for showing a number of bottle images. The result of presenting the Best Wines main page to a user is shown in Figure 7.8.

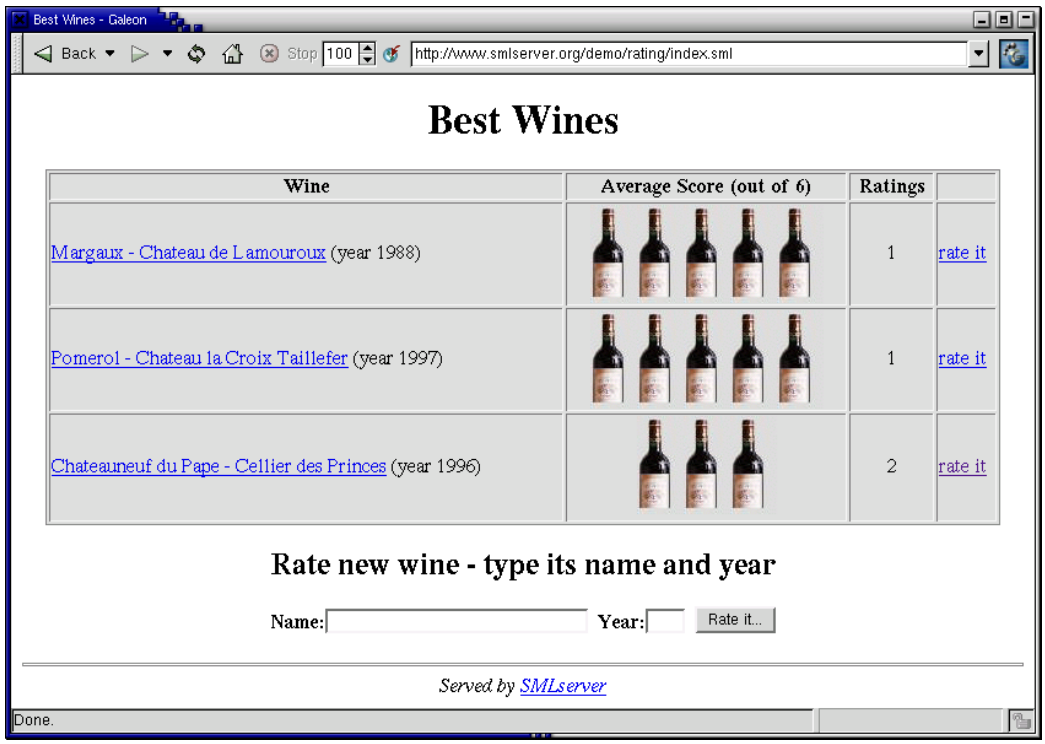


Figure 7.8: The main page for the Best Wine Web site.

Chapter 8

Checking Form Variables

Checking form variables is an important part of implementing a secure and stable Web site, but it is often a tedious job, because the same kind of code is written in all scripts that verify form variables. The `FormVar` module, which we present in this chapter, overcomes the tedious part by defining several functions, which may be used to test form variables consistently throughout a large system.

8.1 The Structure `FormVar`

The idea is to define a set of functions corresponding to each type of value used in forms. Each function is defined to access values contained in form variables of the particular type. For instance, a function is defined for accessing all possible email addresses in a form variable. In case the given form variable does not contain a valid email address, errors are accumulated and may be presented to the user when all form variables have been checked. To deal with error accumulation properly, each function takes three arguments:

1. The name of the form-variable holding the value
2. The name of the field in the form; the user may be presented with an error page with more than one error and it is important that the error message refers to a particular field in the form
3. An accumulator of type `errs`, used to hold the error messages sent back to the user

The functions are named `FormVar.get`T`Err`, where T ranges over possible form types. In each script, when all form variables have been checked using calls to particular `FormVar.get`T`Err` functions, a call to a function `FormVar.anyErrors`

```
structure FormVar :
sig
  exception FormVar of string
  type errs
  type 'a formvar_fn = string * string * errs -> 'a * errs

  val emptyErr       : errs
  val addErr         : Quot.quot * errs -> errs
  val anyErrors      : errs -> unit

  val getIntErr      : int formvar_fn
  val getNatErr      : int formvar_fn
  val getRealErr     : real formvar_fn
  val getStringErr   : string formvar_fn
  val getIntRangeErr : int -> int -> int formvar_fn
  val getEmailErr    : string formvar_fn
  val getUrlErr      : string formvar_fn
  val getEnumErr     : string list -> string formvar_fn

  val wrapOpt    : 'a formvar_fn -> (string -> 'a option)
  val wrapExn    : 'a formvar_fn -> (string -> 'a)
  val wrapFail   : 'a formvar_fn -> (string * string -> 'a)
  ...
end
```

Figure 8.1: The signature of the `FormVar` structure (excerpt).

returns an error page if any errors occurred and otherwise proceeds with the remainder of the script. If an error page is returned, the script is terminated.

An excerpt of the `FormVar` interface[1] is given in Figure 8.1. The type `formvar_fn` represents the type of functions used to check form variables. For instance, the function `getIntErr` has type `int formvar_fn`, which is identical to the type

```
string * string * errs -> int * errs
```

If it is not desirable to return an error page, the programmer may use one of the following wrapper functions to obtain appropriate behavior:

[1] File `smlserver_demo/web_demo_lib/FormVar.sml`.

Wrapper function	Description
`FormVar.wrapOpt`	Returns `SOME(`v`)` on success, where v is the form value; returns `NONE`, otherwise
`FormVar.wrapExn`	Raises exception `FormVar` on error
`FormVar.wrapFail`	On failure, a page is returned. The difference from the `get`T`Err` functions is that with `wrapFail` only one error is presented to the user

Many of the examples in this document make use of the `FormVar` wrapper functions in combination with the `get`T`Err` functions. The Currency Service described in Section 6.8 on page 53 is a good example.

8.2 Presenting Multiple Form Errors

We now turn to an example that uses the multi-error functionality of the `FormVar` structure. The example constitutes a simple email service built from two scripts, one that presents a form to the user (`mail_form.sml`) and one that sends an email constructed on the basis of the form content contributed by the user (`mail.sml`). The script `mail_form.sml` looks as follows:[2]

```
Page.return "Send an email"
‘<form action=mail.sml method=post>
  <table>
  <tr><th align=left>To:</th><td align=right>
    <input type=text name=to></td></tr>
  <tr><th align=left>From:</th><td align=right>
    <input type=text name=from></td></tr>
  <tr><th align=left>Subject:</th><td align=right>
    <input type=text name=subject></td></tr>
  <tr><td colspan=2><textarea name=body cols=40
    rows=10>Fill in...</textarea></td></tr>
  <tr><td colspan=2 align=center>
    <input type=submit value="Send Email"></td></tr>
  </table>
 </form>‘
```

The action of the form is the script `mail.sml`. When the user presses the “Send Email” submit button, the script `mail.sml` is executed with the form variables `to`, `from`, `subject`, and `body` set to the values contributed by the user. Here is the script `mail.sml`:[3]

[2] File `smlserver_demo/www/web/mail_form.sml`.

[3] File `smlserver_demo/www/web/mail.sml`.

```
structure FV = FormVar

val (to,errs)   = FV.getEmailErr ("to", "To", FV.emptyErr)
val (from,errs) = FV.getEmailErr ("from", "From", errs)
val (subj,errs) = FV.getStringErr ("subject", "Subject", errs)
val (body,errs) = FV.getStringErr ("body", "Body", errs)
val () = FV.anyErrors errs

val _ = Web.Mail.send {to=to, from=from,
                       subject=subj, body=body}

val _ = Page.return "Email has been sent"
  `Email with subject "^subject" has been sent to ^to.<p>
  <a href=mail_form.sml>Send another?</a>`
```

Notice the use of the function `anyErrors` from the `FormVar` structure; if there are no errors in the form data, execution proceeds by sending an email using the `Web.Mail.send` function and a message saying that the email has been sent is presented to the user with the `Page.return` function. Otherwise, if one or more errors were found analyzing the form data, an error page is presented to the user; the result of a user submitting the mail form with an invalid "From" field and an empty "Subject" field is shown in Figure 8.2.

For another example of using the multi-error functionality of the `FormVar` structure, see the file `smlserver_demo/www/web/formvar_chk.sml`.

8.3 Implementation

The `FormVar` structure is based on the function `Web.Conn.formvar`, which provides a more primitive way of accessing form variables submitted with a request. The function `Web.Conn.formvar` has type `string->string option` and returns the query data associated with the connection and the argument key, if available.[4]

In addition to the use of the `Web.Conn.formvar` function, the implementation of the `FormVar` structure also makes use of regular expressions (see Section 6.2).

[4] A function `Web.Conn.formvarAll` with type `string->string list` makes it possible to access all values bound to a particular form variable.

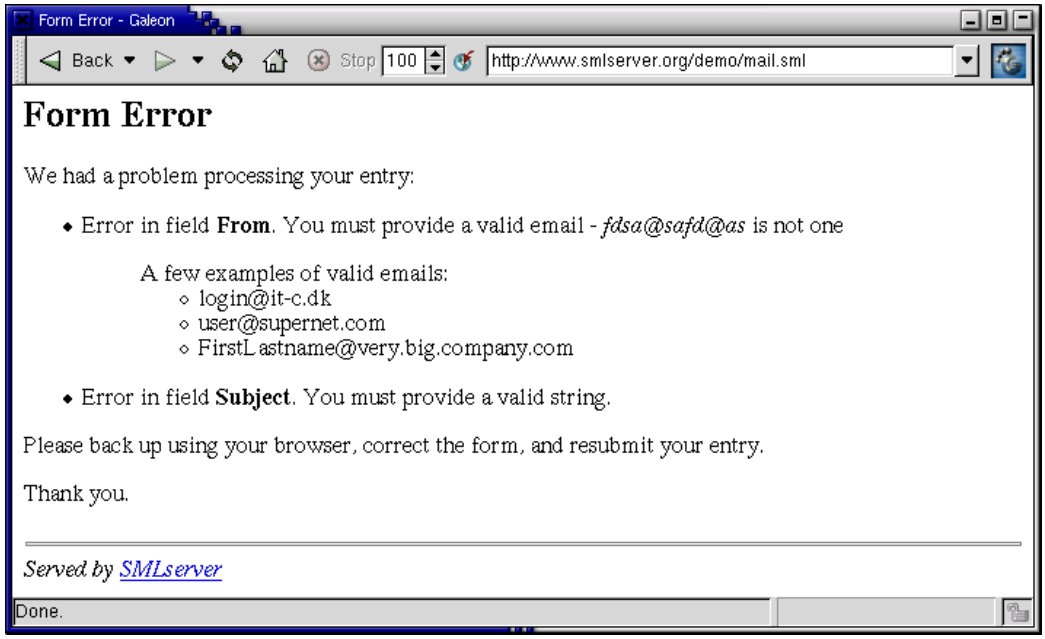


Figure 8.2: When a user submits the email form with invalid entries, such as an invalid email address and an empty subject field, the user is presented with an error page that summarizes all errors.

Chapter 9

Authentication

Dynamic Web sites often make use of an authentication mechanism that provides some form of weak identification of users. The traditional authentication mechanism allows users of a Web site to *login* to the Web site, by providing an email address (or some user name) and a password. There are several reasons for adding an authentication mechanism to a Web site:

- Access restriction. If some information is available to only some users, a mechanism is necessary to hide the restricted information from unprivileged users.

- User contributions. If users are allowed to contribute content on the Web site, it must be possible for the system to (weakly) identify the user so as to avoid spam content. Also, the user that contributes with the content, and only that user, should perhaps be allowed to modify or delete the contributed content.

- Personalization. Different users of a Web site have different needs and different preferences concerning page layout, and so on. By adding personalization to a Web site, there is a chance of satisfying more users.

- User tracking. A particular user's history on a Web site may be of great value, perhaps for an administrator to see what content the user has seen when answering questions asked by the user. For an in-depth discussion about what a user tracking system may be used for, consult [Gre99].

- User transactions. If the Web site is an e-commerce site, for instance, a secure authentication mechanism, perhaps based on SSL (Secure Socket Layer), is necessary to allow a user to perform certain transactions.

In this chapter we present a simple authentication mechanism, based on cookies (see the next section) and on a user table stored in a database. The authentication mechanism makes it possible for users to have a machine-generated password sent by email. Hereafter, users may login to the Web site using their email address and the newly obtained password. The authentication mechanism also provides functionality for users to logout, but the main feature of the authentication mechanism is a simple programmer's interface for checking whether a user is logged in or not. It is straightforward to add more sophisticated features to the authentication mechanism, such as a permission system for controlling which users may do what.

9.1 Feeding Cookies to Clients

Cookies provide a general mechanism for a Web service to store and retrieve persistent information on the client side of a connection. In response to an HTTP request, a server may include a number of cookies in the header part of the response. The cookies are installed on the client (e.g., Netscape and Internet Explorer) and are automatically sent back to the Web server in later requests to the Web service.

Although a client sends a cookie back only to the Web service that issues the cookie, one cannot count on cookies to be a secure mechanism for transferring data between a Web service and its clients. As is the case with form data, cookies are transmitted in clear text, unless some encryption mechanism, such as SSL (Secure Socket Layer), is used. There are other problems with cookies. Because they are often stored locally on client computers, other users that have access to the computer may have access to the cookie information (Windows 98). Also, most client Web browsers support only a limited number of cookies, so if a Web service sends a cookie to a browser, then it is uncertain for how long time the cookie remains on the client.

Despite the problems with cookies, it is difficult to build a useful authentication mechanism without the use of cookies. In particular, authentication mechanisms entirely based on form variables require a user to login to the Web site whenever the user visits the site. Also of importance is that authentication mechanisms entirely based on form variables require more tedious programming than when cookies are used, because authentication information is required on all links and form actions.

SMLserver implements the following Cookie interface:

```
structure Cookie :
  sig
    exception CookieError of string
    type cookiedata = {name   : string,
```

```
                        value  : string,
                        expiry : Date.date option,
                        domain : string option,
                        path   : string option,
                        secure : bool}

    val allCookies     : unit -> (string * string) list
    val getCookie      : string -> (string * string) option
    val getCookieValue : string -> string option

    val setCookie    : cookiedata -> unit
    val setCookies   : cookiedata list -> unit
    val deleteCookie : {name : string, path : string option}
                       -> string
end
```

The function `setCookie` request the cookie to be included in the header part of the HTTP response (instructing the client to store the cookie). The function takes as argument a record with cookie attributes. The `name` and `value` attributes are mandatory strings, which are URL encoded so that it is possible to include characters other than letters and digits in the strings. The function raises the exception `CookieError` if the `name` or `value` attribute contains the empty string. The function `setCookies` generalizes the `setCookie` function by taking a list of cookies as argument.

The `expiry` attribute is a date that defines the life time of the cookie. The cookie is removed from the browser when the expiration date is reached.[1] The life time of a cookie with no `expiry` attribute is the user's session only. A cookie may be removed from a client by specifying an expiration date in the past (or by using the function `deleteCookie`). To generate an expiration date that lasts in 60 seconds from the present time, the following Standard ML code may be used:

```
let open Time
in Date.fromTimeUniv(now() + fromSeconds 60)
end
```

Notice that the symbolic identifier `+` in the expression above refers to the identifier `Time.+`, which has type `Time.time * Time.time -> Time.time`.

[1] The date string format used in cookies is of the form `Wdy, DD-Mon-YYYY HH:MM:SS GMT`.

9.2 Obtaining Cookies from Clients

When a user requests a URL, the user's browser searches for cookies to include in the request. The cookie's `domain` attribute is compared against the Internet domain name of the host being requested. The cookie is included in the request if there is a tail match and a path match according the the definitions below.

A *tail match* occurs if the cookie's `domain` attribute matches the tail of the fully qualified domain name of the requested host. So for instance, a `domain` attribute "`it.edu`" matches the host names "`www.it.edu`" and "`adm.it.edu`". Only hosts within the specified domain may set a cookie for a domain and domains must have at least two periods (.) in them to prevent matching domains of the form "`.com`" and "`.edu`". The default value of the `domain` attribute is the host name of the server that generates the cookie.

A *path match* occurs if the pathname component of the requested URL matches the `path` attribute of the cookie. For example, there is a path match if the pathname component of the requested URL is `/foo/bar.html` and the cookie's `path` attribute is `/foo`. There is no path match if the pathname component of the requested URL is `index.html` and the cookie's `path` attribute is `/foo`. The default `path` attribute is the pathname component of the document being described by the header containing the cookie.

A cookie containing the `secure` attribute is transmitted on secure channels only (e.g., HTTPS requests using SSL). Without the `secure` attribute, the cookie is sent in clear text on insecure channels (e.g., HTTP requests).

The functions `allCookies`, `getCookie`, and `getCookieValue` may be used to access cookies and their values. The cookie name and value are URL decoded by the functions.

If SMLserver fails to read the cookies transmitted from a browser, the exception `CookieError` is raised. This error indicates an error on the browser side.

9.3 Cookie Example

To demonstrate the cookie interface, we present a simple cookie example consisting of three scripts `cookie.sml`, `cookie_set.sml`, and `cookie_delete.sml`.

The entry page is implemented by the `cookie.sml` script. It shows all cookies received in the header of the request and displays two forms; one for adding cookies and one for removing cookies. Figure 9.1 shows the result of a user requesting the file `cookie.sml`.

The code for listing all cookies uses the function `Web.Cookie.allCookies`:

```
val cookies =
```

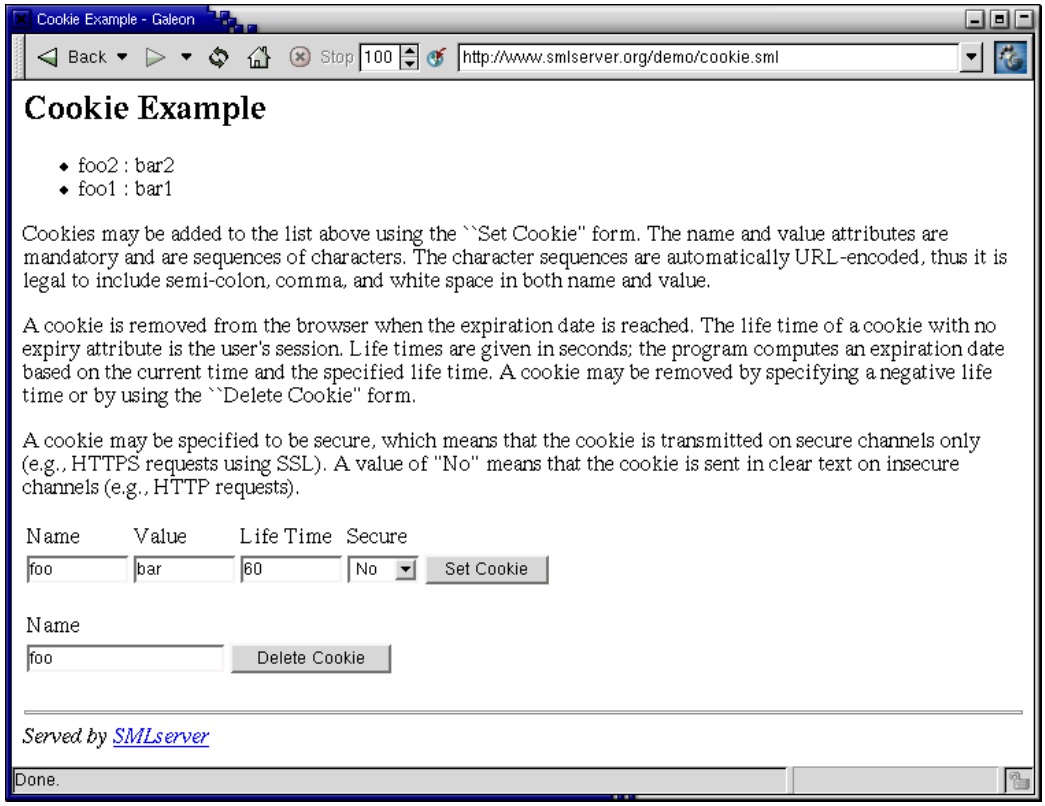


Figure 9.1: The result of a user requesting the file `cookie.sml` with two cookies `foo1` and `foo2`.

```
    foldl (fn ((n,v),a) => `<li> ^n : ^v ` ^^ a)
    `` (Web.Cookie.allCookies())
```

Notice that the use of quotations in the application of `foldl` ensures that the HTML list is built efficiently, without the use of string concatenation.

The action of the "Set Cookie" form is the script `cookie_set.sml`, which returns a redirect to the `cookie.sml` script, with a cookie included in the response header. The redirect is implemented using the function `Web.Conn.returnRedirectWithCode`:[2]

```
structure FV = FormVar

val cv = case FV.wrapOpt FV.getStringErr "cookie_value"
           of NONE => "No Cookie Value Specified"
            | SOME cv => cv

val cn = case FV.wrapOpt FV.getStringErr "cookie_name"
           of NONE => "CookieName"
            | SOME cn => cn

val clt = case FV.wrapOpt FV.getIntErr "cookie_lt"
            of NONE => 60
             | SOME clt => clt

val cs = case FV.wrapOpt FV.getStringErr "cookie_secure"
           of SOME "Yes" => true
            | _  => false

val expiry = let open Time Date
             in fromTimeUniv(now() + fromSeconds clt)
             end

val cookie = Web.Cookie.setCookie
              {name=cn, value=cv, expiry=SOME expiry,
               domain=NONE, path=SOME "/", secure=cs}

val _ = Web.Conn.returnRedirectWithCode(302, "cookie.sml")
```

The variables `cn`, `cv`, `cs`, and `clt` contain the form values received from the first entry form in the page returned by the `cookie.sml` script.

The action of the "Delete Cookie" form is the script `cookie_delete.sml`:[3]

[2] File `smlserver_demo/www/web/cookie_set.sml`.

[3] File `smlserver_demo/www/web/cookie_delete.sml`.

```
val cn =
  case FormVar.wrapOpt FormVar.getStringErr "cookie_name"
    of NONE => "CookieName"
     | SOME cn => cn

val _ = Web.Cookie.deleteCookie{name=cn,path=SOME "/"}

val _ = Web.Conn.returnRedirectWithCode(302, "cookie.sml")
```

The cookie name `cn` is the value received from the second entry form in the page returned by the `cookie.sml` script.

9.4 Storing User Information

The authentication mechanism presented below makes use of information about users stored in a `person` table in a database (see Chapter 7). The SQL for creating the `person` table looks as follows:[4]

```
create table person (
  person_id int primary key,
  password varchar(100) not null,
  email varchar(20) unique not null,
  name varchar(100) not null,
  url varchar(200)
);
```

Each person in the table is uniquely identified by a number `person_id`. Moreover, it is enforced by a consistency constraint that no two persons have the same `email` address. The `name` and `url` columns provide additional information about a user and the `password` column holds passwords that are compared to the passwords entered when users login.

An SQL sequence `person_seq` is used for creating unique `person_id` numbers, dynamically. Two people are inserted in the table by default:

```
create sequence person_seq start 3;

insert into person (person_id, password, email, name, url)
values (1, 'Martin', 'mael@it.edu', 'Martin Elsman',
        'http://www.itu.dk/~mael');
```

[4] File `smlserver_demo/web_demo_lib/pgsql/person.sql`.

```
insert into person (person_id, password, email, name, url)
values (2, 'Niels', 'nh@it.edu', 'Niels Hallenberg',
       'http://www.itu.dk/~nh');
```

Now that the table for storing user information is in place, it is possible to describe the authentication mechanism in detail.

9.5 The Authentication Mechanism

The authentication mechanism is implemented by a library structure `Auth` and a series of SMLserver scripts for managing the issuing of passwords, sending passwords to users, serving login forms to users, and so on:[5]

- `auth_form.sml`. Serves a "Login form" to users
- `auth.sml`. Processes the "Login form" submitted by a user; stores a cookie containing `person_id` and `password` (the password entered in the form, that is) on the client browser
- `auth_logout.sml`. Stores a cookie on the client browser with an expiration date in the past; redirects to a predefined index page
- `auth_new_form.sml`. Serves a "Registration form" to users, querying the user for email address, name, and home page address
- `auth_new.sml`. Processes the "Registration form" submitted by a user; creates a `password` and a unique `person_id` for the user and enters a column for the user in the `person` table; sends an email to the user with the newly created password and serves a page with instructions that an email with a password is available in the user's mail-box
- `auth_send_form.sml`. Serves a form to the user, asking for an email address
- `auth_send.sml`. Processes the form served by the `auth_send_form.sml` script; sends an email to the entered email address with the corresponding password

The three forms are shown in Figure 9.2. The library structure `Auth` provides functionality for checking whether a user is logged in (functions `verifyPerson` and `isLoggedIn`), for issuing passwords (function `newPassword`), and so on:[6]

[5] We do not present the sources for these SMLserver scripts here; the interested reader may find all sources in the directory `smlserver_demo/www/web/`.

[6] File `smlserver_demo/web_demo_lib/Auth.sml`.

Figure 9.2: The three different forms presented by the authentication mechanism. The forms correspond to the SMLserver scripts `auth_form.sml`, `auth_send_form.sml`, and `auth_new_form.sml`, respectively.

```
structure Auth :
  sig
    type person_id = int
    val loginPage    : string
    val defaultHome  : string
    val siteName     : string
    val verifyPerson : unit -> person_id option
    val isLoggedIn   : unit -> bool
    val newPassword  : int -> string
    val sendPassword : person_id -> unit
  end
```

The function `newPassword` takes as argument an integer n and generates a new password constructed from n characters chosen randomly from the character set $\{a \ldots zA \ldots Z2 \ldots 9\} \setminus \{loO\}$.

The function `sendPassword` takes a `person_id` as argument and sends an email with the user's password to the user. The three strings `loginPage`, `defaultHome`, and `siteName` are configuration strings that default to the login page provided by the authentication mechanism, the default page that the user is forwarded to once logged in, and the name of the Web site.

The function `verifyPerson` returns `SOME(`p`)` if the user (1) is logged in, and (2) is identified by the `person_id` p; otherwise the function returns `NONE`. The implementation of the function checks if cookies with the names `auth_person_id` and `auth_password` are available, and if so, proceeds by checking that the password in the database is identical with the password in the cookie. For reasons having to do with caching of passwords (Section 9.6), we define a function `verifyPerson0`, which the function `verifyPerson` calls with a function for extracting a password for a user from the database:

```
fun verifyPerson0 (getPasswd: string -> string option)
  : person_id option =
  (case (Web.Cookie.getCookieValue "auth_person_id",
         Web.Cookie.getCookieValue "auth_password")
     of (SOME person_id, SOME psw) =>
       (case getPasswd person_id
          of NONE => NONE
           | SOME db_psw =>
            if db_psw = psw then Int.fromString person_id
            else NONE
       )
      | _ => NONE
```

```
  ) handle Web.Cookie.CookieError _ => NONE

fun verifyPerson() =
  verifyPerson0 (fn p => Db.zeroOrOneField
                 `select password from person
                  where person_id = ^p`)
```

9.6 Caching Passwords for Efficiency

It is unsatisfactory that a Web site needs to query the database for password information every time a user accesses a restricted page. The solution is to use the SMLserver caching mechanism to avoid looking up passwords for users that have been accessing the Web site within the last 10 minutes (600 seconds).

To implement this idea, the function `verifyPerson` is modified as follows:

```
fun verifyPerson() =
  let fun f p =
        Db.zeroOrOneField
               `select password from person
                where person_id = ^p`
      val cache = Web.Cache.get (Web.Cache.String, Web.Cache.String,
                    "auth", Web.Cache.WhileUsed
                        (SOME (Time.fromSeconds 600), SOME 10000))
      fun g =  Web.Cache.memoizePartial cache f
  in verifyPerson0 g
  end
```

For a discussion of the function `Web.Cache.memoizePartial`, see Section 6.6.

Note that if we were to implement scripts that allow users to modify their passwords, we would, of course, need to flush the cache appropriately when users modify their passwords. This is done by the function `Web.Cache.flush`, presented in Section 6.6 on page 48.

9.7 Applying the Authentication Mechanism

We shall now see how a Web site may apply the authentication mechanism to restrict the transactions and content available to a particular user. The example application that we present serves as a link database to keep track of Web sites developed with SMLserver. The idea is that all visitors of the Web site have access to browse the list of Web sites submitted by SMLserver users. At the same time,

only registered users can add new Web sites to the list or delete entries that they have previously entered.

The first step in the design is to define a data model that extends the data model for the authentication mechanism (the `person` table). The following definition of the table `link` serves the purpose:[7]

```
create table link (
  link_id int primary key,
  person_id int references person not null,
  url varchar(200) not null,
  text varchar(200)
);
```

Each link in the table is identified with a unique `link_id` and each link is associated with a person in the `person` table. The two columns `url` and `text` constitute the link information provided by a user.

The next step in the development is to define a Web site diagram for the link database Web site. Such a Web site diagram is pictured in Figure 9.3, which also pictures the scripts for the authentication mechanism. The figure shows a diagram with all SMLserver scripts for the Web site. Scripts that present forms are pictured as boxes whereas scripts that function as transactions on the database (or have other effects, such as sending emails) are pictured by their name. As a side remark, we add that a user should have access to delete only those Web site entries that the particular user has added.

Now that the Web site diagram for the link database is in place, we are ready to provide implementations for the scripts in the diagram. In the following, we present two of the involved scripts, `link/index.sml`, which shows user-submitted links, and `link/delete.sml`, which deletes a link submitted by the user.[8] The script `link/index.sml`, which is the most involved of the scripts, is implemented as follows:[9]

```
val person = Auth.verifyPerson()

val query =
  `select person.person_id, person.name, link_id,
          person.url as purl, link.url, link.text
     from person, link
    where person.person_id = link.person_id`
```

[7] File `smlserver_demo/web_demo_lib/pgsql/link.sql`.

[8] The directory `smlserver_demo/www/web/link/` holds all involved scripts.

[9] File `smlserver_demo/www/web/link/index.sml`.

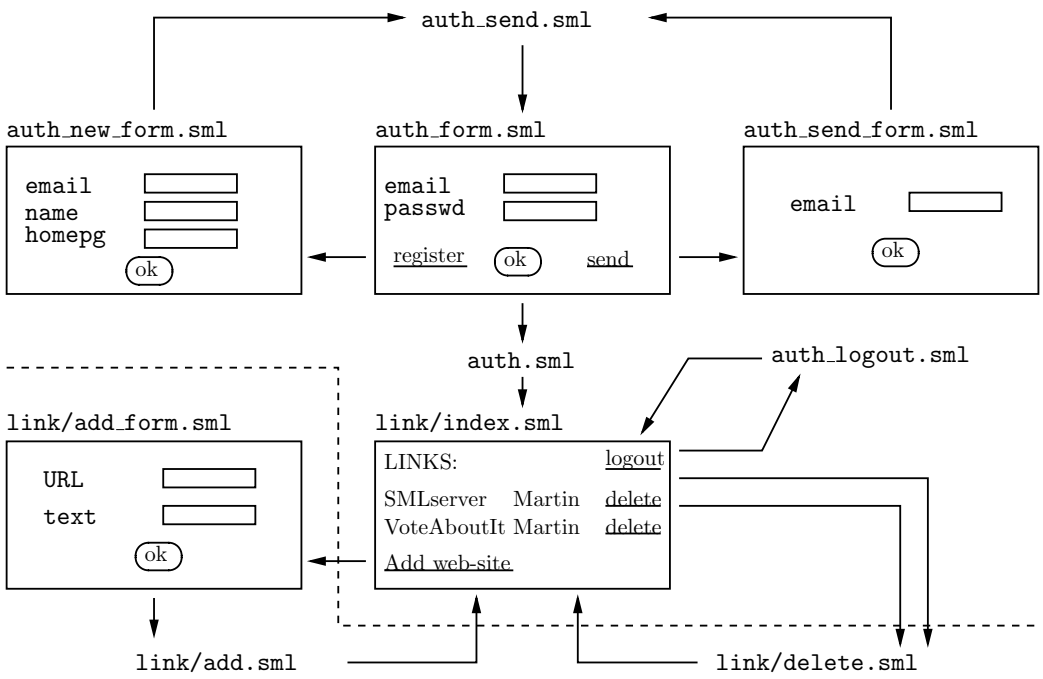


Figure 9.3: Web site diagram for the link database. SMLserver scripts pictured under the dashed line are restricted to users that are logged in; the other SMLserver scripts are accessible for all visitors.

```
fun delete g =
  if Int.fromString (g "person_id") = person
    then
      ‘ <a href=delete.sml?link_id=^(g "link_id")>delete</a>‘
  else ‘‘

fun layoutRow (g, acc) =
  ‘<li><table width=100% cellspacing=0 cellpadding=0
              border=0><tr>
     <td width=50%><a href="^(g "url")">^(g "text")</a>
     <td>added by <a href="^(g "purl")">^(g "name")</a>
     <td align=right>‘ ^^ delete g ^^
  ‘</tr></table>‘ ^^ acc

val loginout =
  case person
    of NONE =>
      ‘To manage links that you have entered, please
       <a href=../auth_form.sml?target=link/>login</a>.‘
    | SOME p =>
      let val name = Db.oneField
            ‘select name from person
             where person_id = ^(Int.toString p)‘
      in ‘You are logged in as user ^name - you may
          <a href=../auth_logout.sml>logout</a>.‘
      end

val list = Db.fold layoutRow ‘‘ query

val _ =
  Page.return "Web sites that use SMLserver"
  (loginout ^^ ‘<ul>‘ ^^ list ^^
   ‘<p><li><a href=add_form.sml>Add Web site</a></ul>‘)
```

The script uses the function `Auth.verifyPerson` to present delete links for those Web site entries that a user is responsible for. Moreover, if a user is already logged in, a “Logout” button is presented to the user, whereas a “Login” button is presented if the user is not logged in. The result of a user requesting the file is shown in Figure 9.4.

The script `link/delete.sml` is implemented by the following Standard ML

Figure 9.4: The result of a user requesting the file `link/index.sml`.

code:[10]

```
val person_id =
  case Auth.verifyPerson()
    of SOME p => p
     | NONE => (Web.returnRedirect Auth.loginPage
                ; Web.exit())

val link_id = FormVar.wrapFail
  FormVar.getNatErr ("link_id", "Link id")

val delete =
  `delete from link
   where person_id = ^(Int.toString person_id)
     and link_id = ^(Int.toString link_id)`

val _ = Db.dml delete
val _ = Web.returnRedirect "index.sml"
```

Notice that users that are not logged in, but somehow request the file, are redirected to the default login page provided in the `Auth` structure. Also notice that a user can delete only those links that the user is responsible for.

[10] File `smlserver_demo/www/web/link/delete.sml`.

Chapter 10

Scheduling and Trapping

SMLserver supports three ways of executing scripts besides execution of scripts based on requests made by clients, namely execution of scheduled scripts, execution of trapping scripts and execution of initialization scripts.

It is possible to *schedule* a script to be executed periodically. This feature can be used to ensure that some state in the server or database is kept up to date. It can also be used for periodic checks on external ressources (e.g., that an external Web site is accessible; this is also known as an uptime monitor; see `http://eveander.com/arsdigita/free-tools/uptime.html`).

It is possible to *trap* requests for specific files in SMLserver, in such a way that a specific trap script is executed instead of the requested file. This feature can be used to control access to Web content.

In order to setup scheduled scripts and other configuration parameters, it is possible to execute a specific SML script at server initialization time.

When executing the initialization file, there is no connection to a client. It is therefore not possible to use functions that depend on a connection. For instance, one cannot use the function `Web.Conn.return`, as described on page 132. Functions that fails in this way raises the exception `Web.MissingConnection`.

10.1 Initialization

To have an initialization script executed at boot time, follow the steps below:

1. Write an initialization script. We have provided one in the `web_sys` directory `~/web/web_sys/init.sml`. See Chapter 2.

2. Include the initialization script in your `mlb`-file. We have included the initialization script `~/web/web_sys/init.sml` in the `mlb`-file `web.mlb`. See Chapter 2.

3. Tell Apache to execute the initialization script at initialization time. This is done by adding the following to the `<IfModule mod_sml.c>` Apache configuration entry:

```
# Initialization script to schedule script execution and
# register trapping of URL requests; remember to include
# the initialization script in the project file.
SmlInitScript "../web_sys/init.sml"
```

   See also Chapter 2 on configuration of Apache for use with SMLserver.

To check that it works, restart Apache as described in Chapter 2. Then consult the log file (`/var/log/apache2/error.log`) and look for the text `executing init.sml...`.

10.2 Scheduling

The signature `WEB` specifies the following functions to control scheduling of script execution:

```
signature WEB =
  sig
    ...
    val schedule       : string -> string option -> Date.date
                         -> Time.time -> unit
    val deSchedule     : string -> unit
    val scheduleScript : string -> string option -> int -> unit
    val scheduleDaily  : string -> string option
                         -> {hour:int, minute:int} -> unit
    val scheduleWeekly : string -> string option
                         -> {day:Date.weekday, hour:int, minute:int}
                         -> unit
    ...
end
```

After an application `schedule f s d i`, the script determined by the file `f` is scheduled to execute the first time at date and time `d` with interval `i` using `s` as host. If `s` is not supplied then `localhost` is used as host. Usually, calls to the schedule function appears in the initialization script (e.g., `init.sml`).

After an application `scheduleScript f s d`, the script determined by the location on the server `f` is scheduled to execute every `d` seconds using `s` as host.

The `s` is interpreted like in `schedule`. Usually, calls to the scheduleScript function appears in the initialization script (e.g., `init.sml`).

After an application `scheduleDaily f s {hour,minute}`, the script determined by the location on the server `f` is scheduled to execute every day at the specified time (`hour` and `minute`). The hour and minute are interpreted modulo 24 and 60. The `s` is interpreted like in `schedule`.

After an application `scheduleWeekly f s {day,hour,minute}`, the script determined by the location on the server `f` is scheduled to execute every week at the specified time (`day`, `hour`, and `minute`). The hour and minute are interpreted modulo 24 and 60. The `s` is interpreted like in `schedule`.

After the application `deSchedule f`, any scheduling of the script determined by the location on the server `f` is canceled.

We have included a small script `log_time.sml` that logs the time of day in the `/var/log/apache2/error.log` file:[1]

```
val time_of_day =
  Date.fmt "%H.%M.%S" (Date.fromTimeLocal(Time.now()))

val _ = Web.log(Web.Notice, "time_of_day: " ^ time_of_day)
```

You can do the following experiment (if you installed SMLserver as described in Chapter 2 then it should work out of the box):

1. Make sure that `log_time.sml` appears in your project file (e.g., `web.mlb`).

2. Insert the following SML code in the file `init.sml`:

```
val time_of_day =
  Date.fmt "%H.%M.%S" (Date.fromTimeLocal(Time.now()))

  val _ = Web.log(Web.Debug, "time of day: " ^ time_of_day)
  val _ = Page.return "Time of day" (‘
     <body bgcolor=white>
      The time of day is ‘  ^^
      Quot.fromString time_of_day ^^ ‘.‘)
```

The script `log_time.sml` is executed every 10 minutes.

[1] File `smlserver_demo//www/web/log_time.sml`.

3. Make sure that the script `init.sml` is executed at boot time, that is, try restart Apache and check the `error_log` file. If it does not work, then check that `init.sml` appears in your project file, and make sure that `init.sml` appears in the Apache configuration file.

It works when you see something similar to the following in your `error_log` file:

```
[Wed Dec 07 18:04:44 2005] [debug] mod_smllib.c(172): [client 127.0.0.1]
  time of day: 18.04.44
```

10.3 Trapping

Trapping is setup using appropriate Apache modules. These are configured in the Apache configuration file and SMLserver is almost unaware of such mechanisms.

The `url` function provides the url from each internal redirect in a list where the current url is in front and the initial url is in the back. In the following example the module *mod_rewrite* is used to redirect all request from a directory to a particular script.

From the Apache configuration:

```
DocumentRoot "/home/user/web/www"
<Directory /home/user/web/www/secret>
  RewriteEngine On
  RewriteBase /secret
  RewriteRule .* server.sml
</Directory>
```

Here requests to anything in `/secret/` is redirected to `/secret/server.sml`

Chapter 11

Configuration

Configuration of SMLserver is divided into two parts; configuration of Apache and configuration of the library and your application. By keeping these concerns apart, the configuration of your application does not clutter up the configuration of Apache and vice versa. Thus the impact of the Apache configuration file on your project is kept at a minimum.

In the main Apache configuration entry for SMLserver, settings for `SmlPrjId` and `SmlPath` must be provided. A setting for `SmlInitScript` is optional.

`SmlPrjId:` The name of the project file (e.g. `web`).

`SmlPath:` The path to the project files (e.g. `/home/user/web/www/`).

`SmlInitScript:` A script to be run at Apache initialization time. This file typically handles configuration of your application and any other administrative tasks you would like (e.g. `/home/user/web/www/../sys/init.sml`).

SMLserver provides two functions for supporting configuration of your application, as described in the information structure in the library (`Web.Info`):

```
val configGetValue : ('a Type.Type * string) -> 'a option
val configSetValue : ('a Type.Type * string * 'a) -> unit
```

The `configSetValue` function lets you map a string to a value of the type given in the first argument. `configGetValue` lets you retrieve the value from the given string. As the first argument to `configGetValue` you must provide the same type as you gave `configSetValue`. A runtime check asserts this, and raises `Domain` if violated. You may only call `configSetValue` in your initialization script. If this rule is violated, the exception `Web.Forbidden` is raised.

With this scheme every change to the initialization script (your configuration) requires a recompilation of the project to take effect. If this is not appropriate you

can create you own configuration file and parse it in the initialization script.[1] We feel that this gives you a maximum of flexibility.

Example of use:

```
init.sml:   val _ = Web.Info.configSetValue
               (Web.Info.Type.Int, "Number of users",10)

myproject.sml:   val numberOfUsers = getOpt(Info.configGetValue(
           Web.Info.Type.Int, "Number of users"),5)
```

[1] You have to program this yourself.

Chapter 12

Summary

This book provides a tutorial overview of programming dynamic Web applications with SMLserver through the presentation of a series of examples. Starting with the basic mechanism for serving dynamic pages to users, the book covers topics such as achieving and validating data from users, fetching data from foreign Web sites, interfacing to Relational Database Management Systems (RDBMSs), and authenticating users.

SMLserver is already used for a series of real-purpose Web sites, including an evaluation system, an alumni system, and a course registration system for the IT University of Copenhagen.

Experience with SMLserver demonstrates that the strict type system of Standard ML combined with its advanced language features, such as modules and higher-order functions, ease maintainability and extensibility. If used properly, the advanced language features make separation of code from presentation straightforward and increase reusability of code.

Although it is possible to create large Web sites with SMLserver, there are currently a few features missing, which we plan to add to SMLserver soon. Among the features missing are support for XML and XSLT translations, support for the Oracle clob datatype and a SOAP interface.

We have implemented several improvements to SMLserver since the previous publication of this book, including a typed cache interface and caching of region pages which speeds up script execution considerably.

We have also implemented the possibility of periodic execution of scripts using the SMLserver API. Similarly, we have added support for an initialization file to be executed at server start up. We have also implemented a simple trap mechanism for supporting filtering.

For technical issues concerning SMLserver, see also [EL04], [EH03], and [EH02].

Bibliography

[EH02] Martin Elsman and Niels Hallenberg. A region-based abstract machine for the ML Kit. Technical Report TR-2002-18, Royal Veterinary and Agricultural University of Denmark and IT University of Copenhagen, August 2002. IT University Technical Report Series.

[EH03] Martin Elsman and Niels Hallenberg. Web programming with SMLserver. In *Fifth International Symposium on Practical Aspects of Declarative Languages (PADL'03)*. Springer-Verlag, January 2003.

[EL04] Martin Elsman and Ken Friis Larsen. Typing XHTML Web applications in ML. In *International Symposium on Practical Aspects of Declarative Languages (PADL'04)*. Springer-Verlag, June 2004.

[FP02] Matthew Fluet and Riccardo Pucella. Phantom types and subtyping. In *Second IFIP International Conference on Theoretical Computer Science (TCS'02)*, pages 448–460, August 2002.

[Gre99] Philip Greenspun. *Philip and Alex's Guide to Web Publishing*. Morgan Kaufmann, May 1999. 596 pages. ISBN: 1558605347.

[HR99] Michael R. Hansen and Hans Rischel. *Introduction to Programming using SML*. Addison-Wesley, 1999. ISBN 0-201-39820-6.

[MTHM97] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

[Pau96] Lawrence C Paulson. *ML for the Working Programmer (2nd Edition, ML97)*. Cambridge University Press, 1996. ISBN 0-521-56543-X (paperback), 0-521-57050-6 (hardback).

[TBE+01] Mads Tofte, Lars Birkedal, Martin Elsman, Niels Hallenberg, Tommy Højfeld Olesen, and Peter Sestoft. Programming with regions in the ML Kit (for version 4). Technical report, IT University

of Copenhagen and Royal Veterinary and Agricultural University of Denmark, September 2001.

Appendix A

HTML Reference

An HTML *start tag* is a name included in angle-brackets like `<name>`, perhaps decorated with *attributes* as in `<name attr=arg>`. An HTML *end tag* is a name included in angle-brackets and pre-fixed with a slash (`/`) as in `</name>`. An HTML *element* is either some text not including `<` and `>`, a start tag, or a start tag and an end tag, with the same name, surrounding another HTML element. Examples of HTML elements include

- `<title> A small element </title>`
- `<a href="index.sml"> some text </a>`

An HTML document is composed of a single element `<html> ... </html>` composed of `head` and `body` elements as follows:

```
<html>
  <head> ... </head>
  <body> ... </body>
</html>
```

For compatibility with older HTML documents, the `<html>`, `<head>`, and `<body>` tags are optional.

A `head` element may include a `title` element—other element types are supported as well:

```
<title> ... </title>
```

The `title` element specifies a document title. Notice that the title does not appear on the document. Instead it may appear in a window bar identifying the contents of the window. The `title` element is also what is used as the title of the document when it is bookmarked in a browser.

A.1 Elements Supported Inside Body Element

The following sections describe elements that may be used inside the `body` element of a document.

A.1.1 Text Elements

`<p>`
: Start a new paragraph.

`<pre> ... </pre>`
: Encloses preformatted text to be displayed as is. Preformatted text may include embedded tags, but not all tag types are permitted.

`<listing> ... </listing>`
: Example computer listing; embedded tags are shown as is and tabs work.

`<blockquote> ... </blockquote>`
: Include a section of quoted text.

A.1.2 Uniform Resource Locators

A Uniform Resource Locator (URL) is of the form

resourceType`:`*additionalInformation*

where *resourceType* may be `file`, `http`, `telnet`, or `ftp` (other resource types exist as well). Each resource type relates to a specific server type, each of which performs a unique function and thus requires different *additionalInformation*. For example, URLs with resource type `http` are of the form

`http://`*host.domain*`:`*port*`/`*pathname*

The colon followed by a TCP port number is optional, and is used when a server is listening on a non-standard port; the standard port for HTTP is port 80.

A.1.3 Anchors and Hyperlinks

An *anchor* specifies a location in a document. A *hyperlink* may be used to refer to a location in a document or to an entire document.

`<a name="`*anchorName*`">`
: Specify a location *anchorName* in a document.

`<a href="#anchorName"> ... </a>`
Link to location *anchorName* in the present document.

`<a href="URL#anchorName"> ... </a>`
Link to location *anchorName* in document specified by *URL*.

`<a href="URL"> ... </a>`
Link to file or resource specified by *URL*.

`<a href="URL?n1=v1&...&nn=vn"> ... </a>`
Link to file or resource *URL* with form variable arguments `n1=v1 ... nn=nv`, separated by `&`.

To be precise, the *anchorName* and form variable arguments included in the `name` and `href` attributes in the examples above are part of the URL.

A.1.4 Headers

`<h1> ... </h1>`	**Highest significant header**
`<h2> ... </h2>`	
`<h3> ... </h3>`	
`<h4> ... </h4>`	
`<h5> ... </h5>`	
`<h6> ... </h6>`	**Lowest significant header**

A.1.5 Logical Styles

`<em> ... </em>`	Emphasis
`<strong> ... </strong>`	Strong emphasis

A.1.6 Physical Styles

`<b> ... </b>`	**Boldface**
`<i> ... </i>`	*Italics*
`<u> ... </u>`	Underline
`<tt> ... </tt>`	`Typewriter font`

A.1.7 Definition Lists

```
<dl>
<dt> First term
<dd> Definition of first term
<dt> Next term
```

```
<dd> Definition of next term
</dl>
```

The `<dl>` attribute `compact`, which takes no argument, can be used to generate a definition list that uses less space.

A.1.8 Unordered Lists

```
<ul>
<li> First item in list
<li> Next item in list
</ul>
```

A.1.9 Ordered Lists

```
<ol>
<li> First item in list
<li> Next item in list
</ol>
```

A.1.10 Characters

`&keyword;`
: Display a particular character identified by a special keyword. For example the entity `&amp;` specifies the ampersand (&), and the entity `&lt;` specifies the less than (<) character. Notice that the semicolon following the keyword is required. A complete listing of possible keywords are available from `http://www.w3.org`.

`&#ascii;`
: Display a character using its ascii code. The semicolon following the ASCII numeric value is required.

A.2 HTML Forms

HTML forms allow documents to contain forms to be filled out by users. An HTML form element looks like this: `<form> ... </form>`.

Inside a `form` element, the following four elements are allowed—in addition to other HTML elements:

- `<input>`

- `<select> ... </select>`
- `<option>`
- `<textarea> ... </textarea>`

A document may contain multiple `form` elements, but `form` elements may not be nested. Attributes to the `form` elements include:

`action="`*URL*`":`
: Specifies the location of the program to process the form.

`method="`*dataExchangeMethod*`"`
: The method chosen to exchange data between the client and the program to process the form: The most important methods are `GET` and `POST` (see Section 3.1).

A.2.1 Input Fields

An `input` element `<input type="`*inputType*`">`, which has no associated ending tag, specifies that the user may enter information in the form. The attribute `type` is required in `input` elements. In most cases, an input field assigns a value to a variable with a specified name and a specified input type. Some possible input types are listed in the following table:

inputType	**Description**
`text`	Text field; `size` attribute may be used to specify length of field.
`password`	As `text`, but stars are shown instead of the text that the user enters.
`checkbox`	Allows user to select zero or more options.
`radio`	Allows user to choose between a number of options.
`submit`	Shows a button that sends the completed form to the server specified by the attribute `action` in the enclosing `form` element.
`reset`	Shows a button that resets the form variables to their default values.
`hidden`	Defines a hidden input field whose value is sent along with the other form values when the form is submitted. This input type is used to pass state information from one Web script to another.

Additional attributes to the `input` element include:

`name="`*`Name`*`"`
: where *Name* is a symbolic name identifying the input variable.

`value="`*`Value`*`"`
: where the meaning of *Value* depends on the argument for `type`.

  For `type="text"` or `type="password"`, *Value* is the default value for the input variable. Password values are not shown on the user's form. Anything entered by the user replaces any default value defined with this attribute. For `type="checkbox"` or `type="radio"`, *Value* is the value that is submitted to the server if that checkbox is selected. For `type="reset"` or `type="submit"`, *Value* is a label that appears on the submit or reset button in place of the words "Submit" and "Reset".

`checked` (no arguments)
: For `type="checkbox"` or `type="radio"`, if `checked` is present, the input field is selected by default.

`size="`*`Width`*`"`
: where *Width* is an integer value representing the number of characters displayed for the `type="text"` or `type="password"` input field.

`maxlength="`*`Length`*`"`
: where *Length* is the maximum number of characters allowed within `type="text"` or `type="password"` variable values. This attribute is used only in combination with the input types `text` and `password`.

A.2.2 Select Elements

The `select` element `<select> ... </select>` allows a user to select between a number of options. The `select` element requires an `option` element for each item in the list (see below). Attributes and corresponding arguments include:

`name="`*`Name`*`"`
: where *Name* is the symbolic identifier for the select element.

`size="`*`listLength`*`"`
: where *listLength* is an integer representing the maximum number of `option` items displayed at one time.

`multiple` (no arguments)
: If present, more than one `option` value may be selected.

A.2.3 Select Element Options

Within the `select` element, `option` elements are used to define the possible values for the enclosing `select` element. If the attribute `selected` is present then the option value is selected by default. In the following example all three options may be chosen but `Standard ML` is selected by default.

```
<select multiple>
  <option>Haskell
  <option selected>Standard ML
  <option>C
</select>
```

A.2.4 Text Areas

A text area of the form

```
<textarea> default text </textarea>
```

defines a rectangular field where the user may enter text data. If "`default text`" is present it is displayed when the field appears. Otherwise the field is blank. Attributes and corresponding values include:

`name="`*Name*`"`
: where *Name* is a symbolic name that identifies the form variable associated with the `<textarea>`.

`rows="`*numRows*`" and cols="`*numCols*`"`
: Both attributes take an integer value which represents the number of rows and number of columns in the text area.

A.3 Miscellaneous

`<!-- text -->`
: Place a comment in the HTML source.

`<address> ... </address>`
: Present address information.

`<img src="`*URL*`" alt="Alternate Text">`
: Embed an image in the document. Attributes:

  `src:` Specifies the location *URL* of the image.

`alt:` Allows a text string to be put in place of the image in clients that cannot display images.

`align:` Specifies a relationship to surrounding text. The argument for align can be one of `top`, `middle`, or `bottom`.

`border=0:` Leaves out the border on the image *img* when it appears within `<a href=`*img*`>...</a>`.

`<br>`

Forces a line break immediately and retains the same style.

`<hr>`

Places a horizontal rule or separator between sections of text.

Appendix B

The `Web` Structure

The `Web` structure gives access to the Web server API.

The structure `Web` implements the `WEB` signature, which holds often used functionality and sub-structures matching a series of signatures. Reading signatures is a good way of familiarizing yourself with an API.

In the sections to follow, we present the following signatures:

`WEB` **(Page 125):** Main Web API.

`WEB_CACHE` **(Page 129):** Support for caching of computations.

`WEB_CONN` **(Page 132):** Connection-dependent functionality.

`WEB_COOKIE` **(Page 135):** Support for setting and retrieving cookie information.

`WEB_DB` **(Page 137):** Database connectivity.

`WEB_DB_HANDLE` **(Page 142):** Handle-based database connectivity for transactions and multiple simultaneous connections.

`WEB_DYNLIB` **(Page 146):** Support for dynamic linking.

`WEB_INFO` **(Page 147):** Support for retrieving various kinds of connection and server information.

`WEB_LOG` **(Page 149):** Support for writing to the Apache log file.

`WEB_LOW_MAIL` **(Page 150):** Low-level mail API.

`WEB_MAIL` **(Page 152):** High-level mail API.

`WEB_MIME` **(Page 153):** Support for determining the MIME type of a file.

`WEB_SERIALIZE` **(Page 154):** Type constructor specifications for type-indexed serialization; used for caching, configurations, etc.

`WEB_SET` **(Page 156):** Support for key-value associations.

B.1 The WEB Signature

```
signature WEB = sig
  include WEB_LOG
  exception MissingConnection
  exception Forbidden

  type quot = Quot.quot

  val return         : quot -> unit
  val write          : quot -> unit
  val returnRedirect : string -> unit
  val encodeUrl      : string -> string
  val decodeUrl      : string -> string
  val returnFileMime : string -> string -> unit
  val returnFile     : string -> unit
  val fetchUrl       : string -> string option
  val fetchUrlTime   : int -> string -> string option
  val buildUrl       : string -> (string * string) list -> string

  val schedule       : string -> string option -> Date.date ->
                       Time.time -> unit
  val deSchedule     : string -> unit
  val scheduleScript : string -> string option -> int -> unit
  val scheduleDaily  : string -> string option ->
                       {hour:int, minute:int} -> unit
  val scheduleWeekly : string -> string option ->
                       {day:Date.weekday, hour:int, minute:int} ->
                       unit

  val exit           : unit -> 'a

  structure Set      : WEB_SET
  structure Conn     : WEB_CONN where type set = Set.set
  structure Cookie   : WEB_COOKIE
  structure Info     : WEB_INFO
  structure Mail     : WEB_MAIL
  structure Cache    : WEB_CACHE
  structure Mime     : WEB_MIME
  structure LowMail  : WEB_LOW_MAIL
```

```
  structure DbOraBackend : WEB_DB_BACKEND where
                           type 'a Type = 'a Info.Type.Type
  structure DbMySqlBackend : WEB_DB_BACKEND where
                             type 'a Type = 'a Info.Type.Type
  structure DbPostgreSQLBackend : WEB_DB_BACKEND where
                                  type 'a Type = 'a Info.Type.Type

  structure WebDynlib : WEB_DYNLIB

  structure XMLrpc : XMLRPC
end

(*
 [MissingConnection] exception raised by functions that cannot be
 called when no connection is present (e.g., at initialization time).

 [Forbidden] exception raised by some functions on illegal input.

 [quot] type of quotations.

 [return s] sends HTML string s with status code 200 to
 client, including HTTP headers. May raise MissingConnection.

 [write s] sends string s to client, excluding HTTP headers.
 May raise MissingConnection.

 [returnRedirect loc] sends redirection HTTP response to
 client (status code 302), with information that the client
 should request location loc. May raise MissingConnection.

 [encodeUrl s] returns an encoded version of the argument s as
 URL query data. All characters except the alphanumerics are
 encoded as specified in RFC1738, Uniform Resource Locators.
 This function can be used to append arguments to a URL as
 query data following a `?'.

 [decodeUrl s] decodes data s that was encoded as URL query
 data. The decoded data is returned.

 [returnFileMime mimetype file] returns the entire contents of the
```

```
given file to the client. In addition to setting the HTTP status
response line to 200 and the Content-Type header from the given
parameter, the function also uses the stat system call to generate
the appropriate Last-Modified and Content-Length headers. May raise
MissingConnection or Fail(msg) if file cannot be accessed.

[returnFile file] as returnFileMime, but gets the
Content-Type (mimetype) argument from calling the function
Web.Mime.getMime with the given file as parameter.

[fetchUrl u] fetches a remote URL u; connects the Web server
to another HTTP Web server and requests the specified URL.
The URL must be fully qualified. Currently, the function
cannot handle redirects or requests for any protocol except
HTTP. Returns NONE if no page is found.

[fetchUrlTime u] as fetchUrl but with a specified timeout in
seconds.

[buildUrl u l] constructs a link to the URL u with the form
variable pairs l appended to u?, delimited by &, and with the
form values URL encoded.

[schedule s serv d t] schedule a script s to be executed on server
serv on date d at time t. If serv is NONE localhost is used as
server.

[deSchedule s] Unschedule the script s from execution.

[scheduleScript s serv d] after a call to this function, the script
determined by the file s on server serv is scheduled to execute every d
seconds. Usually, calls to the scheduleScript function appears in the
initialization script ../web_sys/init.sml to setup scheduled
execution. If serv is NONE localhost is used as server.

[scheduleDaily s serv {hour,minute}] after a call to this
function, the script determined by the file s on server serv is
scheduled to execute every day at the specified time (hour and
minute). The hour can be an integer from 0 to 23, and the minute an
integer from 0 to 59. If serv is NONE localhost is used as server.
```

```
 [scheduleWeekly s serv {day,hour,minute}] after a call to this
 function, the script determined by the file s on server serv is
 scheduled to execute every week at the specified time (day, hour, and
 minute). The day can be an integer from 0 to 6, where 0 represents
 Sunday. The hour can be an integer from 0 to 23, and the minute an
 integer from 0 to 59. If serv is NONE localhost is used as server.

 [exit()] terminates the script by raising the exception
 Interrupt, which is silently caught by the SMLserver module. Other
 uncaught exceptions are logged in the log file.

*)
```

B.2 The WEB_CACHE Signature

```
signature WEB_CACHE = sig
  (* Cache kinds *)
  datatype kind =
     WhileUsed of Time.time option * int option
   | TimeOut of Time.time option * int option

  (* Cache Type *)
  type ('a,'b) cache
  include WEB_SERIALIZE
  type name = string

  (* Get or create a cache *)
  val get : 'a Type * 'b Type * name * kind -> ('a,'b) cache

  (* Entries in a cache *)
  val lookup : ('a,'b) cache -> 'a -> 'b option
  val insert : ('a,'b) cache * 'a * 'b * Time.time option -> bool
  val flush  : ('a,'b) cache -> unit

  (* Memoization *)
  val memoize  : ('a,'b) cache -> ('a -> 'b) -> 'a -> 'b
  val memoizeTime  : ('a,'b) cache ->
                     ('a -> ('b * Time.time option))
                     -> 'a -> 'b
  val memoizePartial : ('a,'b) cache ->
                       ('a -> 'b option) -> 'a -> 'b option
  val memoizePartialTime  : ('a,'b) cache ->
                            ('a -> ('b * Time.time option) option) ->
                            'a -> 'b option
  (* Cache info *)
  val pp_type  : 'a Type -> string
  val pp_cache : ('a,'b) cache -> string
end

(*
 [kind] abstract type for cache kind. A cache kind describes
 the strategy used by the cache to insert and emit cache
 entries. The following strategies are supported:
```

```
  * WhileUsed (t,sz) : elements are emitted from the cache after
    approximately t time after the last use. The cache has a
    maximum size of sz bytes. Elements are emitted as needed in
    order to store new elements.  The size sz should not be too
    small, a minimum size of 1 Kb seems to work fine for small
    caches; larger cache sizes are also supported.

  * TimeOut (t,sz) : elements are emitted from the cache after
    approximately t time after they are inserted.

[('a,'b) cache] abstract type of cache. A cache is a
mapping from keys of type 'a to elements of type 'b. Only
values of type 'a Type and 'b Type can be used as keys and
elements, respectively.

['a Type] abstract type of either a key or element that
can be used in a cache.

[name] abstract type of the name of a cache.

[get (cn,ck,aType,bType)] returns a cache which is named
cn.  The cache will be a mapping from keys of type aType
into elements of type bType. The cache strategy is
described by ck.

 * If no cache exists with name cn, then a new cache is
   created.

 * If a cache c exists with name cn, then there are two
   possibilities:

    1) If c is a mapping from aType to bType, then c is
       returned.

    2) If c is not a mapping from aType to bType, then a
       new cache c' is created and returned.

   It is possible to create two caches with the same name,
   but only if they describe mappings of different type.
```

```
 [lookup c k] returns the value associated with the key k
 in cache c; returns NONE if k is not in the cache.

 [insert (c,k,v)] associates a key k with a value v in the cache c;
 overwrites existing entry in cache if k is present, in which case the
 function returns false. If no previous entry for the key is present
 in the cache, the function returns true.

 [flush c] deletes all entries in cache c.

 [memoize c f] implements memoization on the function f. The function
 f must be a mapping of keys and elements that can be stored in a
 cache, that is, f is of type 'a Type -> 'b Type.

 [memoizePartial c f] memoizes function values y where f returned SOME
 y.

 [pp_type aType] pretty prints the type aType.

 [pp_cache c] pretty prints the cache.
*)
```

B.3 The WEB_CONN Signature

```
signature WEB_CONN = sig
  type set
  val returnHtml         : int * string -> unit
  val returnXhtml        : int * string -> unit
  val return             : string -> unit
  val returnFile         : int * string * string -> unit
  val write              : string -> unit
  val returnRedirect     : string -> unit
  val returnRedirectWithCode     : int * string -> unit
  val setMimeType        : string -> unit
  val getQuery           : unit -> set option
  val getRequestData     : unit -> string
  val formvar            : string -> string option
  val formvarAll         : string -> string list
  val storeMultiformData : string * string -> unit
  val headers            : unit -> set
  val host               : unit -> string
  val location           : unit -> string
  val peer               : unit -> string
  val scheme             : unit -> string
  val port               : unit -> int
  val redirect           : string -> unit
  val server             : unit -> string
  val url                : unit -> string list
  val method             : unit -> string
  val contentLength      : unit -> int
  val hasConnection      : unit -> bool
  val add_headers        : (string * string) -> unit
end

(*
 [set] abstract type identical to Web.Set.set.

 [returnHtml (sc,s)] sends HTML string s with status code sc and
 mime-type text/html to client, including HTTP headers and
 Cache-Control header set to no-cache. May raise MissingConnection.

 [returnXHtml (sc,s)] sends XHTML string s with status code sc and
```

mime-type application/xhtml+xml to client, including HTTP headers and Cache-Control header set to must-revalidate. May raise MissingConnection.

[return s] sends HTML string s with status code 200 to client, including HTTP headers. May raise MissingConnection.

[returnFile (sc,mt,f)] sends file f with status code sc to client, including HTTP headers. The mime type is mt. Raises MissingConnection if the execution is not associated with a connection. Raises Fail(msg) if the file cannot be opened for reading.

[write s] sends string s to client, excluding HTTP headers. May raise MissingConnection.

[returnRedirect loc] sends redirection HTTP response to client (status code 302), with information that the client should request location loc. May raise MissingConnection.

[getQuery()] constructs and returns a set representing the query data associated with the connection. It reads the POST content or the query string. The POST content takes precedence over the query string.

[formvar k] returns the first query data associated with the key k; the function returns NONE if no query data is present for the argument key k.

[formvarAll k] returns all values associated with key k in the query data; the function returns the empty list if no query data is present for the argument key k.

[storeMultiformData (fv,filename)] stores the uploaded file represented by formvariable fv in file filename. Raises Fail if some error happens (e.g., filename can't be opened, fv does not exists or fv is not an uploaded file.

[headers()] returns, as a set, the HTTP headers associated with the connection.

```
 [host()] returns the server hostname associated with the connection.

 [location()] returns the HTTP location associated with the
 connection. For example: http://www.avalon.com:81. A server may be
 associated with more than one location at a given time, although
 there is always only on location associated with a connection..

 [peer()] returns the name of the peer associated with the
 connection. The peer address is determined by the communications
 driver in use by the connection. Typically, it is a dotted IP
 address, for example, 199.221.53.205, but this is not guaranteed.

 [peerPort()] returns the port from which the peer is connected.

 [port()] returns the server port number associated with the
 connection.

 [redirect f] performs an internal redirect, to the file f; i.e.,
 makes it appear that the user requested a different URL and then run
 that request. This form of redirect does not require the running of
 an additional thread.

 [server()] returns the name of the server associated with the
 connection.

 [url()] return the url (relative to server-root) associated with the
 request.

 [hasConnection()] returns true if a connection is available. Returns
 false otherwise. For the execution of init scripts and scheduled
 scripts, no connection is available. This function may be used to
 protect execution of code that requires a connection (e.g., execution
 of library code).

 [add_headers (key,value)] adds key:value to the http header

 [fullRequest()] returns the entire request as a string. Raises
 exception MissingConnection if no connection is present.
*)
```

B.4 The WEB_COOKIE Signature

```
signature WEB_COOKIE = sig
  exception CookieError of string
  type cookiedata = {name   : string,
     value  : string,
     expiry : Date.date option,
     domain : string option,
     path   : string option,
     secure : bool}
  val allCookies     : unit -> (string * string) list
  val getCookie      : string -> (string * string) option
  val getCookieValue : string -> string option
  val setCookie      : cookiedata -> unit
  val setCookies     : cookiedata list -> unit
  val deleteCookie   : {name: string, path: string option}
                       -> unit
end

(*
 [CookieError s] exception raised on error with message s.

 [cookiedata] type of cookie.

 [allCookies()] returns a list [(n1,v1), (n2,v2), ...,
 (nm,vm)] of all the name=value pairs of defined cookies.

 [getCookie cn] returns SOME(value) where value is the
 'cn=value' string for the cookie cn, if any; otherwise
 returns NONE.

 [getCookieValue cn] returns SOME(v) where v is the value
 associated with the cookie cn, if any; otherwise returns
 NONE.

 [setCookie {name,value,expiry,domain,path,secure}] returns
 a string which (when transmitted to a browser as part of
 the HTTP response header) sets a cookie with the given name,
 value, expiry date, domain, path, and security level.
```

```
 [setCookies ckds] returns a string which (when transmitted
 to a browser as part of the HTTP response header) sets the
 specified cookies.

 [deleteCookie {name,path}] returns a string that (when
 transmitted to a browser as part of the HTTP response
 header) deletes the specified cookie by setting its expiry
 to some time in the past.
*)
```

B.5 The WEB_DB Signature

```
signature WEB_DB = sig
  structure Handle : WEB_DB_HANDLE

  type 'a Type
  val config : 'a Type * string * 'a -> unit

  (* Data manipulation language *)
  val dml           : quot -> unit
  val exec          : quot -> unit
  val maybeDml      : quot -> unit
  val panicDml      : (quot -> 'a) -> quot -> unit

  (* Stored Procedure *)
  val execSp  : quot list -> unit

  (* Queries *)
  val fold    : ((string->string)*'a->'a) -> 'a -> quot -> 'a
  val foldCol : (string list -> (string->string option)*'a->'a)
                -> 'a -> quot -> 'a
  val app     : ((string->string)->'a) -> quot -> unit
  val appCol  : (string list -> (string->string option)->'a)
                -> quot -> unit
  val list    : ((string->string)->'a) -> quot -> 'a list
  val listCol : (string list -> (string->string option)->'a)
                -> quot -> 'a list

  val oneField      : quot -> string
  val zeroOrOneField: quot -> string option
  val oneRow        : quot -> string list
  val oneRow'       : ((string->string)->'a) -> quot -> 'a
  val zeroOrOneRow  : quot -> string list option
  val zeroOrOneRow' : ((string->string)->'a) -> quot -> 'a option
  val existsOneRow  : quot -> bool

  (* Sequences *)
  val seqNextvalExp : string -> string
  val seqNextval    : string -> int
  val seqCurrvalExp : string -> string
```

```
  val seqCurrval    : string -> int

  (* Miscellaneous *)
  val sysdateExp    : string
  val qq            : string -> string
  val qqq           : string -> string
  val toDate        : string -> Date.date option
  val timestampType : string
  val toTimestampExp: string -> string
  val toTimestamp   : string -> Date.date option
  val fromDate      : Date.date -> string
  val toDateExp     : string -> string
  val valueList     : string list -> string
  val setList       : (string*string) list -> string
  val toBool        : string -> bool option
  val fromBool      : bool -> string
  val toReal        : string -> real option
  val fromReal      : real -> string
end

(*
 [dml sql] executes the data manipulation language command sql using a
 database handle obtained from the next pool. Raises Fail msg if sql
 is unsuccessful; msg is the error message returned from the database.

 [maybeDml sql] executes sql and returns the value unit. Does not
 raise Fail - errors are suppressed.

 [panicDml f sql] executes sql and returns the value unit. On error
 the function f is applied to an error string. The function always
 returns unit.

 [fold f b sql] executes SQL statement sql and folds over the result
 set. b is the base and f is the fold function; the first argument to
 f is a function that maps column names to values. Raises Fail msg on
 error.

 [foldSet f b sql] similar to fold except that f takes the result set
 as argument. Raises Fail msg on fail.
```

[app f sql] executes SQL statement sql and applies f on each row in the result set. Raises Fail on error.

[list f sql] executes SQL statement sql and applies f on each row in the result set. The result elements are returned as a list. Raises Fail on error.

[oneField sql] executes SQL statement sql, which must return exactly one row with one column, which the function returns as a string. Raises Fail on error.

[zeroOrOneField sql] executes SQL statement sql, which must return either zero or one row. If one row is returned then there must be exactly one column in the row. Raises Fail on error.

[oneRow sql] executes SQL statement sql, which must return exactly one row. Returns all columns as a list of strings. Raises Fail on error.

[oneRow' f sql] executes SQL statement sql, which must return exactly one row. Returns f applied on the row. Raises Fail on error.

[zeroOrOneRow sql] executes SQL statement sql, which must return either zero or one row. Returns all columns as a list of strings. Raises Fail on error.

[zeroOrOneRow' f sql] executes SQL statement sql, which must return either zero or one row. Returns f applied on the row if a row exists. Raises Fail on error.

[existsOneRow sql] executes SQL statement sql and returns true if the query results in one or more rows; otherwise returns false. Raises Fail on error.

[seqNextvalExp seq_name] returns a string to fit in an SQL statement generating a new number from sequence seq_name.

[seqNextval seq_name] executes SQL statement to generate a new number from sequence seq_name. Raise Fail on error.

[seqCurrvalExp seq_name] returns a string to fit in an SQL statement returning the current number from the sequence seq_name.

[seqCurrval seqName] executes SQL statement to get the current number from sequence seq_name. Raises Fail on error.

[sysdateExp] returns a string representing the current date to be used in an SQL statement (to have your application support different database vendors).

[qq v] returns a string with each quote (') replaced by double quotes ('') (e.g., qq("don't go") = "don''t go").

[qqq v] similar to qq except that the result is encapsulated by quotes (e.g., qqq("don't go") = "'don''t go'").

[toDate d] returns the Date.date representation of d, where d is the date representation used in the particular database. Returns NONE if d cannot be converted into a Date.date. Only year, month and day are considered.

[toBool b] returns the Bool.bool representation of a boolean, where b is the bool representation used in the particular database. Returns NONE if b cannot be converted into a Bool.bool.

[timestampType] returns the database type (as a string) representing a timestamp (to have your application support different database vendors).

[toTimestampExp d] returns a string to put in a select statement, which will return a timestamp representation of column d. Example: `select ^(Db.toTimestampExp "d") from t` where d is a column of type date (in oracle) or datatime (in PostgreSQL and MySQL).

[toTimestamp t] returns the Date.date representation of t, where d is the timestap representation from the database. Returns NONE if t cannot be converted into a Date.date. Year, month, day, hour, minutes and seconds are considered.

[fromDate d] returns a string to be used in an SQL statement to

```
 insert the date d in the database.

 [fromBool b] returns a Bool.bool used in an SQL statement to insert a
 bool b in the database

 [valueList vs] returns a string formatted to be part of an insert
 statement:

       `insert into t (f1,f2,f3)
        values (^(Db.valueList [f1,f2,f3]))`

 is turned into

      `insert into t (f1,f2,f3)
       values ('f1_','f2_','f3_')`

 where fi_ are the properly quoted values.

 [setList nvs] returns a string formatted to be part of an update
 statement. Say nvs = [(n1,v1),(n2,v2)], then

       `update t set ^(Db.setList nvs)`

 is turned into

       `update t set n1='v1_',n2='v2_'`

 where vi_ are the properly quoted values.
*)
```

B.6 The WEB_DB_HANDLE Signature

```
signature WEB_DB_HANDLE = sig
  (* Database handles *)
  type db

  val getHandle       : unit -> db
  val putHandle       : db -> unit
  val wrapDb          : (db -> 'a) -> 'a

  (* Data manipulation language *)
  val dmlDb           : db -> quot -> unit
  val execDb          : db -> quot -> unit
  val panicDmlDb      : db -> (quot->'a) -> quot -> unit

  (* Transactions *)
  val dmlTransDb      : db -> (db -> 'a) -> 'a
  val dmlTrans        : (db -> 'a) -> 'a
  val panicDmlTransDb : db -> (quot->'a) -> (db->'a) -> 'a
  val panicDmlTrans   : (quot->'a) -> (db->'a) -> 'a

  (* Stored Procedure *)
  val execSpDb        : db -> quot list -> unit

  (* Queries *)
  val foldDb    : db -> ((string->string)*'a->'a) -> 'a -> quot -> 'a
  val foldDbCol : db -> (string list -> (string -> string option) * 'a
                         -> 'a)
                  -> 'a -> quot -> 'a
  val appDb     : db -> ((string->string)->'a) -> quot -> unit
  val appDbCol  : db -> (string list -> (string->string option)->'a)
                  -> quot -> unit
  val listDb    : db -> ((string->string)->'a) -> quot -> 'a list
  val listDbCol : db -> (string list -> (string->string option)->'a)
                        -> quot -> 'a list
  val zeroOrOneRowDb  : db -> quot -> string list option
  val oneFieldDb      : db -> quot -> string
  val zeroOrOneFieldDb: db -> quot -> string option
  val oneRowDb        : db -> quot -> string list
  val oneRowDb'       : db -> ((string->string)->'a) -> quot -> 'a
```

```
  val zeroOrOneRowDb' : db -> ((string->string)->'a) -> quot
                        -> 'a option
  val existsOneRowDb  : db -> quot -> bool

  (* Sequences *)
  val seqNextvalDb    : db -> string -> int
  val seqCurrvalDb    : db -> string -> int
end

(*
 [db] type of database handle. Whenever the Web server talks to the
 database, it is by means of a database handle. Database handles are
 kept in the Web server using a prioritized set of pools. Each Web
 script obtains and releases database handles from the set of pools in
 a stack-like manner (each script may own at most one database handle
 from each pool). This arrangement is to avoid the possibility of
 deadlocks in case multiple Web scripts run simultaneously.

 [getHandle] returns a database handle from the next available
 pool. Raises Fail if no more pools are available.

 [putHandle db] returns the database handle db to its pool and makes
 the pool available to a subsequent call to getHandle.

 [initPools pools] initializes the set of pools. The pools must be
 defined in the nsd.tcl configuration file. See the file lib/Db.sml
 for a use of this function.

 [dmlDb db dml] executes the data manipulation language command dml
 using database handle db. Raises Fail msg if dml is unsuccessful; msg
 is the error message returned from the database.

 [panicDmlDb db f sql] executes the data manipulation language command
 dml using database handle db. Calls the function f with with an error
 message as argument if the dml command is unsuccessful. panicDmlDb
 returns unit and raises an exception only if f does.

 [dmlTransDb db f] executes function f using handle db, which may send
 a series of SQL statements to the database. All SQL statements are
 executed as one atomic transaction. If any statement fails or any
```

exception is raised inside f, then the transaction is rolled back and the exception is raised.

[dmlTrans f] similar to dmlTransDb, but with a database handle obtained from the next available pool.

[panicDmlTransDb db f_panic f_db] same as dmlTransDb except that on error function f_panic is executed. panicDmlTransDb returns the value returned by f_panic unless f_panic raises an exception, in which case panicDmlTransDb raises this exception.

[panicDmlTrans f_panic f_db] similar to panicDmlTransDb, but a database handle is obtained from the next available pool.

[foldDb db f b sql] executes SQL statement sql and folds over the result set. b is the base and f is the fold function; the first argument to f is a function that maps column names to values. Raises Fail msg on error.

[foldSetdb db f b sql] similar to foldDb except that f takes the result set as argument. Raises Fail msg on fail.

[appDb db f sql] executes SQL statement sql and applies f on each row in the result set. Raises Fail on error.

[listDb db f sql] executes SQL statement sql and applies f on each row in the result set. The result elements are returned as a list. Raises Fail on error.

[zeroOrOneRowDb db sql] executes SQL statement that must return either zero or one row. Returns all columns as a list of strings. Raises Fail on error.

[oneFieldDb db sql] executes SQL statement sql, which must return exactly one row with one column, which the function returns as a string. Raises Fail on error.

[zeroOrOneFieldDb db sql] executes SQL statement sql, which must return either zero or one row. If one row is returned then there must be exactly one column in the row. Raises Fail on error.

```
 [oneRowDb db sql] executes SQL statement sql, which must return
 exactly one row. Returns all columns as a list of strings. Raises
 Fail on error.

 [oneRowDb' db f sql] executes SQL statement sql, which must return
 exactly one row. Returns f applied on the row. Raises Fail on error.

 [zeroOrOneRowDb' db f sql] executes SQL statement sql, which must
 return either zero or one row. Returns f applied on the row if it
 exists. Raises Fail on error.

 [existsOneRowDb db sql] executes SQL statement sql and returns true
 if one or more rows are returned; otherwise returns false. Raises
 Fail on error.

 [seqNextvalDb db seq_name] executes SQL statement using database
 handle db to generate a new number from sequence seq_name. Raise Fail
 on error.

 [seqCurrvalDb db seqName] executes SQL statement using database
 handle db to get the current number from sequence seq_name. Raises
 Fail on error.

 [wrapDb f] obtains a handle db with getHandle. applies f to db and
 before returning the result, the handle db is returned with
 putHandle.
*)
```

B.7 The WEB_DYNLIB Signature

```
signature WEB_DYNLIB = sig
  datatype flag = NOW | LAZY
  type ForeignLib
  val dlopen : string option * flag * bool -> ForeignLib
  val dlsym  : string * string * ForeignLib -> unit
  val isLinked : string -> bool
end
```

B.8 The WEB_INFO Signature

```
signature WEB_INFO = sig
  structure Type : WEB_SERIALIZE
  val hostname            : unit -> string
  val pid                 : unit -> int
  val uptime              : unit -> int
  val configGetValue      : ('a Type.Type * string) -> 'a option
  val configSetValue      : ('a Type.Type * string * 'a) -> unit
  val configSetSpecialValue : ((('a Type.Type * string * 'a) ->
                                 unit)
                               * 'a Type.Type * string * 'a) ->
                                 unit
  val pageRoot            : unit -> string
  val getAuxConfigData    : unit -> string option
  val getUser             : unit -> string option
  val getAuthType         : unit -> string option
end

(*
 [hostname()] returns the host name of the machine.

 [pid()] returns the process id of the server process.

 [uptime()] returns the number of seconds the server process
 has been running.

 [configGetValue(T,key)] fetches value of type T associated with key
 if it exists.

 [configSetValue(T,key,v)] associates with key the value v of type T.

 [pageRoot()] returns the directory for which the server
 serves pages.

 [getAuxConfigData()] returns some string if SmlAuxData is defined
 in you webserver configuration file and NONE otherwise.

 [getUser()] returns SOME username if an authentication check has
 succeeded. Returns NONE otherwise.
```

```
 [getAuthType()] returns SOME authtype if an authentication check of
 type authtype has succeeded. Returns NONE otherwise.
*)
```

B.9 The WEB_LOG Signature

```
signature WEB_LOG = sig
  type LogSeverity
  val Emergency : LogSeverity
  and Alert     : LogSeverity
  and Critical  : LogSeverity
  and Error     : LogSeverity
  and Warning   : LogSeverity
  and Notice    : LogSeverity
  and Info      : LogSeverity
  and Debug     : LogSeverity
  val log : LogSeverity * string -> unit
  val advLog : LogSeverity * 'a * ('a -> string) -> 'a
end

(*
 [LogSeverity] Type of log severity level.

 [Emergency] something extremely bad occurred.

 [Alert]

 [Critical]

 [Error] something bad occurred.

 [Warning] default logging level.

 [Notice]

 [Info]

 [Debug] lowest logging level.

 [log (ls,s)] write the string s to the log file with log
 severity ls.

 [advLog(ls,s,f)] log f(s) with log severity ls and return s.
*)
```

B.10 The WEB_LOW_MAIL Signature

```
signature WEB_LOW_MAIL = sig
  type MX_FQDN
  type mailer
  exception ConnectionErr of
       (string * (string * string) list
        * (string * string) list * (string * string) list)
  val getFQDN_MX : string -> (int * int * MX_FQDN) list
  val FQDN_MX_toString : MX_FQDN -> string
  val FQDN_MX_fromString : string -> MX_FQDN
  val FQDN_MX_compare : MX_FQDN * MX_FQDN -> order
  val initConn : MX_FQDN -> mailer
  val sendmail : string list * string * string * mailer ->
                 (string *string) list * (string * string) list
                 * (string * string) list
  val closeConn : mailer -> unit
  val getDefaultServer : unit -> MX_FQDN option
end

(*
 [MX_FQDN] represents a Fully Qualified Domain Name for an MX record
 (an smtp server).

 [mailer] represents the mail connection (socket and protocol state).

 [ConnectionErr] may be raised by initConn and sendmail if the mailer has
 problems and needs to call closeConn.

 [getFQDN_MX] take a domain names and returns a list of triplets where
 the first element is the preference, the second element is the time
 to live and the third element is the Fully Qualified Domain Name for
 the mail server to use if such exists. The list will be prioritized
 with the best server as the first element, second best server as the
 second element, etc.

 [initConn] initializes a connection with the server represented by
 the FQDN and associates the exception with the connection. initConn
 and sendmail may raise the exception.
```

```
 [sendmail] send mails to the people through the connection given by
 the mailer.  sendmail returns the id on the mails that was accepted
 by the mail server.

 [closeConn] closes the connection given by mailer and returns three
 lists. The first list contain ids on mails accepted by the mail
 server, the second list contain ids on mails that was temporary
 undeliverable and the third list contain ids on permanent
 undeliverable mails.
*)
```

B.11 The WEB_MAIL Signature

```
signature WEB_MAIL = sig
  type email = {to: string list, cc: string list,
                bcc: string list, from: string,
                subject: string, body: string,
                extra_headers: string list}
  val sendmail : email -> unit
  val send     : {to: string, from: string,
  subject: string, body: string} -> unit
  datatype CharSet = UTF8 | ISO88591 | USASCII
  val mail : ('a -> (email * 'a * CharSet) option) ->
             ((email * (string * string) list * 'b) -> 'b) ->
             'a -> 'b -> ('a * 'b)
end

(*
 [sendmail {to,cc,bcc,from,subject,body,extra_headers}] sends
 an email to the addresses in to, cc, and bcc.

 [send {to,from,subject,body}] abbreviated version of
 sendmail.

 [mail f g a b] Advanced mail interface that supports sending of
 multiple emails using one SMTP connection.
*)
```

B.12 The WEB_MIME Signature

```
signature WEB_MIME = sig
  val getMime : string -> string
  val addEncoding : string -> string
end
(*
 [getMime s] returns the mime-type of the file s based on the file's
 extension and it's content.

 [addEncoding s] adds configured encoding to mime-type s.
*)
```

B.13 The WEB_SERIALIZE Signature

```
signature WEB_SERIALIZE =
sig
  type 'a Type = {name: string,
  to_string: 'a -> string,
  from_string: string -> 'a}
  val Pair : 'a Type -> 'b Type -> ('a * 'b) Type
  val Option : 'a Type -> ('a option) Type
  val List : 'a Type -> ('a list) Type
  val Triple : 'a Type -> 'b Type -> 'c Type
       -> ('a * 'b * 'c) Type
  val Unit : unit Type
  val Int : int Type
  val Real : real Type
  val Bool : bool Type
  val Char : char Type
  val String : string Type
  val Time : Time.time Type
end

(*
 [Pair aType bType] returns the pair type representing the
 pairs (a,b) where a is of type aType and b is of type
 bType.

 [Option aType] returns the type aType option, representing
 a option where a is of type aType.

 [List aType] returns the list type representing the list
 of elements of type aType.

 [Triple aType bType cType] similar to Pair except that the
 triple is represented with as one Pair embedded in another
 Pair: ((a,b),c) where a is of type aType, b is of type
 bType and c is of type cType.

 [Unit] predefined type representing units.

 [Int] predefined type representing integers.
```

```
 [Real] predefined type representing reals.

 [Bool] predefined type representing booleans.

 [Char] predefined type representing characters.

 [String] predefined type representing strings.

 [Time] predefined type representing Time.time values.
*)
```

B.14 The WEB_SET Signature

```
signature WEB_SET = sig
  type set
  val get    : set * string -> string option
  val iget   : set * string -> string option
  val getOpt : set * string * string -> string
  val getAll : set * string -> string list
  val igetAll: set * string -> string list
  val size   : set -> int
  val list   : set -> (string * string) list
  val filter : (string*string->bool) -> set
               -> (string*string) list
  val foldl  : ((string*string)*'a->'a) -> 'a -> set -> 'a
  val foldr  : ((string*string)*'a->'a) -> 'a -> set -> 'a
end

(*
 [set] abstract type of sequences of key-value pairs, returned by some
 calls to the web-server.

 [get (s,k)] returns SOME(v) if v is the first value associated with
 key k in set s; returns NONE if no value is associated with k in s.

 [iget (s,k)] is the insensitive counterpart to get.

 [getOpt (s,k,v)] returns the first value associated with key k in set
 s; returns v if no value is associated with k in s.

 [getAll (s,k)] returns all values associated with key k in set s;
 returns the empty list if no values are associated with k in s.

 [size s] returns the number of elements in a set.

 [list s] returns the list representation of set s.

 [filter f s] returns the list of key-value pairs in s for which
 applying f on the pairs (from left to right) returns true.

 [foldl f acc s] identical to (foldl o list).
```

```
 [foldr f acc s] identical to (foldr o list).
*)
```

B.15 The XMLRPC Signature

```
signature XMLRPC = sig
    include XMLRPC_TYPE

    exception TypeConversion
    exception MethodInvocation of (int * string)
    exception ServerConnection of string

    val rpc : 'a T -> 'b T -> {url : string, method : string}
      -> ('a -> 'b)

    type method
    val dispatch : method list -> unit
    val method : string -> 'a T -> 'b T -> ('a -> 'b) -> method
end

(*
 [TypeConversion] is raised whenever a recieved value dosen't match
 the expected type.

 [MethodInvocation (code, str)] is raised when a fault message is
 recieved from the server; code is the error code returned and str the
 fault string returned.

 [ServerConnection] is raised if problems occur during reading or
 writing to the connection.

 [rpc A B {url, method}] returns a function of type (A -> B) that when
 called will connect to the XML-RPC server resident at the address
 specified by url. The function will call the specified method on the
 server.

 The returned function raises:

   - TypeConversion if the returned XML-RPC response cannot be
     converted to a value of type B.

   - MethodInvocation if a fault value is returned from the server
```

```
   - ServerConnection if problems occur during reading or writing
     to the connection.

 [type method] type of method.

 [method m A B f] returns a method of name m bound to the function f
 of type A -> B.

 [dispatch ms] executes the first method in the list ms for which the
 name equals the extracted actual method name from the client
 request. Raises exception ServerConnection in case of connection
 errors.
*)
```

B.16 The XMLRPC_TYPE Signature

```
signature XMLRPC_TYPE = sig
    type 'a T
    val int    : int T
    val bool   : bool T
    val real   : real T
    val string : string T
    val date   : Date.date T
    val pair   : 'a T * 'b T -> ('a * 'b) T
    val list   : 'a T -> 'a list T
    val array  : 'a T -> 'a Array.array T
    val vector : 'a T -> 'a Vector.vector T
end

(*
 [int] type <int> and <i4>

 [bool] type <boolean>

 [real] type <double>

 [string] type <string>

 [date] type <dateTIme.iso8601>

 [pair] type <struct> where type 'a = member with name=1 and 'b =
 member with name=2

 [list] type <array>. Homogeneous lists where elements must be of
 identical type.

 [array] type <array>. Homogeneous arrays where elements must be of
 identical type.

 [vector] type <array>. Homogeneous vectors where elements must be of
 identical type.
*)
```

Index