A Functional Approach to Web Publishing

Martin Elsman

Niels Hallenberg

SMLserver

A Functional Approach to Web Publishing

Martin Elsman (mael@dina.kvl.dk)
Royal Veterinary and Agricultural University of Denmark
IT University of Copenhagen

Niels Hallenberg (nh@it.edu)
IT University of Copenhagen

January 29, 2002

Contents

Preface

The ideas behind the SMLserver project came alive 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 friends 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 friends 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.

The SMLserver project was initialized in the end of 2000 by the construction of an embeddable runtime system and a bytecode backend for the ML Kit [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. Although the cost of using Standard ML for Web applications is a more tedious development cycle due to the compilation phase of Standard ML, its static type system causes many bugs to be found before a Web site is launched.

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

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

user's manual for SMLserver. Instead, the book is meant to give a broad overview of the possibilities of using SMLserver for Web development. The choice of content and the examples presented in the book are inspired from more than two 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 sites with SMLserver, including an alumni 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

Copenhagen, Denmark
January, 2002

Chapter 1

Introduction

SMLserver is a module for AOLserver, an open source Web server provided by America Online (AOL). SMLserver comes with a compiler for compiling Web applications written in Standard ML [MTHM97] into bytecode to be interpreted by the SMLserver module. AOLserver has an extensive Application Programmer Interface (API), including interfaces to most Relational Database Management Systems (RDBMSs) on the market. Although AOLserver scales to work well for very large sites such as the AOL Web site, dynamic Web sites based on AOLserver must be constructed using either a C programming interface or a TCL programming interface. Whereas C provides for very fast execution of compiled code, the TCL programming interface provides programmers with a quick development cycle through the embedded interpreter for the TCL language.

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

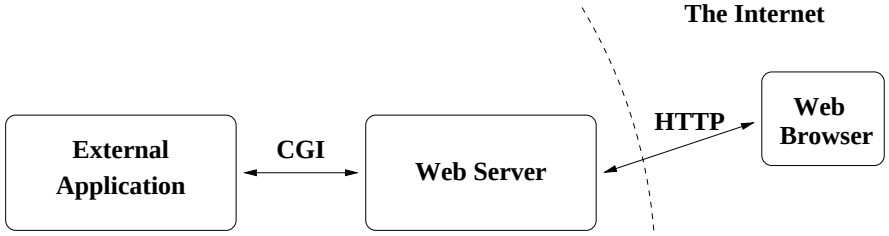
- Different RDBMSs, including Oracle, Postgresql, and MySQL, may be accessed through a generic database interface.
- SMLserver provides easy access to HTTP header information, including form content and cookie information.
- Efficient caching support makes it possible to decrease the load caused

by frequently run database queries, such as the querying of passwords for user authentication.

- SMLserver has a simple interface for Web applications to send emails.
- Secure Socket Layer support (SSL) allows for encrypted communication between the Web server and its clients.

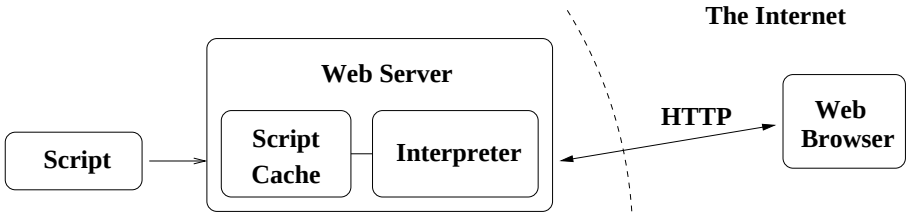
1.1 Web Scripting

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 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 follows:



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 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 systems 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. Finally, Chapter 9 presents a user authentication mechanism based on information stored in a database and cookie information provided by the client browser. A summary is given in Chapter 10.

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

Chapter 2

Getting Started

This chapter describes how to install and setup SMLserver on a Redhat Linux box using the Redhat Package Manager (RPM). To install SMLserver on a Linux box that do not support RPMs, see the file `README_SMLSERVER` in the source distribution, which is available from the SMLserver home page (`www.smlserver.org`).

2.1 Requirements

If your Redhat Linux box does not accept the RPMs, it may be necessary to rebuild the RPMs as described in Section 2.9. Moreover, if your version of Redhat Linux is 6.2 or earlier, you need to install RPM version 4.0.2 or later; see `http://www.redhat.com/support/errata/RHSA-2001-016.html`. Rebuilding the SMLserver RPM or compiling SMLserver from the source distribution requires GCC version 2.96 (or later).

2.2 Installing RPMs

To install the necessary RPMs, run the following two commands on your Redhat Linux box:

```
# rpm -Uvh \
   http://www.smlserver.org/dist/aolserver-3.4-1.i386.rpm
# rpm -Uvh \
   http://www.smlserver.org/dist/smlserver-4.1.0-1.i386.rpm
```

The two commands cause AOLserver and SMLserver to be installed in the directories `/usr/share/aolserver` and `/usr/share/smlserver`, respectively.

2.3 Starting AOLserver

Before you start the AOLserver Web server, a few customizations are necessary:

1. Copy the directory `/usr/share/smlserver/smlserver_demo` to somewhere in your home directory:

   ```
   # cp -a /usr/share/smlserver/smlserver_demo ~/web/
   ```

2. Copy the AOLserver configuration file `~/web/nsd.demo.tcl` to a file `~/web/nsd.user.tcl`, where `user` is your user name. Modify the first five or six lines of the file `~/web/nsd.user.tcl` to suit your needs (see Appendix A).

Your directory structure should now look as follows:

```
/home/user/web/
    nsd.demo.tcl
    nsd.user.tcl
    www/
        demo/
        demo.pm
        ...
    demo_lib/
    lib/
    log/
    ...
```

AOLserver can now be started by executing the command—again substitute `user` with your user name:

```
# /usr/share/aolserver/bin/nsd -t ~/web/nsd.user.tcl -u user
```

By executing the command

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

you should see that AOLserver is running five or six processes. AOLserver writes information into the file `~/web/log/server.log`. By looking at the log, you should see a notice that AOLserver has loaded the module `nssml.so`.

2.4 Compiling the Sample Web Project

Before you can request `sml`-files and `msp`-files from port 8080 on your Linux box (the settings can be altered by editing the file `~/web/nsd.user.tcl`), you need to compile a project file, which mentions the files and libraries that SMLserver should know about. To compile the sample project `demo.pm`, enter the following commands on your system:

```
# cd ~/web/www
# ln -s demo.pm sources.pm
# smlserverc sources.pm
```

Now, try to request the script `http://localhost:8080/demo/index.sml` from a Web browser.

2.5 Interfacing to an RDBMS

To get access to an RDBMS from within your SMLserver scripts, an RDBMS supported by AOLserver must be be installed on your system. One supported RDBMS is Postgresql, which can be obtained from `www.postgresql.org` (you can even find some RPMs there for Redhat Linux). Other options include Oracle 8i, for which a driver can be obtained from ArsDigita (`www.arsdigita.com`), and MySQL (`www.mysql.com`), for which a driver can be obtained from Panoptic Computer Network (`www.panoptic.com/nsmysql/`).

Information on how to interface to Oracle 8i and MySQL is available from the SMLserver home page. The next section describes how to interface to the open source RDBMS Postgresql.

2.6 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 is already installed on the system. We also assume that the sample Web project is compiled as described in Section 2.4.

1. Install the Postgresql driver for AOLserver by executing the following command on your Linux box:

```
# rpm -Uvh \
   http://www.smlserver.org/dist/pgdriver-2.0-1.i386.rpm
```

If the RPM does not install on your system, see Section 2.9 for how to rebuild the package.

2. Start the Postgresql daemon process by executing (as root) the following command:

```
$ /etc/rc.d/init.d/postgresql start
```

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

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

Answer yes to both questions asked by `createuser`.

4. 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 command

```
# pgsql -c "\i ~/web/demo_lib/pgsql/all.sql"
```

5. Insert the password for the new database user in AOLserver configuration file `~/web/nsd.user.tcl`.

6. Restart AOLserver by first killing it using the command

```
# killall nsd
```

and then starting AOLserver again as shown above.

7. Edit the file `~/web/lib/Db.sml`. Make sure that the structure `Db` is bound to the structure `Ns.DbPg`. The lines defining the Oracle structure and the MySQL structure should be commented out:

```
(* For The PgSQL User *)
structure Db : DB = Ns.DbPg
val _ = Db.Pool.initPoolsL ["pg_main","pg_sub"]
```

8. Go start your Web browser and visit the database examples available from `http://localhost:8080/demo/index.sml`.

2.7 Automating Startup of the Web Server

There are basically two reasons why you would want the operating system to control the startup of your Web server:

1. When your machine is rebooted, you may want the Web server to restart once the machine has come back up.

2. If your Web server terminates due to some internal error in the server, you may want the operating system to restart the Web server.

These features are obtained by adding the following line (replace `user` with your actual user name) to the file `/etc/inittab` on your Linux box—you must be root to do this:

```
a1:5:respawn:/usr/share/aolserver/bin/nsd -i \
   -t ~user/web/nsd.user.tcl -u user -g user
```

Then, as root, to have Linux reread the file `/etc/inittab`, execute the command

```
$ /sbin/telinit q
```

Now, if you want to restart your Web server, simply execute—also as root

```
$ killall nsd
```

2.8 So You Want to Write Your Own Project

To write your own project, create a new file `yourproject.pm` and make this project the current project:

```
# cd ~/web
# rm -f sources.pm
# ln -s yourproject.pm sources.pm
```

You can have only one project associated with each Web server that you run. Use the compiler `smlserverc` (located in the directory `/usr/bin`) to compile your project into bytecode. Once your project is compiled, the Web server answers requests of the files listed in the `[...]` part of your project file (see Section 3.5).

Library code to be shared between scripts may be stored anywhere on the system and mentioned in the local part in the project file—look in the sample project file `demo.pm` for examples.

2.9 Rebuilding The RPMs

To rebuild the RPMs for a Redhat Linux box, as root, execute the commands:

```
$ rpm --rebuild \
    http://www.smlserver.org/dist/aolserver-3.4-1.src.rpm
$ rpm --rebuild \
    http://www.smlserver.org/dist/smlserver-4.1.0-1.src.rpm
```

After doing so, the newly created packages can be installed by—also as root—executing the commands:

```
$ rpm -Uvh \
    /usr/src/redhat/RPMS/i386/aolserver-3.4-1.i386.rpm
$ rpm -Uvh \
    /usr/src/redhat/RPMS/i386/smlserver-4.1.0-1.i386.rpm
```

Similarly, to rebuild the RPM for the Postgresql driver, as root, execute the command:

```
$ rpm --rebuild \
    http://www.smlserver.org/dist/pgdriver-2.0.src.rpm
```

When rebuilt, the driver is installed by executing the command:

```
$ rpm -Uvh /usr/src/redhat/RPMS/i386/pgdriver-2.0-1.i386.rpm
```

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 `Ns.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]

[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

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 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 D on page 127.

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

val _ = Ns.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 `Ns` structure, which gives access to the Web server API; to get an overview of what functions are available in the `Ns` structure, consult Appendix E, which lists the SML signature for the structure. The function `Ns.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 30 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.

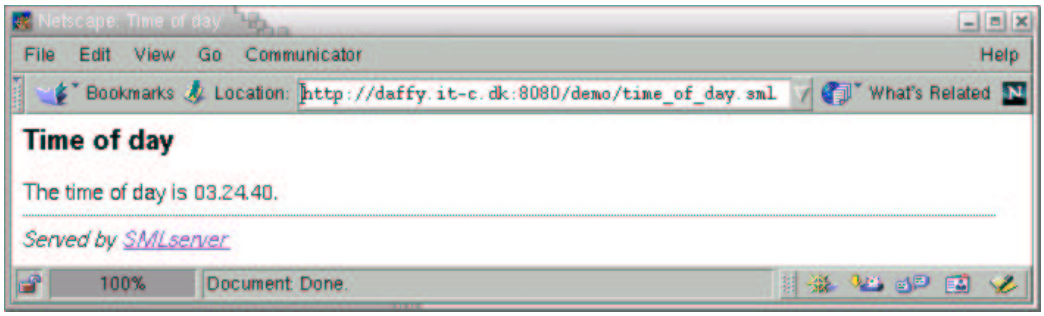


Figure 3.3: The result of requesting the file `time_of_day.sml` using the Netscape browser. The HTTP request causes the compiled `time_of_day.sml` program to be executed on the Web server and the response is sent (via the HTTP protocol) to the Web browser.

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` and are listed in project files along with `.sml`-files (Section 3.5 on page 21 has more to say about projects.)

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>"
```

[3]The program is available as `www/mul.msp` in the demonstration directory.

```
          && $(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>
  <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 and `.msp`-files) are executed when requested by a client.

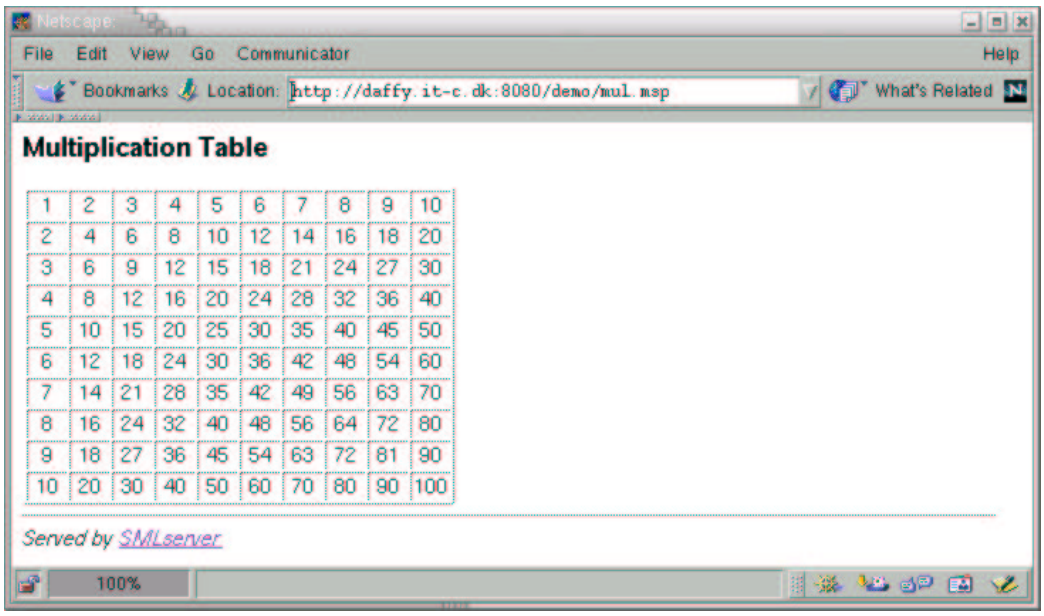


Figure 3.4: The result of requesting the file `mul.msp` using the Netscape browser. The HTTP request causes the compiled `mul.msp` program to be executed on the Web server and the response is sent (via the HTTP protocol) to the Web browser.

AOLserver 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 AOLserver starts is configured in a *configuration file*; a sample AOLserver configuration file is listed in Appendix A.

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

3.5 Project Files

The SMLserver compiler `smlserverc` takes as argument a *project file*, which lists the `.sml`-files and `.msp`-files that a client may request along with Standard ML library code to be used by the client-accessible `.sml`-files and `.msp`-files. By invoking `smlserverc` without arguments, a simple text-based menu-system appears, which supports efficient recompilation of a project upon modification of `.sml`-files and `.msp`-files.

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

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

The two example scripts `time_of_day.sml` and `mul.msp` 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`) as well as to identifiers declared by the Standard ML Basis Library[4] and the project `../lib/lib.pm`. Moreover,

[4]To see what parts of the Standard ML Basis Library that SMLserver supports, consult the file `/usr/share/smlserver/basislib/basislib.pm` on your system.

```
import ../lib/lib.pm
in
    local
     ../demo_lib/Page.sml
     ../demo_lib/FormVar.sml

    in
     (* Leaf files; may refer to identifiers declared in
      * library files, but cannot refer to identifiers
      * in other leaf files. *)
      [ time_of_day.sml
        mul.msp ]
    end
end
```

Figure 3.5: A project file for the two examples in this chapter.

in the local-part of the project file, it is allowed for an `.sml`-file to refer to identifiers declared by previously mentioned `.sml`-files. However, an `.sml`-file or an `.msp`-file mentioned in the square-bracket part of a project file may not refer to identifiers declared by other files mentioned in the square-bracket part of the project file. Thus, in the example project file, `mul.msp` 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.pm` is the default name to use) given as argument:

```
smlserverc sources.pm
```

The bytecode files resulting from compilation of a project are stored in a directory named `PM`, located in the same directory as the project file. To work efficiently with SMLserver, you need not know anything about the content of the `PM` directories. In particular, you should not alter the content of these directories.

3.7 Loading and Serving Pages

The first time SMLserver serves an `.sml`-file or an `.msp`-file, SMLserver loads the bytecode for the Standard ML Basis Library along with user libraries mentioned in the project file before the bytecode for the `.sml`-file or `.msp`-file is loaded. Upon subsequent requests for an `.sml`-file or an `.msp`-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 `NS` signature in Appendix E). 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 AOLserver starts (see Chapter 2), initialization information is written to a *server log file*. The location and name of the server log file is configured in the AOLserver configuration file (see Appendix A). The default name of the server log file is `server.log`.

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

Value	Description (intended use)
`Notice`	Something interesting occurred.
`Warning`	Maybe something bad occurred.
`Error`	Something bad occurred.
`Fatal`	Something extremely bad occurred. The server will shut down after logging this message.
`Bug`	Something occurred that implies there is a bug in your code.
`Debug`	If the server is in *Debug mode*, specified by a flag in the `[ns/parameters]` section of 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:

```
[20/Jul/2001:20:50:02][833.4101][-conn0-]
   Warning: /home/mael/web/www/demo/temp.sml raised Overflow
```

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 `Ns.exit`, which has type `unit -> ty`, for any type `ty`, exits by raising the exception `Interrupt`.

An important aspect of using the function `Ns.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.[5]

3.10 Accessing Setup Information

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

[5]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 D.2 on page 131.

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]

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

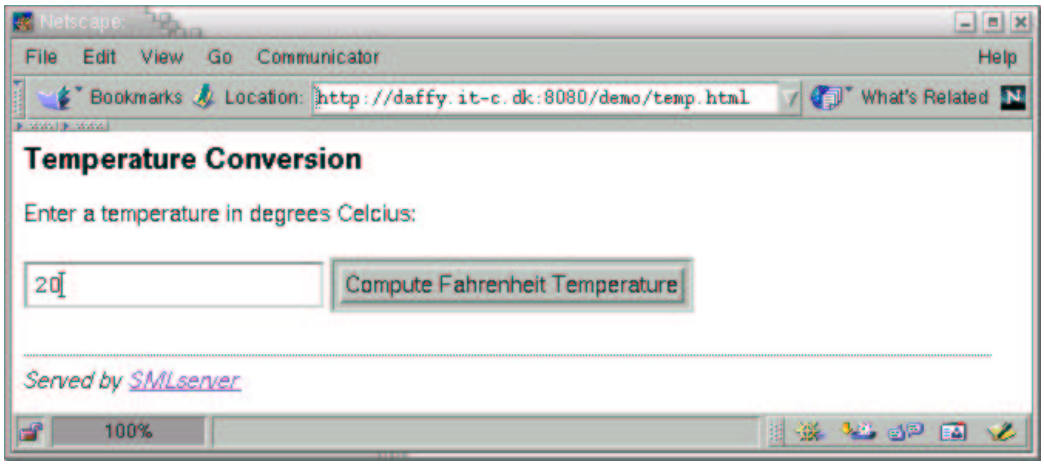


Figure 4.1: The result of displaying the file `temp.html` using the Netscape browser.

```
<html>
 <body bgcolor=white>
 <h2>Temperature Conversion</h2>
 Enter a temperature in degrees Celcius:
 <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>.",
   "<hr> <i>Served by <a href=http://www.smlserver.org>",
   "SMLserver</a></i> </body></html>"]

val _ = Ns.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.

[2]File `smlserver_demo/www/demo/temp.sml`.
[3]File `smlserver_demo/demo_lib/FormVar.sml`.

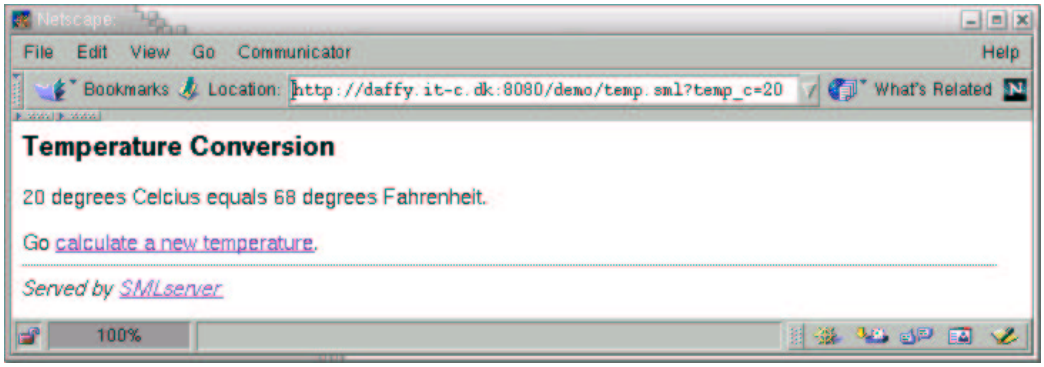


Figure 4.2: The result of a user converting a temperature in degrees Celsius to a temperature in degrees Fahrenheit.

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>
     <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 ML quotations to embed HTML code in your Standard ML Web applications. Similarly to the tem-

[4]File `smlserver_demo/lib/Quot.sml` lists the signature for the `Quot` structure.

perature 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>
```

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.`
   ; Ns.exit())

val persons =
```

[5] File `smlserver_demo/www/demo/recipe.html`.
[6] File `smlserver_demo/www/demo/recipe.sml`.

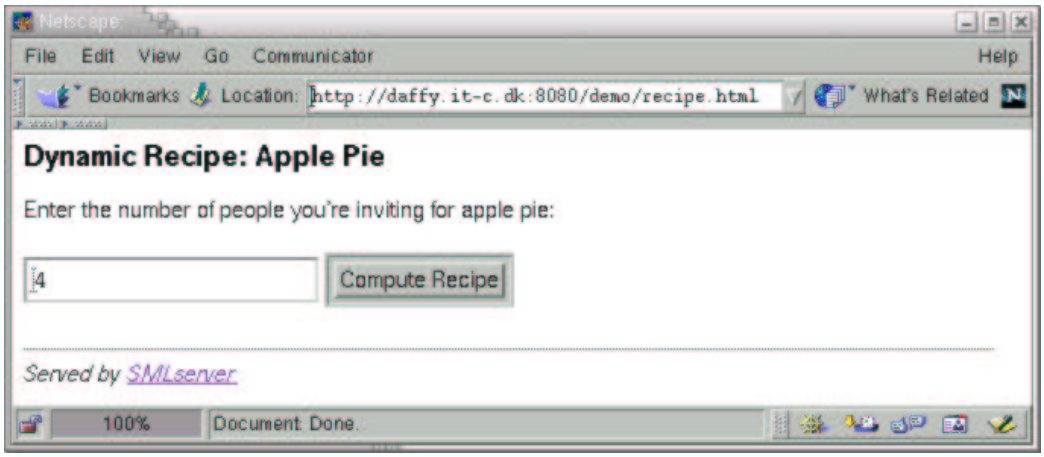


Figure 4.3: The result of requesting the file `recipe.html` using the Netscape browser.

```
  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
      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]

```
fun return head body = Ns.return
  (‘<html>
    <head><title>^head</title>
    </head>
    <body bgcolor=white>
     <h2>^head</h2> ‘ ^^
       body ^^
       ‘<hr><i>Served by
```

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

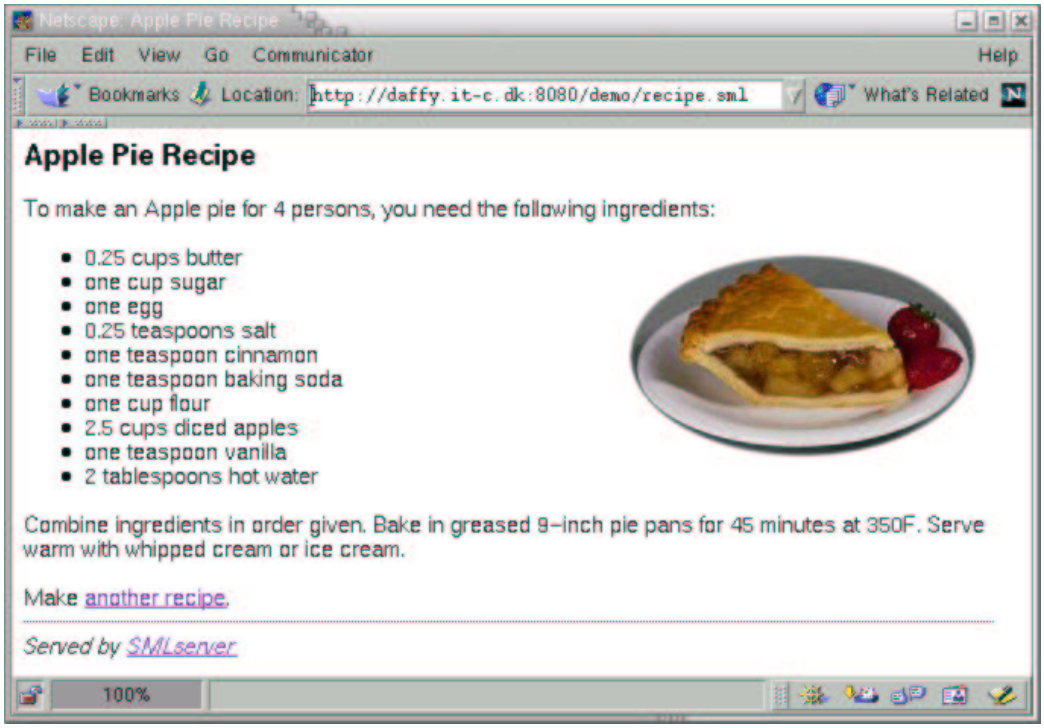


Figure 4.4: The result of computing a recipe for a four-person apple pie.

```
    <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 simple counter consists of one `.sml`-file named `counter.sml`, which uses the `FormVar` functionality (described on page 29 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 `Ns.Conn.formvar` on which the `FormVar` structure is built (see Section 8.3 on page 91). The script `counter.sml` takes the following form:[1]

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

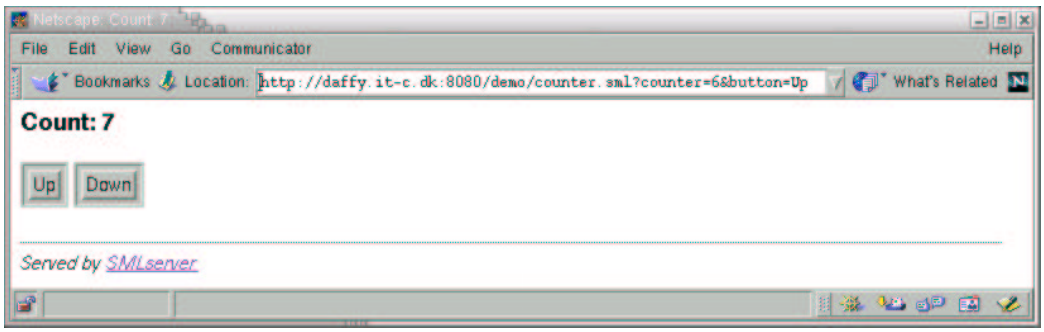


Figure 5.1: The counter rendered by Netscape Navigator after a few clicks on the "Up" button.

```
val counter = Int.toString
  (case FormVar.wrapOpt FormVar.getIntErr "counter"
     of SOME c => (case Ns.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 34 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 = Ns.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)
```

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

```
| 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
   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 services, readily available for use by your Web site! Available services include real-time population clocks (e.g., `http://www.census.gov/cgi-bin/popclock`), stock quote services (e.g., `http://quotes.nasdaq.com`), currency rate services (e.g., `http://se.finance.yahoo.com`), and many others. In this chapter, we shall see how to extract data from another Web site and use the data for content on your own Web site, using so-called regular expressions.

6.1 Grabbing a Page

The SMLserver API has a built-in function `Ns.fetchUrl`, with type `string -> string option`, for fetching a page from the Internet and return the page as a string. Upon calling `Ns.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`.

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://se.finance.yahoo.com/m5?s=USD&t=DKK`.

This URL specifies two form variables, source currency (`s`), and target currency (`t`). 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/m5?s=" ^
  Ns.encodeUrl s ^ "&t=" ^ Ns.encodeUrl t

fun errPage () =
  (Page.return "Currency Service Error"
   `The service is currently not available, probably
    because we have trouble getting information from
```

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

```
    the data source: <a href="^url">^url</a>.‘
    ; Ns.exit())

val pg = case Ns.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 `Ns.encodeUrl` for building the URL; the function `Ns.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 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.

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*.

- `([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 Web addresses `www.itu.dk`, `www.it-c.dk`, and `www.it.edu`.
- `http://hug.it.edu:8034/ps2/(.*)\.sml` : matches all `.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.

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`. Additional examples are available in the file `smlserver_demo/www/demo/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: 66 66 66 66"
```

  evaluates to

```
SOME ["Hans Hansen", "66 66 66 66"]
```

- 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/demo/`.

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 45:

```
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 such example is the *Bill Gates Personal Wealth Clock* (`http://db.photo.net/WealthClockIntl`). This service 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`), the estimated holding of Microsoft shares owned by Bill Gates, and information about the world population from the U.S. Census Bureau (`http://www.census.gov/cgi-bin/ipc/popclockw`). The Web site provides a precise description of the math involved. As of January 24, 2002, the Web site estimates that each and every person in the world has contributed $11.7642 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

A *cache* in SMLserver has type `cache`. Caches may be created, flushed (i.e., emptied), items may be added, and items may be deleted. (Other operations on caches are possible as well.) A cache maps a unique key k to a value v; k and v must be of type `string`.[1] The signature is shown below:

```
structure Cache :
  sig
    type cache
    val createTm : string * int -> cache
    val createSz : string * int -> cache
    val find : string -> cache option
    val findTm : string * int -> cache
    val findSz : string * int -> cache
    val flush : cache -> unit
    val set : cache * string * string -> bool
    val get : cache * string -> string option
    val cacheForAwhile :
      (string -> string) * string * int -> string -> string
    val cacheWhileUsed :
      (string -> string) * string * int -> string -> string
  end
```

Caches are created using either `createTm`, `createSz`, `findTm` or `findSz`. With `createTm` and `findTm` a timeout t in seconds is specified and an item added to the cache lives until it has not been accessed for approximately t seconds. With `createSz` and `findSz` a cache size is specified and the oldest items are deleted when items are added to a full cache. The `find` functions do not create a new cache if one with the same name already exists. Flushing a cache deletes all entries, but the cache still exists. A cache cannot be deleted. Items may be retrieved from a cache with the functions `set` and `get`.

The `cacheForAwhile` and `cacheWhileUsed` functions adds caching functionality ("memorization") to a function f of type `string->string`. For instance, the expression `cacheForAwhile` f *name sec* returns a function f'. The function f' caches the results of evaluating f. Succeeding calls to f' with

[1] It is unsound to have the `cache` type and the primitives support caching of values of arbitrary types.

the same argument results in cached results, except when a cached result no longer lives in the cache. When that is the case, f is evaluated again. With `cacheForAwhile` a result lives for *sec* seconds after it was added to the cache. With `cacheWhileUsed` a result lives for *sec* seconds after the last use.

Thus, to cache HTML pages obtained from foreign sites, a Web site should use the function `cacheForAwhile`, which guarantees that the cache is updated with fresh information even if the cache is accessed constantly. In Section 6.7, 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.

However, when caching password information to evaluate login on every page request (see Chapter 9), a Web site should use the `cacheWhileUsed` function, which makes the password information live in the cache for as long as the user accesses the site.

In the remainder of this section, we present a small caching demonstration, which implements caching of names based on associated email addresses.[2] Figure 6.4 shows the entry form.

The function `findTm` is used to find (or create) a cache with a timeout value of 20 seconds. The script `cache_add.sml` processes the data entered in the form; here is the content of the file:[3]

```
val cache = Ns.Cache.findTm ("people", 20)

val new_p = (* new_p true if new value added *)
  case (Ns.Conn.formvar "email", Ns.Conn.formvar "name")
    of (SOME email, SOME name) =>
      Ns.Cache.set(cache,email,name)
     | _ => false

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

val _ = Page.return "Caching Demonstration"
  `^head <p>
   Go back to <a href=cache.sml>Cache Demo Home Page</a>.`
```

The code to lookup a name in the cache is in the script `cache_lookup.sml`. Again, the function `findTm` is used to get access to the cache and the function

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

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

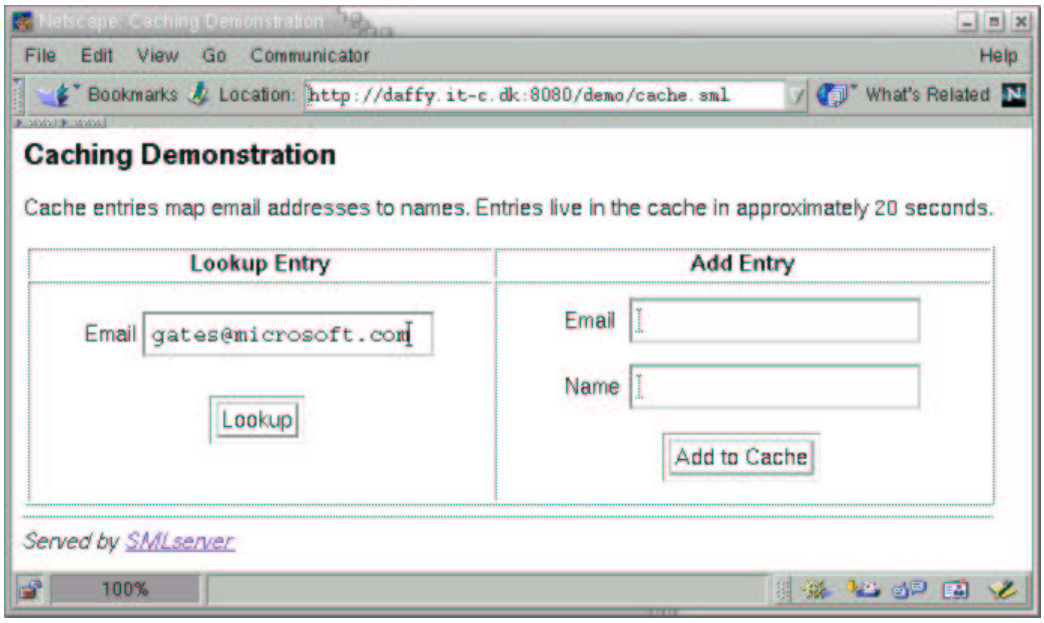


Figure 6.4: The example uses a cache to store pairs of email addresses and names. Cached values are accessible 20 seconds after the last use.

`get` is used to find a name associated with an email address in the cache. The function `get` returns `NONE` if the email address is not in the cache:[4]

```
val cache = Ns.Cache.findTm ("people", 20)

fun returnPage s = Page.return "Caching Demonstration"
  `^s <p>
   Go back to <a href=cache.sml>Cache Demo Home Page</a>.`

val _ = (* new_p is true if new value added *)
  case Ns.Conn.formvar "email"
    of NONE => Ns.returnRedirect "cache.sml"
     | SOME email =>
  returnPage
  (case Ns.Cache.get(cache,email)
     of SOME n => "Name for " ^ email ^ " is: " ^ n
      | NONE => "No name in cache for " ^ email)
```

6.7 Caching Version of Currency Service

In this section we demonstrate the caching function `cacheForAwhile` 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 every access. Instead, the currency rates obtained from Yahoo are cached in 300 seconds (five minutes).

Notice the distinction between the function `cacheForAwhile` and the function `cacheWhileUsed`; the service should not make use of the function `cacheWhileUsed` because rates must be updated every 300 seconds—irrespectively of whether the service is accessed every minute. Here is the script `currency_cache.sml`,[5] which implements the cached version of the Currency Service:

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

val a = getReal ("a", "amount")
```

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

[5] File `smlserver_demo/www/demo/currency_cache.sml`.

```
val s = getString ("s", "source currency")
val t = getString ("t", "target currency")

val url = "http://se.finance.yahoo.com/m5?s=" ^
  Ns.encodeUrl s ^ "&t=" ^ Ns.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>.`
    ; Ns.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 fetch = Ns.Cache.cacheForAwhile
  (fn url => case Ns.fetchUrl url
               of NONE => ""
                | SOME pg =>
                  (case RegExp.extract pattern pg
                     of SOME [r1,r2] => r1 ^ "." ^ r2
                      | _ => ""),
   "currency", 5*60)

val _ =
  case fetch url of
    "" => errPage ()
  | rate_str =>
      let val rate = Option.valOf (Real.fromString rate_str)
      in 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>‘
    end
```

The anonymous function passed to the function `Ns.Cache.cacheForAwhile` 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 `Ns.Cache.cacheForAwhile`, 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.[1]

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`.

[1] SMLserver also supports the popular MySQL database server. However, because MySQL does not implement transaction (in the sense of the ACID test), we do not recommend using MySQL for building Web sites that manipulate important data.

```
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:[2]

```
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;
```

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

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');
```

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',
set 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`[3] 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.

[3]File `smlserver_demo/demo_lib/pgsql/employee.sql`.

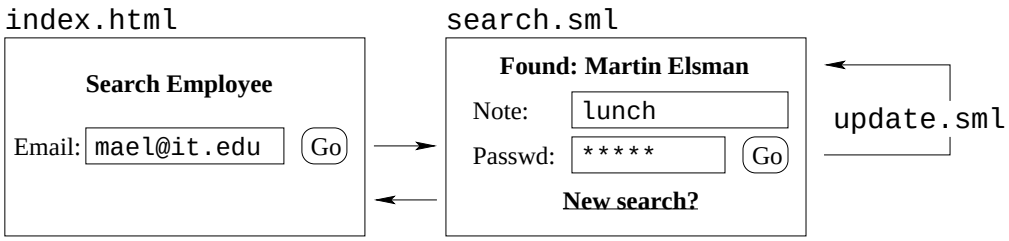


Figure 7.1: Web site diagram for the employee example. Administrator pages for adding and deleting employees are not shown.

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`:[4]

```
<html>
  <head><title>Search the Employee Database</title></head>
```

[4]File `smlserver_demo/www/demo/employee/index.sml`.

```
    <body bgcolor=white>
       <center> <h2>Search the Employee Database</h2> <p>
         <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 `NS_DB` signature.[5] Because SMLserver supports the Oracle RDBMS, the Postgresql RDBMS, and MySQL, there are three structures in the `Ns` structure that matches the `NS_DB` signature, namely `Ns.DbOra`, `Ns.DbPg`, and `Ns.DbMySQL`. The example Web server project file includes a file `Db.sml`, which binds a top-level structure `Db` to the structure `Ns.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., AOLserver) 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

[5] See the file `smlserver_demo/lib/NS_DB.sml`.

```
signature NS_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 `NS_DB` signature.

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 another thread holds on to resource B when attempting to gain access to resource A. An example AOLserver configuration file, which also specifies the initialization of pools and opening of database connections, is shown in Appendix A.

The `NS_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 79 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:[6]

```
fun returnPage title body = Ns.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
```

[6]File `smlserver_demo/www/demo/employee/search.sml`.

```
    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">
                </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:[7]

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

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

val update = `update employee
```

[7] File `smlserver_demo/www/demo/employee/update.sml`.

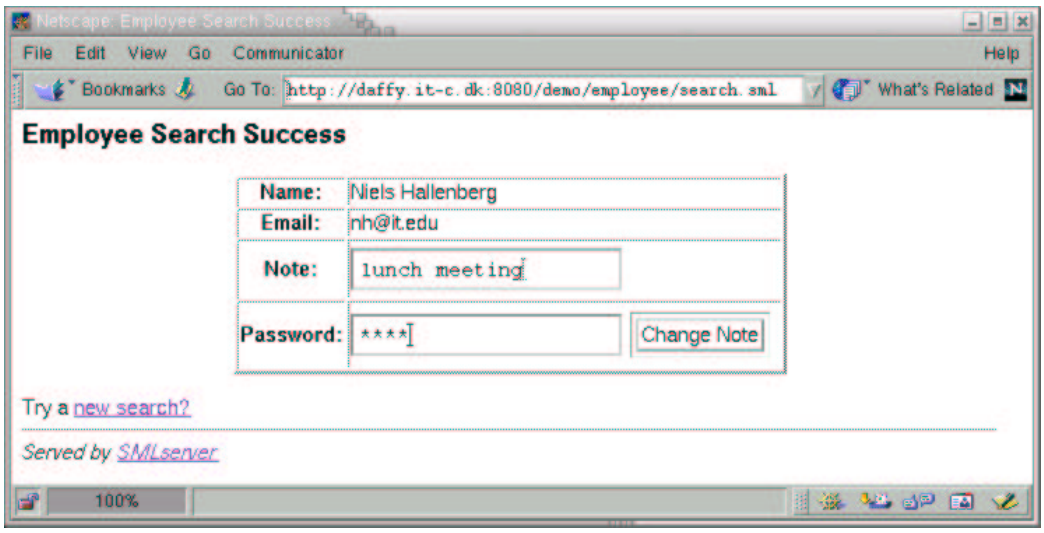


Figure 7.3: The result of searching for an employee with email `nh@it.edu`

```
                set note = ^(Db.qqq note)
                where email = ^(Db.qqq email)
                  and passwd = ^(Db.qqq passwd)‘

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

The function `Ns.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 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

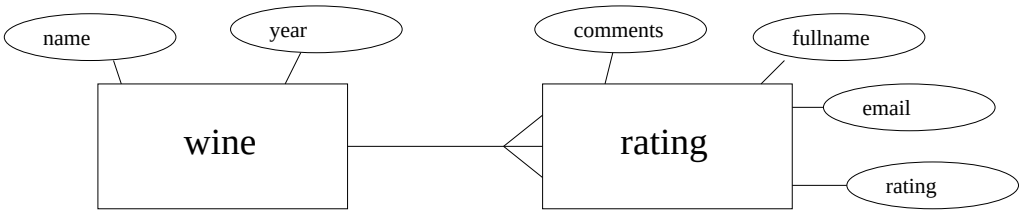


Figure 7.4: E-R diagram for the Best Wine Web site. The fork in the diagram specifies that the relation between the wine-entity and the rating-entity is a one-to-many relation; to every one wine there may be many ratings.

When an entity-relationship 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 entity-relationship 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 entity-relationship diagram for the Best Wine Web site in Figure 7.4.

The entity-relationship 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 that correspond to the entity-relationship diagram in Figure 7.4 look as follows:[8]

```
create sequence wid_sequence;
create table wine (
       wid       integer primary key,
```

[8]File `smlserver_demo/demo_lib/pgsql/rating.sql`.

```
        name      varchar(100) not null,
        year      integer,
        check     ( 0 <= 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 zero and 3000
- No two rows in the `wine` table is associated with the same name and the same year
- 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 to users. The fourth script `add0.sml` implements the rating-insert transaction for inserting a rating in the `rating` table.

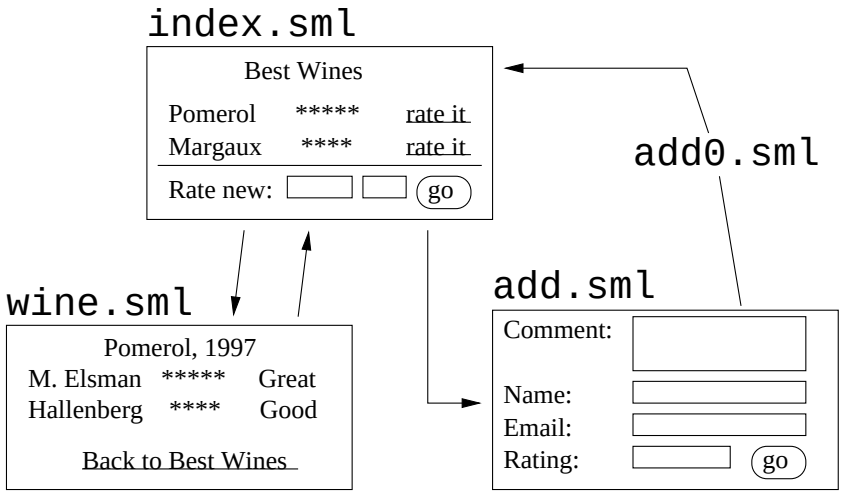


Figure 7.5: Web site diagram for the Best Wine Web site.

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

Implementation of SMLserver Scripts

The cripts `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:[9]

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

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

[9]File `smlserver_demo/demo_lib/RatingUtil.sml`.

```
    (* [bottleImgs n] returns html code for
     * 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 70) 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`:[10]

```
(* 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)
```

[10] File `smlserver_demo/www/demo/rating/wine.sml`.

```
        <td> ^(g "comments")
        <td> ^(RatingUtil.mailto (g "email") (g "fullname"))‘
    end ^^ r) ‘‘ query

val body =
  ‘<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`:[11]

```
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
```

[11] File `smlserver_demo/www/demo/rating/add.sml`.

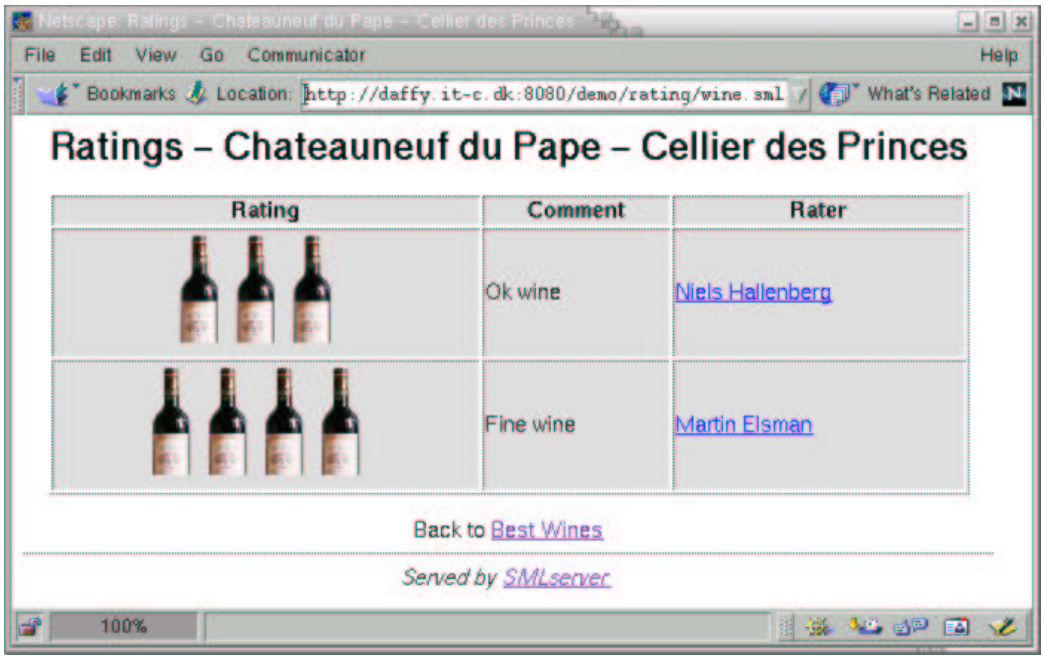


Figure 7.6: The Best Wines comment page.

```
            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 0 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))`
              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>
```

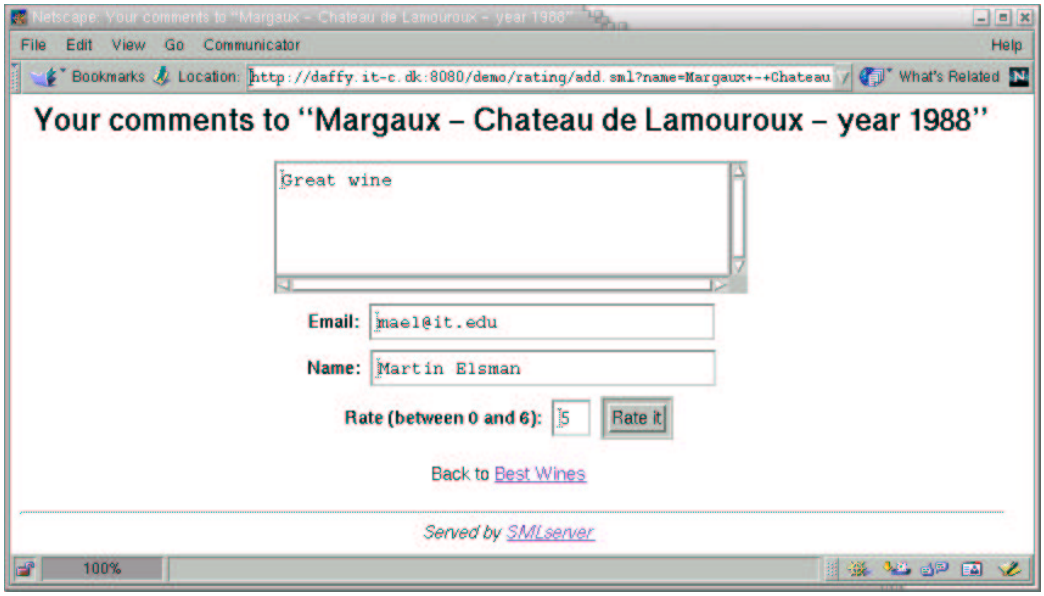


Figure 7.7: The wine rating form. Users are asked to provide ratings between 0 and 6.

```
 <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:[12]

```
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)
               ("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 _ = Ns.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 `Ns.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`:[13]

[12] File `smlserver_demo/www/demo/rating/add0.sml`.
[13] File `smlserver_demo/www/demo/rating/index.sml`.

```
(* 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"
      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...">
```

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

```
</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.

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` 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:

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 6.7 on page 56 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]

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

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

```
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).

```
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]

```
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 _ = Ns.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

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

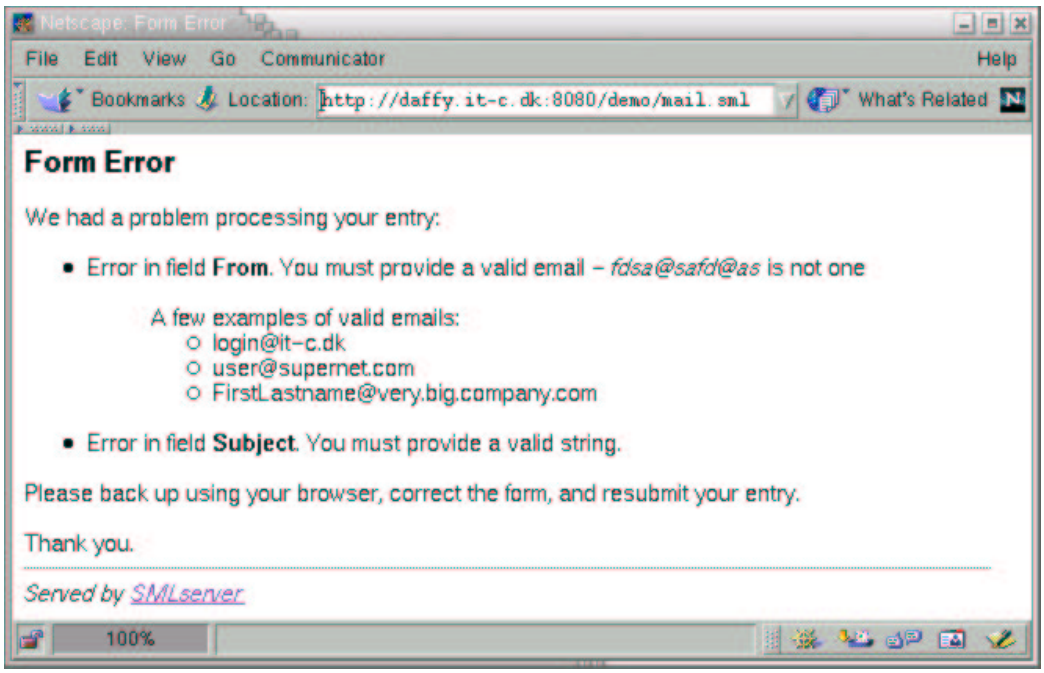


Figure 8.2: When a user submits the email form with invalid entries, such as an invalid email address and an empty subject field, the user is presented with an error page that summarizes all errors.

the `Ns.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/demo/formvar_chk.sml`.

8.3 Implementation

The `FormVar` structure is based on the function `Ns.Conn.formvar`, which provides a more primitive way of accessing form variables submitted with a request. The function `Ns.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 `Ns.Conn.formvar` function, the implementation of the `FormVar` structure also makes use of regular expressions (see Section 6.2).

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

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. (See Appendix C for information on setting up SSL with SMLserver.)

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.[1]

[1] Authentication mechanisms entirely based on form variables requires a user to login to

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

The function `setCookie` returns a cookie formatted string to be included in the header part of an 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.[2] The life time of a cookie with no `expiry` attribute is the user's session only.

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.

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

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`.

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 `Ns.Cookie.allCookies`:

```
val cookies =
  foldl (fn ((n,v),a) => `<li> ^n : ^v ` ^^ a)
  `` (Ns.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 `Ns.write`:[3]

```
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
```

[3]File `smlserver_demo/www/demo/cookie_set.sml`.

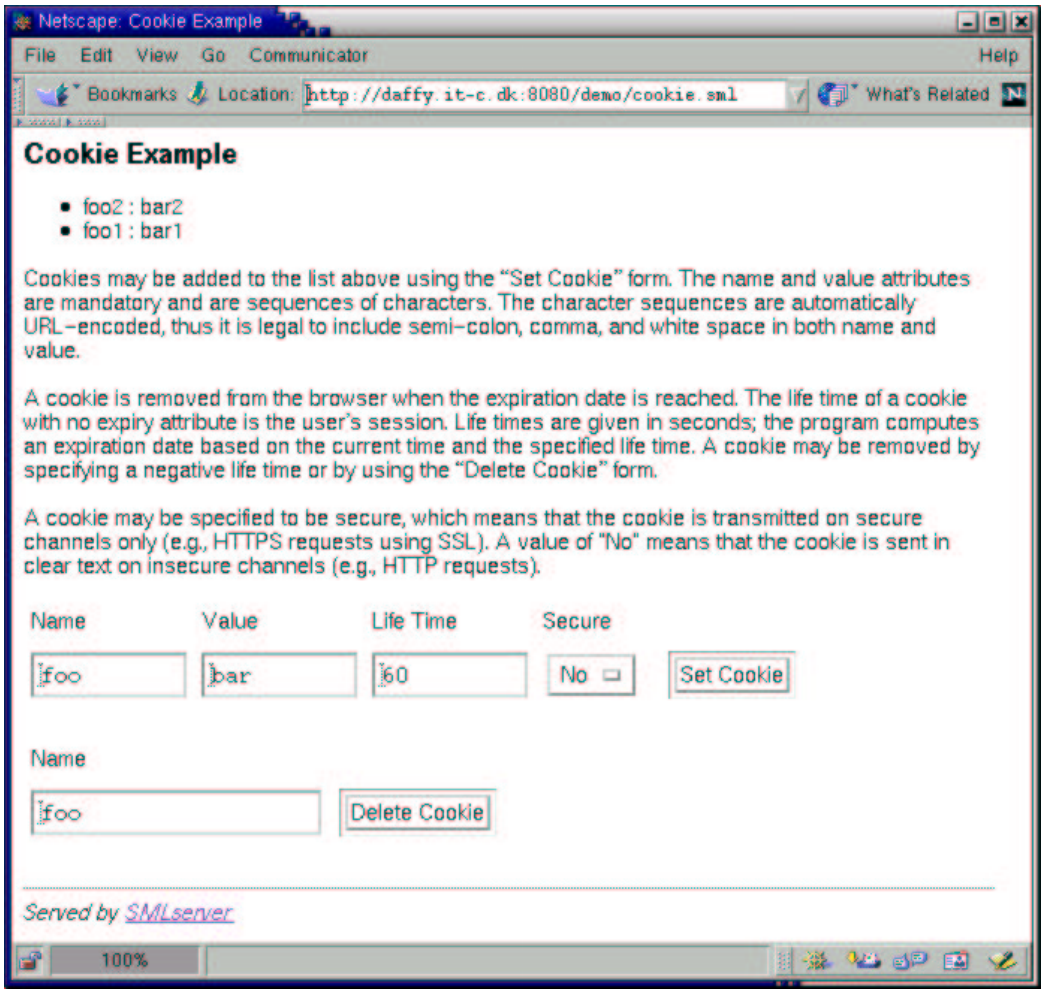


Figure 9.1: The result of a user requesting the file `cookie.sml` with two cookies `foo1` and `foo2`.

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

val _ = Ns.write
`HTTP/1.0 302 Found
Location: /demo/cookie.sml
MIME-Version: 1.0
^cookie

You should not be seeing this!`
```

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. Because HTTP with status code 302 is returned, the content following the HTTP headers is ignored.

The action of the "Delete Cookie" form is the script `cookie_delete.sml`:[4]

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

val _ = Ns.write
`HTTP/1.0 302 Found
```

[4]File `smlserver_demo/www/demo/cookie_delete.sml`.

```
Location: /demo/cookie.sml
MIME-Version: 1.0
^(Ns.Cookie.deleteCookie{name=cn,path=SOME "/"})

You should not be seeing this!‘
```

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:[5]

```
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.dina.kvl.dk/~mael');
```

[5] File `smlserver_demo/demo_lib/pgsql/person.sql`.

```
insert into person (person_id, password, email, name, url)
values (2, 'Niels', 'nh@it.edu', 'Niels Hallenberg',
       'http://www.it.edu/~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 (`.sml`-files) for managing the issuing of passwords, sending passwords to users, serving login forms to users, and so on:[6]

- `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

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

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.

functionality for checking whether a user is logged in (functions `verifyPerson` and `isLoggedIn`), for issuing passwords (function `newPassword`), and so on:[7]

```
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 cookie values `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 (Ns.Cookie.getCookieValue "auth_person_id",
         Ns.Cookie.getCookieValue "auth_password")
    of (SOME person_id, SOME psw) =>
```

[7] File `smlserver_demo/demo_lib/Auth.sml`.

```
        (case getPasswd person_id
           of NONE => NONE
            | SOME db_psw =>
             if db_psw = psw then Int.fromString person_id
             else NONE
        )
       | _ => NONE
  ) handle Ns.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, all that is needed is to modify the function `verifyPerson` as follows:

```
fun verifyPerson() =
  let fun f p = case Db.zeroOrOneField
                       `select password from person
                        where person_id = ^p`
                 of SOME pw => pw
                  | NONE => ""
      fun g p =
        case Ns.Cache.cacheWhileUsed (f, "auth", 600) p
          of "" => NONE
           | pw => SOME pw
  in verifyPerson0 g
  end
```

For a discussion of the function `Ns.Cache.cacheWhileUsed`, 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 overwrite the cache appropriately when users modify their passwords. This overwriting may be implemented using a combination of the functions `Ns.Cache.findTm` and `Ns.Cache.set`, presented in Section 6.6 on page 53.

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:[8]

```
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)

[8] File `smlserver_demo/demo_lib/pgsql/link.sql`.

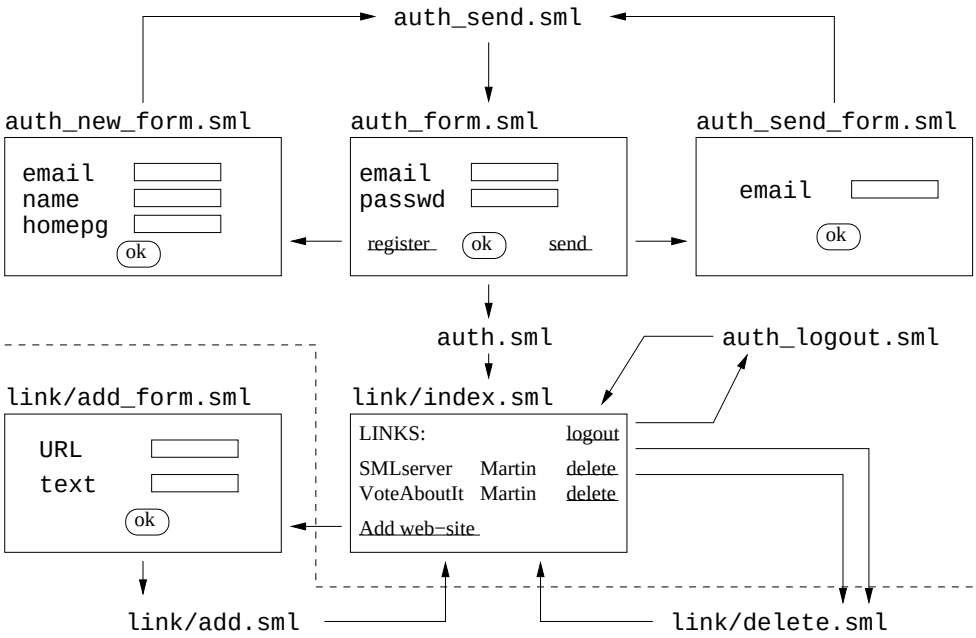


Figure 9.3: Web site diagram for the link database. SMLserver scripts pictured under the dashed line are restricted to users that are logged in; the other SMLserver scripts are accessible for all visitors.

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.[9] The script `link/index.sml`, which is the most involved of the scripts, is implemented as follows:[10]

```
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`

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
```

[9] The directory `smlserver_demo/www/demo/link/` holds all involved scripts.
[10] File `smlserver_demo/www/link/index.sml`.

```
        <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 SML code:[11]

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

[11] File `smlserver_demo/www/link/delete.sml`.

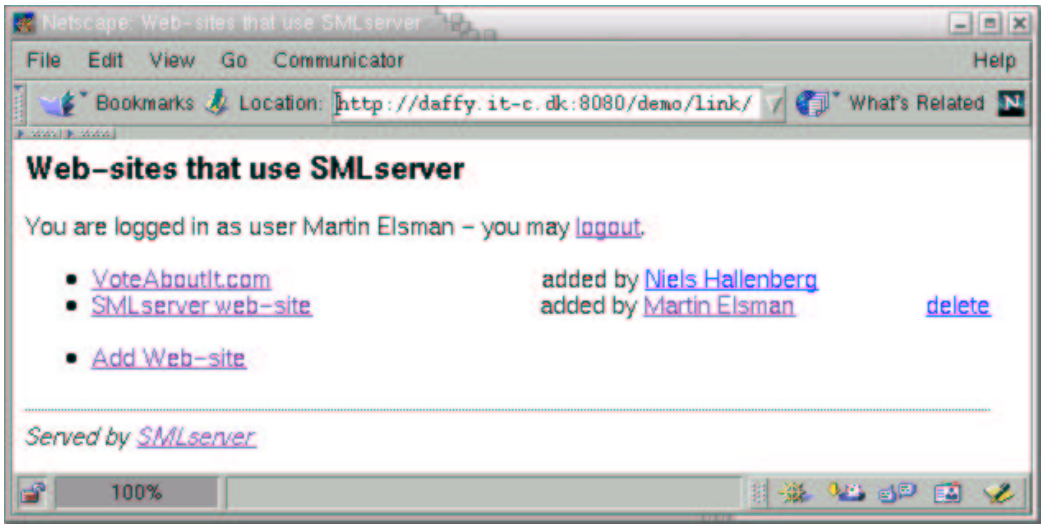


Figure 9.4: The result of a user requesting the file `link/index.sml`.

```
val _ = Db.dml delete
val _ = Ns.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.

Chapter 10

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.

The authors are currently working on a series of composable modules, called the SMLserver Community Suite (SCS), for building customizable Web sites with SMLserver. Modules in the suite include a module for verifying user submitted form content based on a large set of form variable types, a module for constructing and managing multilingual Web sites, and a generic authentication module.

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 is the possibility (using the SMLserver API) of scheduling execution of scripts to run at a particular time in the future. Similarly, it is currently not possible to arrange for periodic execution of

scripts using the SMLserver API.

Bibliography

[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

A Sample Web Server Configuration File

A sample AOLserver configuration file, to be read by AOLserver when it starts, is given below.[1] It should be necessary to change only the first five or six lines, provided Postgresql and the Postgresql driver for AOLserver are installed on your system.

```
#---------------------------------------
# Sample AOLserver configuration file
# with SMLserver and Postgresql support
#---------------------------------------

set port 8080
set user yourlogin
set webdir /home/${user}/web
set pg_passwd XXXX
set nssml_so ../../smlserver/bin/nssml.so

set home /usr/share/aolserver

set host [ns_info hostname]
set bindir [file dirname [ns_info nsd]]

ns_section "ns/mimetypes"
```

[1] File `smlserver_demo/nsd.tcl`.

```
ns_param .wml text/vnd.wap.wml
ns_param .wbmp image/vnd.wap.wbmp
ns_param .wmls text/vnd.wap.wmlscript
ns_param .wmlc application/vnd.wap.wmlc
ns_param .wmlsc application/vnd.wap.wmlscriptc

ns_section "ns/parameters"
ns_param debug off
ns_param Home $home
ns_param serverlog ${webdir}/log/server.log
ns_param pidfile ${webdir}/log/nspid.txt
ns_param user ${user}
ns_param stacksize 500000

ns_section "ns/servers"
ns_param ${user} "${user}'s server"

ns_section "ns/server/${user}"
ns_param directoryfile "index.sml"
ns_param pageroot ${webdir}/www
ns_param enabletclpages off

ns_section "ns/server/${user}/module/nslog"
ns_param file ${webdir}/log/access.log

ns_section "ns/server/${user}/module/nssock"
ns_param port ${port}
ns_param hostname $host

ns_section "ns/server/${user}/module/nssml"
ns_param prjid sources

#
# Database drivers
#
ns_section "ns/db/drivers"
ns_param postgres /usr/share/pgdriver/bin/postgres.so
```

```
ns_section "ns/db/pools"
ns_param pg_main "pg_main"
ns_param pg_sub "pg_sub"

ns_section "ns/db/pool/pg_main"
ns_param Driver postgres
ns_param Connections 5
ns_param DataSource localhost::${user}
ns_param User ${user}
ns_param Password ${pg_passwd}
ns_param Verbose Off
ns_param LogSQLErrors On
ns_param ExtendedTableInfo On

ns_section "ns/db/pool/pg_sub"
ns_param Driver postgres
ns_param Connections 5
ns_param DataSource localhost::${user}
ns_param User ${user}
ns_param Password ${pg_passwd}
ns_param Verbose Off
ns_param LogSQLErrors On
ns_param ExtendedTableInfo On

ns_section "ns/server/${user}/db"
ns_param Pools pg_main,pg_sub
ns_param DefaultPool "pg_main"

ns_section "ns/server/${user}/modules"
ns_param nssock nssock.so
ns_param nslog nslog.so
ns_param nssml ${nssml_so}
```

Appendix B

SMLserver and MySQL

If you do not have a truck load of money to buy Oracle then we recommend Postgresql, which has become a fairly stable and reliable database server. An example of its use is OpenACS (`www.openacs.org`), a large community system implemented on a Postgresql database server using AOLserver. Another option is MySQL, but SMLserver does not support MySQL as elegantly as Postgresql and Oracle because MySQL does not support sequences and transactions, and thus does not pass the ACID test (Section 7.2 on page 61).

If you choose to use MySQL anyway, then this appendix emphasizes what functions in the database interface `NS_DB` do not work with MySQL, and how you may work around the shortcomings. Some of the functions that are not supported for MySQL are shown in Figure B.1.

MySQL does not support transactions. In particular, the `dmlTrans` function is undefined if used with MySQL.

Moreover, MySQL does not support Oracle and Postgresql style sequences, for which there is support in the `NS_DB` signature (functions `seqNextval`, `seqNextvalExp`, `seqCurrvalExp` and `seqCurrval`). Instead, MySQL has support for an auto increment mechanism, which leaves two ways to program sequences in MySQL. The first way uses the *auto increment* feature of MySQL, with which sequence numbers are created when a row is inserted in a table. The second way simulates the traditional Oracle sequences where a sequence number is generated from a sequence generator and then separately inserted in a table. In the following, we discuss the two ways in turn.

```
signature NS_DB =
  sig
    val dmlTrans      : (db -> 'a) -> 'a

    val seqNextvalExp : string -> string
    val seqNextval    : string -> int
    val seqCurrvalExp : string -> string
    val seqCurrval    : string -> int
  end
```

Figure B.1: Parts of the `NS_DB` signature.

B.1 Auto Incrementation

The traditional MySQL way of generating unique keys is quietly supported by the function `seqNextvalExp`. Consider the table `link` from the link database example:[1]

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

The field `link_id` is implemented in the MySQL style using the auto increment feature. A new row is inserted in the table with the use of the `seqNextvalExp` function:[2]

```
val insert =
  `insert into link (link_id, person_id, url, text)
   values (^(Db.seqNextvalExp "link_seq"),
           ^(Int.toString Login.person_id),
           ^(Db.qqq url),
           ^(Db.qqq text))`
```

[1] File `smlserver_demo/demo_lib/mysql/link.sql`.
[2] File `smlserver_demo/www/demo/link/add.sml`.

```
val _ = Db.dml insert
```

The name `link_seq` is important for Oracle and Postgresql, which uses explicit sequences, but is ignored when using MySQL. The function `seqNextvalExp` always returns the string `"null"` when using MySQL, and a new number is created by MySQL (because of the `auto_increment` declaration in the `create table` statement).

B.2 Sequence Simulation

The traditional Oracle version with explicit sequences can be simulated with an extra table and the function `seqNextval`. Consider the Best Wine Web site, which builds on the following table definitions:[3]

```
create table wid_sequence (
  seqId integer primary key auto_increment
);

create table wine (
       wid       integer primary key,
       name      varchar(100) not null,
       year      integer,
       unique    ( name, year )
);

create table rating (
       wid       integer not null,
       comments  text,
       fullname  varchar(100),
       email     varchar(100),
       rating    integer
);
```

The table `wid_sequence` simulates a sequence generator. The numbers generated are used in the two tables `wine` and `rating`. The following code inserts a new wine in the `wine` table:[4]

[3] File `smlserver_demo/demo_lib/mysql/rating.sql`.
[4] File `smlserver_demo/www/demo/rating/add.sml`.

```
val (wid, name, year) =
  ...
    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))`
    in
      (wid, name, year)
    end
```

A fresh wine identification number (`wid`) is generated by `seqNextval` using the name of the table simulating sequences (`wid_sequence`). The function `seqNextval` assumes that the field in table `wid_sequence` is named `seqId`. The number generated (stored in variable `wid`) is then used when inserting a row in the `wine` table.

The function `seqCurrval` *seqName* returns the last generated number in table *seqName*. The function `seqCurrvalExp` does not work with MySQL.

Appendix C

Securing Your Site with SSL

This appendix introduces the Secure Socket Layer (SSL). Information on how to install SSL on AOLserver can be found at SMLserver's home page: `http://www.smlserver.org/inst/ssl.html`.

SSL runs below the higher-level HyperText Transport Protocol (HTTP) and on top of the lower-level Transmission Control Protocol/Internet Protocol (TCP/IP), see Figure C.1. Thus, as we shall see, the use of SSL does not show through at the SML level, only at the AOLserver level.

The TCP/IP protocol controls the sending and receiving of data packets between two computers on the Internet. The HTTP protocol uses TCP/IP to implement the communication between Web servers and clients (browsers). With SSL, the communication between a client and a Web server is encrypted and the Web server is by default authenticated to the client.

SSL uses public-key encryption to establish a connection (called the SSL Handshake) and uses symmetric key encryption after the connection is established. Symmetric key encryption is faster than public-key encryption. As a result of the SSL handshake, the client and the Web server agrees to use a pair of symmetric keys (the shared secret) to encrypt future messages.

In the following we summarize the SSL Handshake assuming that RSA Key Exchange is used (many details are left out).

1. The client sends a `client_hello` message containing the SSL version number supported by the client (e.g., v2 and v3), the supported cipher algorithms (e.g., RSA), and some randomly generated data.

2. The server responds with a `server_hello` message containing the SSL

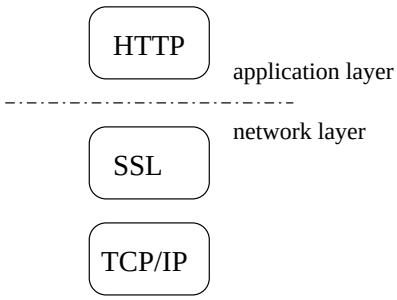


Figure C.1: The SSL layer runs on top of TCP/IP that implements the sending and receiving of data packets between two computers on the Internet. The SSL protocol encrypts the communication.

version number supported by the server, the supported cipher algorithms, and some randomly generated data.

3. The server sends its own `certificate` and a `server_done_message` indicating that the server now waits for a client response.

4. The client verifies the server certificate (Server Authentication). The certificate contains among others: (1) the server's public key (*pub_key*), (2) the *server_name* (e.g., `www.company.com`), (3) a validity period and (4) information about the Certificate authority (CA) that has signed the certificate.

5. The client sends a `client_key_exchange` message containing: a 48 byte *pre-master secret* encrypted with the public key from the server certificate, *pub_key*.

6. The client computes the *shared master secret*. The client sends a `change_cipher_spec` message to the server containing various parameters computed using the *shared master secret*. These parameters are used by the server to decrypt future messages from the client.

7. The client sends a `finished` message to the server.

8. The server computes the same *shared master secret* based on the *pre-master secret* received from the client. The server also computes various

parameters using the *shared master secret* and sends them to the client using a `change_cipher_spec` message. These parameters are used by the client to decrypt future messages from the server.

9. The server then sends a `finished` message to the client and the SSL Handshake is completed.

During the SSL Handshake the client verifies the certificate received from the server, called Server Authentication. Server Authentication is summarized below:

1. The client checks that today's date is within the validity period contained in the certificate.

2. The client checks that the Certificate Author (CA) is trusted. Most browsers trust several CAs by default.[1] A user may change the list of trusted CAs. For instance, a company may choose to be its own CA.

   A certificate issued by a CA says that the CA guarantees that the *server_name* (`www.company.com`) and the public key *pub_key* of the server are tied together (i.e., if the server can decrypt a message encrypted with *pub_key*, then the server has proved that it is the server named *server_name*). The CA encrypts this information in the certificate with its private key. This often requires the company to mail documents confirming the right to use the company name and that the company owns the *server_name*. The CA issues a certificate only when they have received thorough documentation. Example CAs are Verisign (`www.verisign.com`) and Thawte (`www.thawte.com`).

3. The client checks that the CA's public key can be used to validate the signature in the certificate.

4. The client checks that the domain name in the certificate matches the *server_name* of the Web server.

It is also possible to have clients authenticate themselves to a Web server, but this kind of authentication, which is normally not used on the Internet

[1] On Netscape Communicator 4.77 a list of trusted CAs can be seen by choosing `Communicator::Tools::Security Info` in the menu and then click `Certificates::Signers`

(but sometimes on Intranets where the users are known), requires the use of client certificates. SMLserver does not support client certificates.

Below is a link to related and more in-depth literature:

- The SSL Protocol, version 3.0 is explained in the Internet-draft, `http://home.netscape.com/eng/ssl3/draft302.txt`.
- The Transport Layer Security protocol (TLS) which is basically SSL version 3.0 with a few minor differences is explained in RFC2246, `http://www.ietf.org/rfc/rfc2246.txt?number=2246`.
- There is an article about SSL and AOLserver at ArsDigita written by Scott S. Goodwin who is also releasing the SSL module for AOLserver, `http://www.arsdigita.com/asj/aolserver-ssl.adp`.
- A fairly detailed introduction to SSL can be found at iPlanet.com, `http://docs.iplanet.com/docs/manuals/security/sslin/` together with an introduction to public-key cryptography, `http://docs.iplanet.com/docs/manuals/security/pkin/index.htm`.

Appendix D

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.

D.1 Elements Supported Inside Body Element

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

D.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.

D.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.

D.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.

D.1.4 Headers

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

D.1.5 Logical Styles

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

D.1.6 Physical Styles

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

D.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.

D.1.8 Unordered Lists

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

D.1.9 Ordered Lists

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

D.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 at

`http://www.w3.org/pub/WWW/MarkUp/html-spec/html-spec_9.html`

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

D.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).

D.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`.

D.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.

D.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>
```

D.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.

D.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 E

The Ns Structure

The Ns structure gives access to the Web server API. The structure name Ns stands for Navi Server, which was the name of the AOLserver Web server before it was purchased by America Online.

The following sections present the signatures NS_SET, NS_INFO, NS_CACHE, NS_CONN, NS_MAIL, NS_COOKIE, and NS. The structure Ns implements the NS signature, which holds sub-structures matching each of the other listed signatures. A summary of the NS_DB signature is presented in Figure 7.2 on page 70.

E.1 The NS_SET Signature

```
signature NS_SET = sig
  type set
  val get    : set * string -> string option
  val getOpt : set * string * string -> string
  val getAll : set * string -> string list
  val size   : set -> int
  val unique : set * string -> bool
  val key    : set * int -> string option
  val value  : set * int -> string option
  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.

 [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.

 [unique (s,k)] returns true if key k appears (exactly) once
 in s (case sensitive). Returns false otherwise.

 [key (s,i)] returns SOME(k) if k is the key name for the
 i'th field in the set s; returns NONE if size s <= i.

 [value (s,i)] returns SOME(v) if v is the value for the
 i'th field in the set s; returns NONE if size s <= i.

 [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).
```

```
*)
```

E.2 The NS_INFO Signature

```
signature NS_INFO = sig
  val configFile          : unit -> string
  val configGetValue      : {sectionName: string,
                             key: string} -> string option
  val configGetValueExact : {sectionName: string,
                             key: string} -> string option
  val errorLog            : unit -> string
  val homePath            : unit -> string
  val hostname            : unit -> string
  val pid                 : unit -> int
  val uptime              : unit -> int
  val pageRoot            : unit -> string
end

(*
 [configFile()] returns the location of the configuration
 file.

 [configGetValue{sectionName,key}] returns SOME s, if s is
 the string associated with the (sectionName, key) pair in
 the configuration file. Returns NONE, otherwise. Case
 insensitive on sectionName, key.

 [configGetValueExact{sectionName,key}] as configGetValue,
 but case sensitive.

 [errorLog()] returns the location of the log file.

 [homePath()] returns the directory where the Web server is
 installed.

 [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.

 [pageRoot()] returns the directory for which the server
 serves pages.
*)
```

E.3 The NS_CACHE Signature

```
signature NS_CACHE = sig
  type cache
  val createTm  : string * int -> cache
  val createSz  : string * int -> cache
  val find      : string -> cache option
  val findTm    : string * int -> cache
  val findSz    : string * int -> cache
  val flush     : cache -> unit
  val set       : cache * string * string -> bool
  val get       : cache * string -> string option

  val cacheForAwhile : (string -> string) * string * int
                       -> string -> string
  val cacheWhileUsed : (string -> string) * string * int
                       -> string -> string
end

(*
 [cache] abstract type of cache.

 [createTm (n, t)] creates a cache, given a cache name n and
 a cache timeout value t in seconds.

 [createSz (n, sz)] creates a cache, given a cache name n
 and a maximum cache size sz in bytes.
```

```
 [find n] returns a cache, given a cache name n. Returns
 NONE if no cache with the given name exists.

 [findTm (cn,t)] as find, except that the cache with name cn
 is created if it does not already exist. If the cache is
 created then t is used as cache timeout value in seconds.

 [findSz (cn,s)] as find, except that the cache with name cn
 is created if it does not already exist. If the cache is
 created then s is used as cache size in bytes.

 [flush c] deletes all entries in cache c.

 [set (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.

 [get (c,k)] returns value associated with key k in cache c;
 returns NONE if key is not in cache.

 [cacheForAwhile (f,cn,t)] where f is a function, cn is a
 cache name, and t a cache timeout value in seconds. Returns
 a new function f' equal to f except that its results are
 cached and only recalculated when the cached results are
 older than the timeout value. This function can be used, for
 instance, to cache fetched HTML pages from the Internet. The
 timestamp is not renewed when items are accessed.

 [cacheWhileUsed (f,cn,t)] as casheForAwhile, except that
 the timestamp is renewed at each access. An item is removed
 from the cache if t seconds have passed after the last
 access.
*)
```

E.4 The NS_CONN Signature

```
signature NS_CONN = sig
  eqtype status
  type set
  val returnHtml     : int * string -> status
  val return         : string -> status
  val write          : string -> status
  val returnRedirect : string -> status
  val getQuery       : unit -> set option
  val formvar        : string -> string option
  val formvarAll     : string -> string list
  val headers        : unit -> set
  val host           : unit -> string
  val location       : unit -> string
  val peer           : unit -> string
  val peerPort       : unit -> int
  val port           : unit -> int
  val redirect       : string -> status
  val server         : unit -> string
  val url            : unit -> string
end

(*
 [status] abstract type identical to Ns.status.

 [set] abstract type identical to Ns.Set.set.

 [returnHtml (sc,s)] sends HTML string s with status code sc
 to client, including HTTP headers. Returns Ns.OK on success
 and Ns.ERROR on failure.

 [return s] sends HTML string s with status code 200 to
 client, including HTTP headers. Returns Ns.OK on success
 and Ns.ERROR on failure.

 [returnFile (sc,mt,f)] sends file f with status code sc to
 client, including HTTP headers. The mime type is mt.
```

```
Returns Ns.OK on success and Ns.ERROR on failure.

[write s] sends string s to client, excluding HTTP headers.
Returns Ns.OK on success and Ns.ERROR on failure.

[returnRedirect loc] sends redirection HTTP response to
client (status code 302), with information that the client
should request location loc. Returns Ns.OK on success and
Ns.ERROR on failure.

[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 query data associated with the
connection and 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.

[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. Multiple
communications drivers can be loaded into a single server.
This means a server may have more than one location. For
example, if the nsssl module is loaded and bound to port
8000 and the nssock module is loaded and bound to port 9000,
the server would have the following two locations:
     http://www.avalon.com:9000
     https://www.avalon.com:8000
```

```
 For this reason it is important to use the location function
 to determine the driver location at run time.

 [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.
*)
```

E.5 The NS_MAIL Signature

```
signature NS_MAIL = sig
  val sendmail : {to: string list, cc: string list,
                  bcc: string list, from: string,
                  subject: string, body: string,
                  extra_headers: string list} -> unit
  val send     : {to: string, from: string,
                  subject: string, body: string} -> unit
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.
*)
```

E.6 The NS_COOKIE Signature

```
signature NS_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 -> string
  val setCookies     : cookiedata list -> string
  val deleteCookie   : {name: string, path: string option}
                       -> string
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.
*)
```

E.7 The NS Signature

```
signature NS = sig
  (* logging *)
  eqtype LogSeverity
  val Notice  : LogSeverity
  val Warning : LogSeverity
  val Error   : LogSeverity
  val Fatal   : LogSeverity
  val Bug     : LogSeverity
  val Debug   : LogSeverity
  val log     : LogSeverity * string -> unit

  (* status codes *)
  eqtype status
```

```
  val OK       : status
  val ERROR    : status
  val END_DATA : status

  (* various functions *)
  type quot = Quot.quot
  val return         : quot -> status
  val write          : quot -> status
  val returnHeaders  : unit -> unit
  val returnRedirect : string -> status
  val getMimeType    : string -> string
  val getHostByAddr  : string -> string option
  val encodeUrl      : string -> string
  val decodeUrl      : string -> string
  val buildUrl       : string -> (string * string) list
                       -> string
  val fetchUrl       : string -> string option
  val exit           : unit -> 'a

  (* sub-structures *)
  structure Set      : NS_SET
  structure Conn     : NS_CONN where type status = status
                                 and type set = Set.set
  structure Cookie   : NS_COOKIE
  structure Cache    : NS_CACHE
  structure Info     : NS_INFO
  structure Mail     : NS_MAIL
  structure DbOra    : NS_DB where type status = status
                            and type set    = Set.set
  structure DbPg     : NS_DB where type status = status
                            and type set    = Set.set
  structure DbMySQL  : NS_DB where type status = status
                            and type set    = Set.set
end

(*
 [LogSeverity] abstract type of log severity.
```

[Notice] something interesting occurred.

[Warning] maybe something bad occurred.

[Error] something bad occurred.

[Fatal] something extremely bad occurred. The server shuts down after logging this message.

[Bug] something occurred that implies there is a bug in your code.

[Debug] if the server is in Debug mode, the message is printed. Debug mode is specified in the [ns/parameters] section of the configuration file. If the server is not in debug mode, the message is not printed.

[log (ls,s)] write the string s to the log file with log severity ls.

[status] abstract type of status code returned by functions.

[OK] status code indicating success.

[ERROR] status code indicating failure.

[END_DATA] status code indicating end of data.

[quot] type of quotations.

[return q] sends HTML string q to browser with status code 200, adding HTTP headers. Returns OK on success and ERROR on failure.

[write q] sends string q to browser. Returns OK on success and ERROR on failure.

[returnHeaders()] sends HTTP headers to browser.

[returnRedirect loc] sends a redirection HTTP response to location loc. Returns OK on success and ERROR on failure.

[getMimeType f] guesses the Mime type based on the extension of the filename f. Case is ignored. The return value is of the form "text/html".

[getHostByAddr ip] converts a numeric IP address ip into a host name. If no name can be found, NONE is returned. Because the response time of the Domain Name Service can be slow, this function may significantly delay the response to a client.

[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.

[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.

[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.

[exit()] terminates the script by raising the exception Interrupt, which is silently caught by the SMLserver module (other uncaught exceptions are logged in the server.log file).

```
*)
```

Index