ROXEN™ CHALLENGER MANUAL

Detta dokument har framställts med FrameMaker 4.0.4

First printing: October 1996.

TABLE OF CONTENTS

Detta dokument har framställts med FrameMaker 4.0.4

ROXEN USER'S GUIDE 31

INTRODUCTION TO ROXEN™ CHALLENGER

Detta dokument har framställts med FrameMaker 4.0.4

CHAPTER 1

WELCOME TO ROXEN™ CHALLENGER!

Congratulations on your acquisition of Roxen™ Challenger, the best information management tool you can find on the market today! In this part of the guide we will try to safely guide you through the possible intricacies of installing and setting up Roxen™ Challenger, no matter what your previous experience of web server software.

GETTING IN TOUCH WITH US

If you need to get in touch with us or simply wish to express your opinions of Roxen™ Challenger, the easiest way is by sending an e-mail to us at info@infovav.se, or if it specifically concerns Roxen™ Challenger, to roxen@infovav.se. Of course you can send us ordinary mail too:

InformationsVävarna AB
Skolgatan 10
S-582 35 Linköping
SWEDEN

If you have bought support and live in the USA, call 800-345-0046 and ask for "Collect to Sweden 013-37 68 10". If you live in Sweden, you can call 013-37 68 10 directly.

By the way; as you can see in our domain name the short form of *InfomationsVävarna* is *Infovav*. This abbreviation will be used hereafter in the manual.

ROXEN, THE STORY

In the beginning there weren't many webservers around (quite naturally!) and those there were, weren't very good. The growth of the World Wide Web (WWW) interested and attracted a large number of members of the Lysator Computer Society at the Linköping University. Lysator set up the world's tenth WWW-server, a server which today has around 500,000 accesses per day (September, 1996). Of course it has run Roxen since the early days.

SPIDER

One member was Per Hedbor. He wasn't satisfied with the performance of other servers and, accordingly, he wrote his own, a very small, very buggy, but working C program, launched from inetd[1]. Per named his creation Spider.

After a while, he moved on to LPC4, a language with excellent string and socket support, which made it ideal for writing WWW-servers. This language was also created by a member of Lysator, initially intended for use as a tool when building muds.

As Spider evolved, it became more and more complex, and even more features were added. It might be said that this program is, in the same way as the NCSA httpd and the CERN-server, suffering from a severe case of feeping creaturism[2], but in a more pronounced way. The program has been conceived with extensibility in mind; it is quite easy to extend the functionality of Roxen™ Challenger.

The first version went on-line in November 1993. Back then it was a quick and dirty hack, but it worked, and it had a few extra features that were used at Lysator, like certain special pathnames, i.e. /~{name}, i.e. automagically[3] generated information for the user {name}. This was

1. A daemon in the Unix operating system.
2. This is a hacker term for a steadily increasing amount of features and the problems this entails.
3. Since programmers like to accomplish complicated feats of programming this term has arisen to signify something that the program does automatically, something so immensely complicated that to the layman it can be likened to magic. Oh well.

done without CGI scripts. As a matter of fact, CGI didn't even exist, or it had at least not been publically announced.

This, and the fact that a lot of the information published by Lysator and its members had to be preprocessed[4] made it hard to adapt an existing server. Therefore, Per continued his work, and other people became interest in the development as well. Soon, Spider 2 was born, containing even more features and also the occasional bug fix.

SPINNER

By the end of 1994, Per decided to make a new version from scratch. He left LPC4 in favour of µLPC, a language inspired by LPC4. Pike is a lot more general and even better suited for network programming than LPC4. Another reason was that LPC4 has very restrictive rules concerning its use in commercial products. Pike is a professional programming language with many features that make it extremely easy to learn and use in a professional way.

Instead of using one huge executable file, Spider 3 would be based on modules. When the first version of Spider 3 went online, everyone was surprised by it's extreme speed, compared to Spider 2.

In the first part of 1995, Spider 3 changed name to Spinner and the graphical WWW based configuration interface started to work, although it was a pain to use it, especially when a lot of changes had to be made.

As 1995 progressed, we (InformationsVävarna) became aware of Spinner's probable market potential. Since then, Per and a host of other programmers have been developing Spinner, steadily making it more stable and versatile.

ROXEN™ CHALLENGER

When nearing the release date we unfortunately had to change the name again, this time due to the abundance of software and software-related products named Spinner, not to mention copyright-related issues. We decided to rename Spinner to Roxen™ Challenger.

4. Dynamic pages are easier to administer. You let the server process a few pages and some accessory data instead of maintaining huge amounts of static HTML pages yourself.

In addition we have made available a new version of µLPC and changed that name too; Pike is the new one.

Now you hold the first commercial release of Roxen™ Challenger in your hands. The graphical configuration interface has been rewritten from scratch and the bugs have been exterminated. Many routines have been rewritten in C, for reasons of speed.

Welcome, and good luck!

CREATORS

PROGRAMMING

Per Hedbor, David Hedbor, Mattias Wingstedt, Pontus Hagland and Peter Bortas.

Roxen is mostly written in Pike, a language designed and developed by Fredrik Hübinette.

MANUAL

Johan Mellberg, Per Hedbor, David Hedbor, Tobias Karlsson, Peter Bortas and Fredrik Hübinette.

QUALITY CONTROL

Erik Persson.

PROJECT MANAGERS

Johan Mellberg and Lars Mårelius.

THANKS

Many thanks to Mikael Widenius and Pete Ashdown for their valuable suggestions and contributions. Many others, too many to mention, have also contributed. Thank you!

Many thanks to those who took the time to comment on the first versions of this manual; Francesco Chemolli, Karin Fransson, Linus Tolke and of course all the others.

HARDWARE REQUIREMENTS

MEMORY

To run Roxen you should have at least 8MB of real memory available.

CPU

At least a 386/33 or equivalent (e.g. a Sun3/200) is recommended to experience acceptably smooth performance. This depends heavily on the amount of traffic you need to take care of. At Infovav we have a 512 kbps connection to the Internet. Running Roxen on a Sun Sparcstation 4 under Solaris 5.1 we can theoretically serve around 3 million people per day. The limit is the operating system network routines, not Roxen itself.

HARD DISK SPACE

The binary distribution of Roxen requires around 3MB of disk space, and the full source version approximately 6MB. While compiling you will probably need about another 3MB

SOFTWARE REQUIREMENTS

OPERATING SYSTEMS

Most SysV or BSD Unixes, including Solaris 1.0 or later, Linux 1.1.57 or later and SGI Irix.

ABOUT THE MANUAL

Text with fixed width is used extensively throughout the manual to represent `examples of code`, `lead text in the configuration interface` and `variable values` here and there.

Bolded text represents URL:s, e-mail addresses and paths.

When you have to fill in a value we almost always indicate this by writing

a dummy value between »...«. Replace the dummy value and the »...« with the proper value.

Buttons are written using SMALLCAPS.

The first time we use a word or concept it will be in *italic*.

For easier reading and to shorten the manual by about 15 percent we only write ”Roxen” instead of ”Roxen™ Challenger”. Almost always when the text reads ”Roxen”, it means ”Roxen™ Challenger”.

When referring to configuration files and external files necessary for Roxen’s operation, we most often provide the path relative to the server/ directory. Sometimes we provide the full path, though.

Normally, values of attributes to tags should be enclosed in quotation marks. However, when there is no whitespace in the value (a single word instead of several words for instance) it is not absolutely essential to include the quotation marks. We have not been consistent in the manual, but that goes to show you that there are often several ways to do things.

CHAPTER 2

INSTALLING ROXEN

If there is a binary distribution available for your computer, get that and move on to “Installing a binary distribution”. Otherwise, get the source distribution and move on to “Installing the source distribution”.

INSTALLING A BINARY DISTRIBUTION

- cd to the directory where you want Roxen to be installed and place the archive there, typically /usr/ or /usr/www/. The server will run from anywhere in the filesystem, though.
- Issue the command to unpack the archive, depending on the file extension of the archive:
  - .tar.gz
    If you have GNU tar;
    ```
    tar xzf »roxen-archive-name«
    ```
    If you don’t have GNU tar;
    ```
    gunzip < »roxen-archive-name«|tar xf -
    ```
  - .tar.Z
    ```
    uncompress < »roxen-archive-name«|tar xf -
    ```
  - .tar
    ```
    tar xf »roxen-archive-name«
    ```

Now move on to the section on finishing the installation on page 11.

INSTALLING THE SOURCE DISTRIBUTION

- Issue the command to unpack the archive, depending on the file extension of the archive:
  - `.tar.gz`
    If you have GNU tar;
    `tar xzf »roxen-archive-name«`
    If you don't have GNU tar:
    `gunzip < »roxen-archive-name«|tar xf -`
  - `.tar.Z`
    `uncompress < »roxen-archive-name«|tar xf -`
  - `.tar`
    `tar xf »roxen-archive-name«`
- Type `cd Roxen1.0` to change to the new directory. This directory should contain at least four directories: **server/** (Roxen server source), **pike/** (the Pike intepretator), **extern/** (miscellaneous programs used by Roxen) and **tools/** (miscellaneous tools).
- Type `./configure --prefix=»desired roxen location«` followed by

  `make`

  and then

  `make install`

  to build the binaries and copy them to the correct place. If you get a message saying that **bison** couldn't be found, you need to get, compile and install **bison**, available from **ftp://roxen.com/**. **Bison** is necessary for Pike to compile.

When the compilation has been successfully completed , continue with the section on finishing the installation on page 11.

FINISHING THE INSTALLATION

- Type `cd /path_to_roxen/roxen/server/` and start the install script by typing `./install`.
- The installation script will find a free port for the configuration interface and ask you if you are happy with the default values for machine name and IP number. Answer the questions posed by the script.
- If everything worked, connect to the configuration interface and configure the server, otherwise move on to the troubleshooting section, see page 12. For a brief description of a basic configuration see "Configuring Roxen – the first steps" on page 15.
- If you want Roxen to start automatically when the computer has rebooted, add this to one of the startup files:

  ```
  cd /path_to_roxen/roxen/server/;./start
  ```

  To shut down Roxen, just use the KILL button in the configuration interface.

  If you use a flavour of System V Unix, sample start-up scripts are included with your distribution. These scripts use `/etc/init.d`. You can find the scripts in `roxen/tools/init.d_roxen`.

FIGURE 2.1 *The button for shutting Roxen down*

EXAMPLE INSTALLATION SESSION

```
bash$ tar xzf Roxen.tar.gz
bash$ cd Roxen1.0
bash$ mkdir solaris
bash$ cd solaris
bash$ ../configure --prefix=/usr/www
<Lots and lots of checks...>
bash$ make
<Lots and lots of compilations>
bash$ make install
<Lots and lots of compilations>
bash$ cd /usr/www/roxen/server
bash$ ./install
<Answer a few questions>
```

Note that it will take quite some time to compile everything. On a Sun Sparc 5 with 128 Mbytes of memory it took around half an hour. Just let

the compiler run its course. Have some coffee, relax or learn how to program in Pike while waiting.

TROUBLESHOOTING

There may be several reasons why Roxen doesn't work. Listed below are some of the things to check and try in order to diagnose the nature and source of the problem.

1. *First do a* `ps -ax` *or* `ps -ef`.

   If you see processes called:

   ```
   bin/pike -m etc/master.pike roxen
   /bin/roxen
   ```

   then the server is running.

2. *You have to connect to a default port on your machine.*

   The default port is given by the install script (usually 18830). You connect by entering the URL: `http://your.machine.name:18830`. Notice that you can change the global variable `Configuration port`. Study the installation walkthrough earlier in this chapter, and the configuration tutorial in "Configuring Roxen – the first steps" on page 15.

3. *Make sure you have configured at least one virtual server.*

   Roxen does not automatically start serving pages. Instead it serves them through one or several virtual servers. In order to get Roxen to send pages to clients you must configure at least one virtual server. Read more in "Adding a new virtual server." on page 19.

4. *Make sure you have a filesystem module installed.*

   One of the possible default settings of a virtual server does not include a filesystem. However, there must be at least one filesystem module installed and properly configured under one of the virtual servers or Roxen will not respond to any incoming requests. The reason is that without the filesystem module, there is no indication of from where Roxen should get the pages to send.

5. *Check the Roxen log files.*

   The log files reside in `roxen/logs/debug/default.x`, where x is a number between 1 and 3. If any errors are noted, check the searchable mailinglist archive (http://www.roxen.com/)to see if someone else has had the same problem. If you can't seem to find a solution, please send a bug report (roxen-bug@infovav.se) or ask on the Roxen mailinglist (roxen@infovav.se).

DONE INSTALLING?

Once you have installed the server, consider joining the Roxen mailinglist by sending an e-mail to roxen-request@infovav.se with the word `subscribe` on the `Subject` line. Apart from this, the message should be empty. You will also be automatically added if you send mail to the mailinglist itself, roxen@infovav.se.

CHAPTER 3

CONFIGURING ROXEN – THE FIRST STEPS

This chapter is a step-by-step instruction on how to configure Roxen at its most basic level, i.e. to set up a running Roxen server without any exceptional features. By the end of this tutorial, you should also be able to customize Roxen's functionality even further all on your own, using the configuration interface to add or remove modules.

BASIC CONFIGURING

If you haven't run the install script yet, please do so now, see page 11. Start by filling in the text fields on the first page, see figure 3.1 on page 16. The *username* and *password* are needed to stop unauthorized people from accessing the configuration interface. The *IP-pattern* makes it possible to limit configuration access to a few well-known computers , thus ensuring higher security.

Fill in the proper values and press USE THESE VALUES to continue. You will now be prompted for username and password. Fill them in, and press OK to continue. If you are not happy with what you have entered here you can redo it by following the link in the documentary text of the password field under Global Variables / Configuration Interface

As you can see in the example, it is possible to use pattern matching in the IP number. The IP-pattern in figure 3.1 would only allow you to ac-

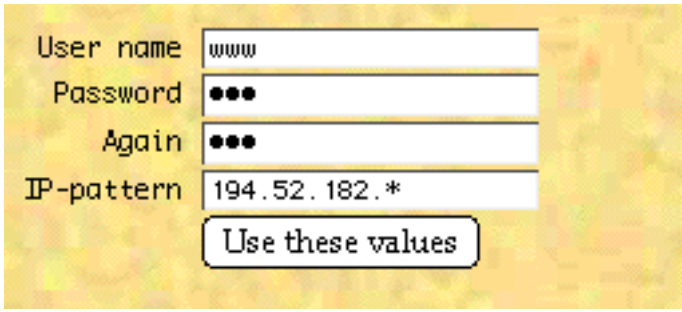

FIGURE 3.1 *The first contact with the Roxen configuration interface.*

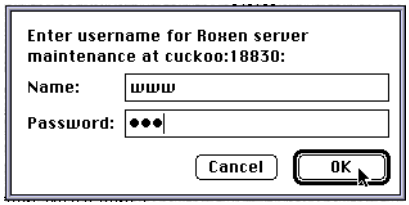

FIGURE 3.2 *The dialog that should greet you before you gain access to the configuration interface proper.*

cess the configuration interface from computers in the domain whose IP numbers begin with 194.52.182.

THE CONFIGURATION INTERFACE

Roxen is now ready to be configured. What follows is a brief explanation of the configuration interface.

The configuration interface looks like a Macintosh directory listing. In figure 3.3 you can see what the top of the screen contains. Except for the SAVE button and possibly some of the images, which you can configure not to be used by setting `Compact layout`, under GLOBAL VARIABLES, in `Configuration interface...` to `Yes`, this is what you will see on all pages of the configuration interface. Clicking on a tab, SERVERS for example, shows you the associated information.

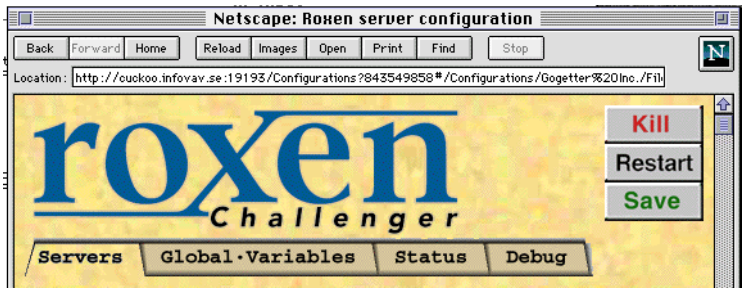

FIGURE 3.3 *Top view of the configuration interface.*

THE FOLD / UNFOLD PRINCIPLE AND FOCUSING

The arrows () are fold / unfold buttons. If you click on one of the arrows pointing right it will change into a down-pointing arrow and all the underlying sub-menus with the associated variables become visible; they *unfold*. Clicking on a down-pointing arrow will fold the menu again.

As you can see it is also possible to click on the main menu items, not only unfold them. Doing this is called *focusing*. Sometimes focusing is necessary to access certain functions, for example when you wish to add new modules to a virtual server you have to focus on the virtual server, or the button NEW MODULE won't appear, see figure 3.12. We consider every clickable item a so-called *node*.

FIGURE 3.4 *When lots of nodes are unfolded, press this button.*

When you have focused, you will see this button:  beside that which is now the top-most visible header, i.e. the *node name*. Clicking on it will get you back up one level in the configuration interface. Clicking on the active tab(SERVERS for example) returns you to an "unfocused" state.

When you have made several changes or descended deep into the configuration hierarchy you can fold all the unfolded sub menus by clicking on FOLD ALL.

FIGURE 3.5 *The Save button.*

COLOUR ENCODING

As you already have noticed, some arrows are red and some are blue. This colour encoding is a great help when you configure Roxen. The red colour is meant to catch your attention, showing you where unsaved changes have been made. If you click the SAVE button, you'll notice that all

the arrows turn blue instead. Clicking on this button saves any changes that you've made.

YOUR FIRST CHANGES

Now when you have begun using the configuration interface, why not try to change something? If you run Roxen as *root*, it might be a good idea to change the *User ID* (uid) and *Group ID* (gid) of the server, after it has started. To do this, unfold *Change uid and gid* under the *Global Variables...* menu, see figure 3.6.

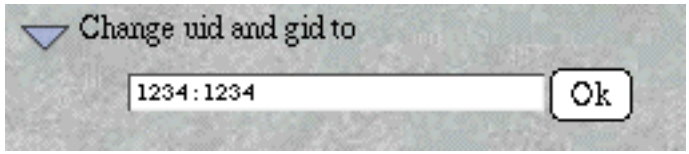

FIGURE 3.6 *Changing the user and group id:s.*

As you might know, only the root of a system is allowed to open the ports up to port 1024. The port used by the http protocol is port 80, thus it has to be opened by someone with root privileges. However, it is not a good idea to have Roxen actually run as root, having access to everything on the system. But if you change `uid:gid` to something else (which is highly recommended), you will have to hit RESTART to ensure that newly added virtual server(s) can use this port.

FIGURE 3.7 *The Restart button.*

The user and group id:s must be numerical values, which means that you cannot use symbolical values (for example www:www). The reason for this is that the change is made before any modules are called, i.e. before Roxen is given the ability ro resolve symbolic id:s.

The server has to be able to write logfiles and save configurations. In order to ensure this, do (as root and in the `server/` directory):

```
chown -R »uid« configurations ../logs
```

and possibly also

```
chgrp -R »gid« configurations ../logs
```

where `uid` and `gid` are the values you set before.

Another thing you might want to change is the *configuration port* which defaults to the one assigned by the install script. The main reason for changing is that it might be hard to remember to use something like port

number 18391.

Press SAVE to save the changes. Notice that you will have to change the URL to the configuration interface manually (i.e. from http://www.whatever.domain:oldport/ to http://www.whatever.domain:newport/), to be able to continue configuring. If you forget to change the configuration URL Roxen will protest.

Do not change the port unless you know for sure that the new port isn't busy. If that is the case, Roxen will stop and you will not be able to continue configuring. For now, the easy workaround is to install a new Roxen and try to use the desired port for this copy. If the install script doesn't protest, KILL and remove this temporary Roxen, then connect to the configuration interface of the copy of Roxen whose configuration port you wish to change and change to the port number you found.

ADDING A NEW VIRTUAL SERVER.

Roxen makes it possible to easily have any number of virtual servers, bound to different ports and/or different network interfaces. See "Setting up Virtual Servers" on page 25.

FIGURE 3.8 *Hit this button to add a new virtual server.*

To add a new virtual server, go to the Servers page by clicking on the *SERVERS* panel, instead of clicking on the arrow to just unfold its configuration. Focusing is necessary to make certain actions available. This is true in several cases.

As you haven't added any configurations, the page you come to should be empty. Click on NEW VIRTUAL SERVER to add a new one.

You will be prompted to enter a name of the new virtual server. Enter something appropriate. You will also have to chose the configuration type, whereupon you should click ADD IT!, see figure 3.9. The configuration type tells Roxen which modules, if any, should be installed initially in the new virtual server. Note that if you have previously configured one or more virtual servers you will be able to copy one of those configurations by choosing its name from the the list of possible configurations, cf. the last choice in figure 3.10.

After clicking ADD IT! you will be able to see the Servers page again, which should have a new entry - the one you just created.

CONFIGURE THE VIRTUAL SERVER.

The first thing you should do is configure the *Server specific variables*.

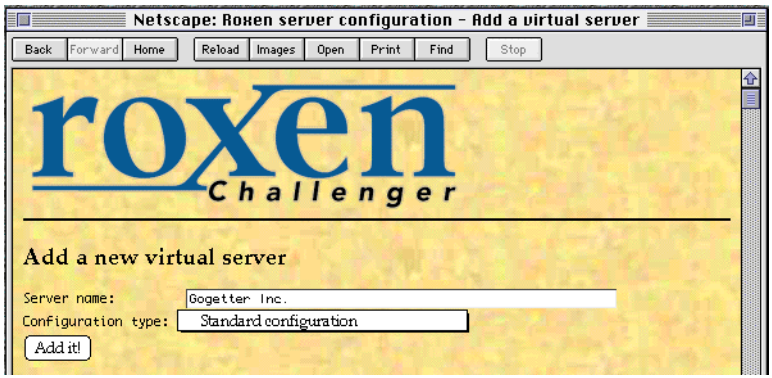

FIGURE 3.9 *Adding a virtual server.*

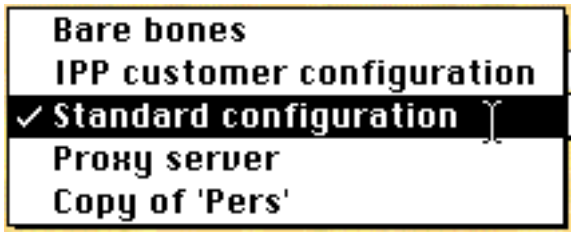

FIGURE 3.10 *Configuration types to choose among. The last choice makes a copy of a previously configured virtual server.*

When a new server has been added Roxen will have unfolded all the nodes where there are variables that should be set or at least checked.

Most of these variables should be correct by default. However, it is a good idea to check them all, especially *Server URL*, *Listen ports* and *Domain*.

We want our server to run on the default WWW port number, which is 80. As we noted earlier, Roxen has to run as root to be able to open this port. If you're using the uid / gid feature, restart Roxen now by hitting the RESTART button.

N.B.: Sometimes, it may be necessary to restart Roxen in order to properly make use of port 80. Also, when you've made changes and they don't seem to stick in spite of having clicked on the SAVE button it doesn't hurt hitting RESTART as a first remedial action.

When you do not wish the server to use port 80 or you're not allowed to use it on your system you have to tell Roxen what port to use. Port numbers range from 0 to 65535 but not all of them are available for use. Under the server specific variables you can set the ports on which Roxen should listen, see figure 3.11. If the virtual interface that you wish to bind

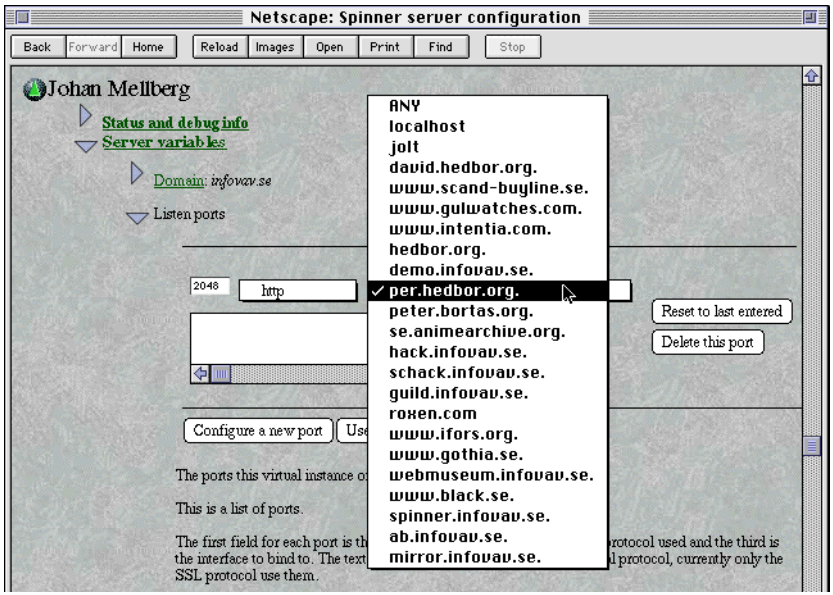

FIGURE 3.11 *Binding a virtual server to a virtual interface.*

to is not present in the list you can also manually anter the IP number of that interface in a separate field, which is hidden under the interface pop up menu in the figure.

ADDING MODULES

FIGURE 3.12 *The main buttons when working with virtual servers.*

To be able to add modules (filesystems for example), you need to focus on the virtual server. You do this by clicking on the name of the server.

As you can see, two new buttons appear at the bottom of the page; NEW MODULE and ZAP VIRTUAL SERVER. There is also the UNFOCUS button () next to the name of the virtual server. This button will be present every time you have focused on a menu item, or node.

If you click on ZAP VIRTUAL SERVER, the whole virtual server is deleted. If this sounds dangerous, relax; Roxen will ask you to confirm your choice before executing this potentially destructive command.

FIGURE 3.13 *Roxen lets you confirm your decision.*

To add a new module, click NEW MODULE. You will now get a list of all available modules, each with a short description attached, as in figure 3.14. Read the module documentation for more information

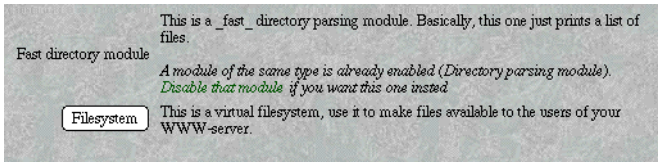

FIGURE 3.14 *Example of entries in the long list of available modules to add.*

about the modules shipped with Roxen, see “Available modules” on page 111.

By default there are several modules enabled, among them the *RXML Parser*, *Contenttypes*, and the *Filesystem* modules. N.B.: The Filesystem module is not present if you choose a bare bones configuration. In fact no module at all is there. Unless you have very specialised needs, choose the standard configuration.

Looking in figure 3.14 you see that it is not possible to add a *Fast Directory Module* due to the fact that there can only be one module of this type enabled at the same time in any one virtual server. See the chapter on

modules.

Now let's assume that you wish to add a new filesystem module or that you have installed a bare bones Roxen.

To add a module, click NEW MODULE and choose *Filesystem Module* by clicking on the button, see figure 3.14. When the module has been added you will be returned to the configuration interface. If there are any variables that perhaps should be changed, they will be unfolded and all you have to do is to change them and press SAVE. The variable to change in the filesystem module is the search path. Set it to what it should be, i.e. the path in the real filesystem where the server's HTML files reside, and press SAVE. If you try to enter an invalid path, Roxen notices this and complains. Enter a valid path, and try again.

FIGURE 3.15 *Roxen protests. Loudly.*

Press the SAVE button when you're done and the fileystem is thereby configured. You should now try using your newly configured server, by connecting to your server's URL. If everything went well, you now have a working WWW-server!

This is the end of the tutorial. You should now be able to add and configure more modules on your own. We would appreciate if you send us mail (roxen@infovav.se) with your thoughts and comments about this tutorial. Of course you can use the normal address too, see the first chapter.

CHAPTER 4

SETTING UP VIRTUAL SERVERS

Some operating systems have support for ”virtual net interfaces”, meaning that one computer can have many IP-addresses on the same physical ethernet interface. When using Roxen this means that you can easily have many virtual servers, with different IP numbers on the same computer.

A FEW EXAMPLES OF VIRTUAL INTERFACES

ON A LINUX MACHINE

If you have a recent Linux kernel it is a simple task to set up a virtual interface. Do this from the command line or in a startup script.

```
/sbin/ifconfig eth0:0 »IP-NUMBER« netmask »NET-
MASK« broadcast »BCAST«
/sbin/route add -host »IP-NUMBER« dev eth0:0
```

You might not need to provide the `netmask` and `broadcast` parameters. To add more virtual interfaces, simply use `eth0:1`, `eth0:2` etc

If you use Linux you can read more in /usr/src/linux/Documentation/networking/alias.txt in the kernel source tree.

ON A SOLARIS MACHINE

```
ifconfig leY:X »IP-ADDRESS« broadcast »BCAST« net-
mask »NETMASK« up
```

Y is the number of the ethernet interface, X is a number between 1 and 255 (the virtual interface number).

To automate the process instead of having to enter the line from the command line every time after booting you can do like this:

- Create a file named `hostname.»interface«:»copyNo«` in /etc/ containing nothing but the name of the computer.
- Put this name in /etc/hosts

  Example 4.1)

The file /etc/hostname.le0:1 contains the single line

```
roxen.com
```

The line

```
194.152.182.74 roxen.com
```

is then added to /etc/hosts.

Why do this then? Well, the next time you boot, the interface is added automatically. Quite nice, eh?

ON A FREEBSD MACHINE

```
ifconfig le0 »IP-ADDRESS« alias netmask 0xffffffff
```

can be entered from the command line or entered into a startup script.

SGI'S RUNNING IRIX 5.3

First, get patch 797 and install it. This allows the virtual host functionality.

Now there are two different methods. First, the standard command line method:

```
# ifconfig ec3 alias »IP_ADDRESS«
```

However, the best way to do it is using the IRIX configuration state checker which has an option for IPaddress aliases. Turn this option on with the following command:

```
# chkconfig ipaliases on
```

Then, edit /etc/config/ipaliases.options which is a self-documenting

file which contains the list of hosts or IP addresses to be used as IP address aliases. The format of this file is simply:

```
interface host1 [netmask addr] [broadcast addr]
interface host2 [netmask addr] [broadcast addr]
```

Where `host?` is either a valid name in `/etc/hosts`, or an IP address in dot notation. `netmask addr` and `broadcast addr` are optional. For example:

```
ec3 205.160.174.9
```

That's it. Reboot the machine, and the aliases will be correctly configured after the machine is up and running.

GENERAL STUFF

Don't forget to add the names in your DNS configuration!

Selected reading concerning this issue is "Two Servers, One interface", http://www.thesphere.com/~dlp/TwoServers/

If you have an ifconfig that supports the alias option you can use:

```
ifconfig le0 »IP-ADDRESS« alias
```

Read the manual pages for `ifconfig` on your system.

CHAPTER 5

MIGRATING TO ROXEN FROM OTHER SERVERS

Roxen can take care of many of the special features of other well-known servers. Basically it is quite painless to switch to Roxen. However, in version 1.0 there isn't any easy way of moving the configurations of another web server to Roxen. Below you'll find the common special features of other servers and how Roxen treats them.

IMAGEMAPS

NCSA IMAGE MAP TIP

By adding a redirect from /your_cgi-bin_dir/imagemap/ to /, Roxen will handle all files using the NSCA imagemap script internally. This means that you won't have to change any links to get your old imagemaps to work with Roxen.

SERVER SIDE INCLUDES

Roxen is compatible with NCSA / Apache style Server Side Includes, SSI. However, we suggest that you rewrite your files to take advantage of

Roxen's native features instead, features that can accomplish the same things, but faster, mainly the various RXML commands.

If you already use SSI you know how it works. If this is the first time you encounter it and wish to use it despite the existence of equivalent Roxen-native alternatives, take a look att http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html.

By default, support for the execute script command <!--#exec --> is turned off, while the others are on. This is controlled in the Main RXML parser, cf. page 128.

Server side includes are internally treated like RXML tags.

CGI

All your existing CGI scripts should work but be sure to read the section on the CGI script module, page 112.

ROXEN USER'S GUIDE

Detta dokument har framställts med FrameMaker 4.0.4

CHAPTER 6

HTML - A SHORT REFERENCE

What follows is a short reference for you, the user of Roxen, so that you can write your Roxen-enhanced HTML pages without having to have separate manuals for the RoXen Macro Language (RXML) and HyperText Markup Language (HTML). Note that this guide only provides you with some basic HTML. If you need to know more we encourage you to seek out a dedicated instruction, preferably on the Internet.

Examples on using several of the tags, both RXML and HTML, are at the end of the next chapter.

HTML INTRODUCTION

An HTML file consists only of text. This might seem surprising since you can display pictures, play sounds and even let users interact with the content; in short, present a rich, interactive, multimedia environment on a page. Now this is possible since you can embed ”commands” or tags in the text which will tell the browser to react in different interesting ways.

One basic example is the inclusion of images. When you order your browser to go find a WWW-page (by either clicking on a link or typing in a locationin the location field) it will go fetch that page and read through it. The tags will cause the browser to display text in various fashions, like bold or italic and when it comes upon an instruction to

display an image it looks at the URL (the location of the image) and then goes to fetch the image referred to and inserts it on the page where the instruction was found. The same procedure is used for every non-text file you wish to include on a page; they are loaded separately, after the HTML-file itself.

In short, HTML is a way of describing the contents of a page but not the exact layout of the page.

ESSENTIAL HTML

Now, for an HTML-file to be recognised as such there has to be some easily recognisable feature. This means certain tags:

```
<HTML>
<BODY> The text the document consists of.</BODY>
</HTML>
```

These are necessary but it is also a good idea to give the document a title by including the following between <HTML> and <BODY> in the above piece of code:

```
<HEAD>
<TITLE>Title of the document</TITLE>
</HEAD>
```

Since this title is what ends up in the lists of bookmarks (or on the hotlist) of the user's browser as well as in the window title it is a nice touch to provide a title that gives some kind of clue as to the contents of your page. People may thank you for doing this!

You can control the colours of your page through the *attributes* of the <BODY>-tag;

- bgcolor="#xxxxxx"; background colour.
- text="#xxxxxx"; colour of normal text.
- link="#xxxxxx"; colour of link text.
- vlink="#xxxxxx"; colour of followed link.
- alink="#xxxxxx"; colour of an active link.

Note that since the person reading your page can order his or her client to not care about your settings you do not have complete control over the result of your page. Since different browsers display HTML different-

ly it is not sure what your page will look like.

xxxxxx is any hexadecimal number (base 16). 000000 gives the colour black and ffffff gives you white, while ffff00 results in light gray. Instead of using one background colour, you can tell the client to use a certain image file as background:

- background="images/background.gif"; the browser will tile this image in the background.

Most tags have a starting tag and an ending tag (<BODY>...</BODY> for example). We refer to this type of tag as a *container*. In some cases the end is *implied* which means that it doesn't matter if you include the final tag or not. You will soon come across this type of tag.

CHARACTERS (STYLES)

<B>...</B>

The enclosed text will show up bold.

<I>...</I>

The enclosed text will show up in italic.

<STRONG>...</STRONG>

In most browsers the enclosed text becomes bold. <STRONG> is a *logical* tag where it is up to the browser to display the text in a way that gives an impression of, perhaps, something important.

<EM>...</EM>

This is also a logical style which most browsers display as italic. It is intended to convey emphasis.

<TT>...</TT>

Gives you typewriter text, i.e. text with fixed character width.

<FONT>...</FONT> AND <BASEFONT>...</BASEFONT>

These tags affect the way text is displayed. The most common use for these tags is to change size by using the attribute `SIZE=(+/-)N` where `N` lies between 1 and 7 where 1 is smallest and 7 largest. The bulk of the text is by default considered to be size 3 which then corresponds to the fontsize set in the browser's preferences. This default value can be changed with the <BASEFONT>-tag. The <FONT>-tag on the other hand only changes the size, not the *default* size.

If you use `N=+`*`value`* or `N=-`*`value`* you change size relative to the basefont size (normally 3). See an example of using the <FONT> tag in "Examples of HTML and RXML" on page 68.

By using the attribute `color="#xxxxxx"` you can have different text colours.

As you may or may not know, there has been little or no possibility to display different typefaces up until now. However, the Microsoft Internet Explorer makes use of the `face` attribute. As this is not yet even remotely standardised, we recommend that you avoid it for now.

PARAGRAPHS

<P>...</P>

A paragraph of normal text *begins* with <P>. The ending </P> isn't necessary (it is implied). Paragraphs are separated by one empty line.

Attributes:

- `align=left|center|right`; How the paragraph text should be aligned on the page.

<HX>...</HX>

This is the container for a heading. `X` is an integer between 1 and 6 where 1 is largest and 6 smallest (note the difference from the `size`-attribute of the font-tag). See "Examples of HTML and RXML" on page 68 for examples. The text of a heading is considered a closed paragraph.

Attributes:

- `align=left|center|right`; How the header should be aligned on the page.

LISTS

With HTML you can display three kinds of lists; *unordered* (with discs, circles or squares as bullets for the elements), *ordered* (with numbers, roman numbers or letters as counters for the elements) and *definition* lists. The definition list element consists of one term and then a definition for that term. Of course it doesn't have to be like this. If you intend to present a group of people and some facts about them this list type might be appropriate. Lists can be nested and the bullets change depending on the indentation level.

<UL>...</UL> AND <OL>...</OL>

Containers for unordered and ordered lists respectively.

Attributes:

- `type=disc|circle|square` (unordered list); you can order the browser to change the bullet independent of indentation level.
- `type=A|a|I|i|1` (ordered list); large / small letters, large / small roman numerals or ordinary numerals.

<LI>

The start of a new list item in both ordered and unordered lists. Since, as long you are in a list, a new **<LI>** means that the previous list item ends, the **</LI>** is unnecessary. The end is *implied*. There can be any number of other tags within a list item.

<DL>...</DL>

Container for a definition list.

<DT>

The term to be defined or descibed. The ending tag is unnecessary since it is considered to end when the definition tag (**<DD>**) comes along.

<DD>

The definition or description. It is usually displayed indented relative to the term itself.

TABLES

Tables are not handled by all browsers but many display them correctly.

<TABLE>...</TABLE>

Contains the table.

<CAPTION>...</CAPTION>

The title of the table. Must be placed within the <TABLE>-container but not within any row or cell.

Attributes

- `align=top|bottom`; places the title either above or below the table. Default is top.

<TR>...</TR>

The container for a row in the table.

<TD>...</TD>

The individual cell. Can only be placed within the row container.

<TH>...</TH>

Identical to the <TD>-tag but the text is automatically bold and centered. TH is short for Table Header.

Attributes common to <TR>, <TD> and <TH>:

- `align=left|center|right`; alignment of a cell's contents.
- `valign=top|middle|bottom`; vertical alignment of contents.

Default values are center and middle, respectively.

LINKS

Making hypertext links is the thing about the web that is so fascinating, i.e. being able to refer to any document on any computer connected to the Internet.

```
<A HREF="URL">LINKED TEXT</A>
```

When the user clicks on the words "Linked text" the browser sends a request to the relevant computer for the document referred to by the URL.

URL is an acronym standing for Uniform Resource Locator. This is what we today use to refer to resources (documents, programs, files, etc.) on the Internet.

If the document referred to resides in the same web server as the document containing the link, the URL need not include the web server's name. To refer to a document anywhere, the full form is `http://some.server.somewhere/my_documents/info.html`. This is an absolute URL. The part `/my_documents/info.html` is the absolute path to the document in the part of the real file system that is visible to the web server. By visible, we imply that outsiders do not have access to the complete filesystem of the machine that runs the web server. The real absolute path may be `/usr/www/vinnies_server/my_documents/info.html`. By only providing a file name, the file is supposed to be in the same server and directory as the referring document. You can also use the standard way of referring to directories relative to the present directory by using `../` to refer to the directory above the current and so on.

```
<A NAME="ANCHORNAME">THE ANCHORED TEXT</A>
<A HREF="URL#ANCHORNAME">LINKED TEXT</A>
```

Sometimes you wish to refer to a specific place in a document. This is done by first defining an anchor (the end of the link, or landing spot) as in the first header line above. Then you can refer to this exact location within an HTML document by just adding `#anchorname` as in line two, to the URL of the document containing the anchor. If you refer to an anchor in the same document you don't need to provide a URL, i.e. it is sufficient with `HREF="#ANCHORNAME"`. Don't forget the hash sign, though.

FORMS

A form is a way of letting the viewer provide data to your server, a way of implementing a certain interactivity on your pages.

<FORM>...</FORM>

This is the container for a form. Between <FORM>...</FORM> there may be many possible fields to fill in.

Attributes

- ACTION="URL"; The URL is the location of the program that should take care of the form data, for example a Pike script.
- METHOD=GET|POST; GET appends the form data to the URL (i.e. the action) like this:

  ```
  action?name=value&name=value&name=value
  ```

  As you can see, the URL is separated from form data by a question mark and the name=value pairs are separated by ampersands. Every pair corresponds to a field and the value entered by the user.

  POST sends the data in a separate block of data and is thus better when there is a lot of data to send.
- ENCTYPE="application/x-www-form-urlencoded"|"multipart/form-data"; The first is when you have an ordinary form and the second is when one of the fields corresponds to a file, i.e. the user can upload files using his or her favourite browser.

The only attribute that may be omitted from the <FORM> tag is ENCTYPE.

<INPUT>

This tag just displays a simple field.

Attributes

- TYPE="type"; the type of the field. Type must be one of these:
  - text: a text entry field, the default.
  - password; a text entry field, but the characters entered are displayed as asterisks.

  - checkbox; displays a single ”button”, where the value is either true or false.
  - radio; This is also a button as the checkbox, but if you have several fields of the same name they are grouped together, i.e. only one can be chosen at the same time.
  - submit; displays a pushbutton that causes the form data to be sent according to the action and method defined in the <FORM>.
  - reset; displays a pushbutton that returns all the form fields to their initial values.
- NAME=name; the name used for the field when sending the data, not displayed by the browser on the page. Use normal HTML to put text next to fields on the page. This attribute is required in fields of all types except submit and reset.
- VALUE=value; can be used to put a default value in a field. Text entry fields display the value, checkboxes and radiobuttons are only relevant when they are ”on” as the form is submitted. If value is specified for a pushbutton it specifies the label on the button.
- CHECKED; Turns checkboxes and radiobuttons *on*.
- SIZE=size; The displayed size of the field in characters; thus applicable only on textentry fields.
- MAXLENGTH=maxlength; The maximum number of characters allowed to be entered in a textentry field.

<SELECT>

This is a container that allows you to provide multiple choice fields. No HTML except <OPTION> is allowed within this container. The display is either option menus or scrolled lists.

Attributes

- NAME=name; the symbolic name of the field, cf. the name attribute of the <INPUT> tag.
- SIZE=size; Determines how many selectable items are displayed. If SIZE=1 or SIZE is not present the display is an option menu, otherwise a scrollable list.
- MULTIPLE; allows the selection of multiple options. If present it forces the display to be a scrolled list, regardless of the value of SIZE.

<OPTION>

The tag that specifies an option within a <SELECT> container. It works like a list item in a list.

Attributes

- SELECTED; This option is the one selected by default. Unless the MULTIPLE attribute of the <SELECT> is set, only one option can be selected.

<TEXTAREA>...</TEXTAREA>

This container allows the user to enter multiple lines of text. The font is of fixed width.

Attributes

- NAME=name; symbolic name of field.
- ROWS=rows; Number of rows displayed on screen.
- COLS=cols; Number of columns displayed.

If you enter any text between the starting and ending tags it will be displayed in the textarea field by default.

MISCELLANEOUS

<IMG>

Insert an image.

Attributes

- SRC="URL"; specifies where the image file resides.
- ALT="Text"; If there is a failure in the transfer of the image, or if the person browsing the page on which the picture should be displayed is using a text-only browser (and the image is a link), this text will be shown instead. It is a nice thing to do to provide this text. If the image is nothing more than decoration, text-only browsers will treat it as non-existant.
- ISMAP; Instructs the browser to treat this as an image with hot spots defined. See “Image Maps” on page 77.

Note that there are many more attributes but these are the most interesting ones. Netscape Navigator and the Microsoft Internet Explorer also support an attribute called USEMAP which allows you to use client-side image maps. More on this in "Image Maps" on page 77.

<HR>

Insert a horizontal line (ruler). You can set its thickness by adding the attribute SIZE=x, where x is a number of pixels. You can also direct the browser to display a ruler of smaller length by adding WIDTH=x, where x might be a fixed pixel value or a percentage of the window size. If you then have a ruler smaller than the actual window you can align it by setting ALIGN=left|center|right.

<PRE>...</PRE>

In case you wish to retain spaces and newlines you use this tag which will make the textblock look exactly like you wrote it instead of the browser's compacting any kind of whitespace (tab, newline and space) into one single space. Text is displayed using a fixed-width font.

This tag gives you a linebreak, and only a linebreak, no empty line as with paragraphs. Inserting linebreaks, no matter how many in a row, does not give you a new paragraph.

<CENTER>...</CENTER>

Instead of using the align-attribute of the <P> and <Hn> tags you can center text using this tag. The advantage of this tag is that it is more flexible than the the align attribute and you can center anything. It's originally a Netscape-specific tag.

SPECIAL CHARACTERS

Since the browser interprets lesser than (<) and greater than (>) as a kind of signal, these characters have to be coded to display correctly;

- lesser than: `<`
- greater than: `>`

Do not forget the trailing semicolons! As you understand, this means that the ampersand is also a special character and the code for the ampersand is `&`. There are many more of these but they aren't all that important.

In HTML tutorials originating from mainly english speaking countries, you will find that they sincerely believe that no characters other than a-z and A-Z can be displayed by browsers without resorting to abominations like `&auml;` for ä, `&Ouml;` for Ö and so on. This is not true! It may have been once but today there are perfectly valid eight bit character sets used on computers everywhere. Use the letter itself, not some horrible code. Using these codes make your source harder to read, especially if you are using a lot of national characters.

Note that the Macintosh does not use the Latin1 character set, wherefore the eight-bit characters may show up strange in the browser when looking at local files. Don't worry, when you place your page on a server, it will display correctly. If it doesn't, your system administrator has some work to do.

If you want proof that this indeed works, look at our pages at `http://www.infovav.se/`. Nowhere do we use these character codes and we guarantee you that we do not use any "server side tricks".

Also, beware of HTML editors. Some of them translate those eight bit characters into their corresponding HTML code.

CHAPTER 7

RXML - THE ROXEN MACRO LANGUAGE

Right! Now that you know how to make ”standard” web pages, let’s get on with using Roxen to easily customise your web site to provide a dynamic content.

INTRODUCTION TO RXML

Roxen tags are used in the same way as normal HTML tags. Before files are sent, they are parsed by your Roxen’s *Main RXML parser*, cf. “Available modules” on page 111. The RXML tags are then replaced by something suitable whereupon the file is sent out as a normal HTML-file. This of course means that Roxen is not the fastest webserver around, but it is infinitely customisable thanks to the module concept. It also kicks some serious butt with many smaller web servers.

Note that this parsing is not the same thing as server side includes (SSI), although Roxen supports that too in order to facilitate an upgrade from another server without having to immediately upgrade your pages.

Detta dokument har framställts med FrameMaker 4.0.4

RXML TAGS

These are the RXML tags included with Roxen. They are quite sufficient for most purposes but if you have specialised needs you can also write your own. Refer to the chapter on programming your own modules for an introduction to this topic.

Refer to the last section of this chapter, “Examples of HTML and RXML” on page 68, for a somewhat larger exampe of how to use several of the tags.

For easy reference you will find all the available tags in table C.1 on page 199 with references to the page where the description may be found.

<ACCESSED>

This is a basic access counter. Instead of using a CGI-script, you just insert <accessed>. To use this tag you have to activate it in the RXML parser module.

Attributes

- add; Add one to the number of accesses of the file that is accessed, or, in the case of no file, the current document.
- addreal; If you use cheat or other modifiers to the count, this attribute adds the correct count as a comment in the HTML code, like this: <!-- (4711) -->.
- capitalize; Capitalize the string.
- cheat=num; Add num to the actual number of accesses.
- factor=mult_factor; Multiply the actual number of accesses by mult_factor.
- file=filename; Show the number of times the file filename has been accessed instead of how many times the current page has been accessed. If filename does not begin with ”/”, it is assumed to be a URL relative to the directory containing the file in which the <accessed> tag was found. Note that you will have to type in the

full name of the file. If there is a file named `tmp/index.html`, you cannot shorten the name to `tmp/`, even if you've set Roxen up to use `index.html` as a default page.

One limitation is that you cannot reference a file that does not have it's own `<accessed>` tag. It suffices to enter this in a comment in the file, i.e. you do not need to display it in everyone of these files:

```
<!-- <accessed> -->
```

This is done for reasons of efficiency, since it would be very costly to automatically log accesses for all files..

- `lang=language`; When using `type=string`, the number string is replaced by the string in the language indicated by this attribute. This also affects the `since` and `part` attributes. Available languages are Swedish (se), Finnish (fi), German (de), Catala (ca or es_CA), Dutch (du), Spanish (es), French (fr), English (en, default) and Norwegian (no).
- `lower`; Lowercase the string.
- `per=second|minute|hour|day|week|month`; how many accesses per unit of time.
- `prec=number`; Round the number of accesses. The result is an integer with `number` valid digits. If `prec=2`, show `12000` instead of `12149`.
- `precision=number`; exactly the same as above.
- `reset`; Resets the access counter. Hint: Only do this under special conditions, i.e. within an `<if>...</if>`. Otherwise, why use `<accessed>` at all, if you reset it every time someone accesses the page?
- `silent`; do not show the count.
- `since`; Insert the date that the accessed number is counted from. The language will depend on the 'lang' tag, default is english.
- `type=string|roman|mcdonalds|linus|number`; This attribute allows you to specify how the accesscounting should print; as a number (default), as a roman numeral or something else[5]. Try them!
- `upper`; Uppercase the string.

See also: "<DATE>" on page 51 and "<MODIFIED>" on page 62.

<ACONF>

Works like <apre> below but is used for turning on and off configurations for individuals using *cookies*[6]. This is done by preceding the name of the configuration type with a plus sign (+) to turn it on or a minus sign (-) to turn it off. Without the href attribute it is just a link to the same page. Note that the user won't see the change until he has requested the page again due to the way cookies work.

Example 7.2)

```
<html>
<if config=bg>
<body background=foo.jpg>
<otherwise>
<body>
</if>

Welcome to my page on the web.
<hr>
```

5. The type linus demands some kind of explanation; Linus is the name of a member of the Lysator Academic Computer Society of the university of Linköping. In Roxen's infancy (as Spider2) there were problems with the access database crashing all time. This meant that the access counter was reset to zero - not very good. Someone decided to temporarily hide this problem by providing a random number between 1000 and 2000 instead of reading the access log. Now, Linus changed the behaviour again by rewriting the module to not only show the low access count of the access log, but also since when this count was made, i.e. since the database last crashed. The problem with his solution was the abominably inflexible way of displaying the information. His punishment for this is to forever be included in the <accessed> tag.
6. A cookie is a way of keeping information about someone by saving information at the client side of the http exchange. As a simple example you could have a cookie named counter and give one to everyone accessing your page for the first time and then increasing its value by one everytime they request the page, thereby keeping an individual access count. Supported by the Netscape Navigator and the Microsoft Internet Explorer.

```
<if config=bg>
<aconf -bg>[Turn off background]</aconf>
<otherwise>
<aconf +bg>[Turn on background]</aconf>
</if>


...Some HTML...
</body></html>
```

In this example we first check if ”background mode” is on. If this is the case we send the <body> tag with the background attribute, otherwise without. Below that we define two links of which only one is sent. If ”background mode” is on we give the user the possibility of turning it off and vice versa. The configurations are present in the URL in the same way as prestates are except that they are between <...>, cf. footnote 7. on page 49 and <apre> below.

<APRE>

<apre href=URL>...</apre> is used instead of ... to make it possible to add *prestate-relative* links[7]. If used without href, it's just a link to the same page.

Example 7.3)

Suppose we have requested the following URL: http://www.foo.com/(sv,img)/index.html. If the source looks like this:

```
<if prestate=sv>
Du har valt svensk text
</if>
<else>
You don't want swedish text
</else>
<apre -sv>I want english text!</apre>
<apre sv -img href=images.html>
Show me the image-index, w/o indexpictures and in
```

7. A prestate is part of the URL and is a way of setting variables so that you can for example use <if> to change the layout of a page without having to make multiple files. The prestate is part of the URL, but it isn't a separate directory.

```
swedish
</apre>
```

the text ”Du har valt svensk text” will be sent to us and also two links; the ”not swedish” link and the ”image index in swedish but no images” link. Clicking on the first will request the URL http:/ /www.foo.com/(img)/index.html (i.e. the same page). The page that is sent contains the text ”You don’t want swedish text” and the same links. Clicking on the second will request the URL http:/ /www.foo.com/(sv)/images.html.

As you might understand, the parenthesised part is the prestate. It is part of the URL, but Roxen removes the prestate(s) when choosing the page to send, remembering them while parsing the page so that you can check them using <if> in order to customise the page.

N.B.: Prestate links only work with URL:s within the server, absolute URL:s do not work.

Example 7.4)

```
<apre -sv href=”http://your.own.domain/”>HO-
ME</apre>
```

does not work, but

```
<apre -sv href=”/”>HOME</apre>
```

does. Of course you may use the ordinary anchor tag doing prestate insertion by hand;

```
<a href=”http://your.own.domain/(sv)”>HOME</a>
```

to achieve the same thing.

See also “<IF>, <ELSE> and <OTHERWISE>” on page 56.

<BLINK>

If the Lysator module is enabled and the variable Blink enabled is set to No it replaces every occurence of the <BLINK> tag with <B>. Why? Well, because <BLINK> is the most annoying tag ever invented. So there!

<BOFH>

This tag is quite reminiscient of the <dthought> tag and is replaced by a random Bastard operator excuse. It is only available if you have installed the BOFH module.

BOFH is an acronym for Bastard Operator From Hell, and if you’ve read

the BOFH series on the Internet you know what to expect.

<CLIENTNAME>

The name of the client used, in case the user forgets.

Attributes

- `full`: Insert the full name of the client instead of just the first part of it. For Netscape Navigator, as an example, the full name is 'Mozilla <version> <platform> <OS full version>', and the short name is 'Mozilla <version>'.

<COMMENT>

All text written in the comment container are comments. This means that they will be removed before the file is sent to the client. Note that this is not the same kind of comment as the HTML comment using `<-- blah blah -->` syntax. Note the difference between this one and the `<comment>` container of the lysator module, see page 51.

<COMMENT>

Used inside the `<lysator>` container it is different from the normal RXML `<comment>` tag. Outputs the string `"<!-- Text between starting and ending comment tags -->"`, which is the normal HTML comment.

<DATE>

Insert the (more or less) current date.

Attributes

- `day=(-)X`; Add (remove) X days to (from) the date.
- `hour=(-)X`; Dito, but add (remove) X hours instead.
- `minute=(-)X`; Dito, but add (remove) X minutes instead.
- `second=(-)X`; Dito, but add (remove) X seconds instead.

See also "<ACCESSED>" on page 46 and "<MODIFIED>" on page 62

<DEFINE> AND <INSERT>

Roxen has support for making macros. This is useful for making sitewide

definitions of titles or menu items, thus making it easier to create uniform-looking pages, not to mention changing the whole layout in one fell swoop. It is possible to make a define in a define, and using `<insert name=foo>` inside a definition also works. Try what you like, it will probably work as you expect.

<DEFINE>

Define a macro to be used by `<insert>` later on.

Attributes

- `name=macro`; Define this macro.

  Example 7.5)

```
<define name=1>This is macro number one</define>
```

<INSERT>

Used to insert macros, variables, cookies (cf. footnote 6. on page 48) and files. The format is as follows:

```
<insert name=name|variable=name|cookie=name|vari-
ables[=full]|cookies[=full]|file=path from-
word=toword>
```

Note the "|" characters. You can only use one of the attributes in every `<insert>`. Replacing words is of course possible all the time. Only lowercase character sequences can be replaced.

Attributes

- `cookie=name`; Insert the value of the cookie by name `name`.
- `cookies[=full]`; Insert the values of all cookies, more or less verbose.
- `file=path`; Insert the file `path`. This file will then be fetched just as if someone had tried to fetch it with an HTTP request. This makes it possible to include things like Pike script results and such.

  If `path` does not begin with "/", it is assumed to be a URL relative to the directory containing the file that has the `<insert>` tag in it, i.e. the file where the inserted text is finally parsed.

  N.B.: Included files will be parsed if they are named with the extension `.html` (or whatever extension the RXML parser should parse according to configurations in the main RXML parser module). It's

not a good idea to name include files like this, because it might render defines unusable. On the other hand, it could probably be useful in some cases, depending on your intentions.

Example 7.6)

If we put `<insert file="gazonk/foo">` in the file `bar.html` and `gazonk/foo` contains `<insert file=fubar>`, Roxen will look for `fubar` in the directory where `bar.html` is. If we change the first `<insert>` to `<insert file="gazonk/foo.html">` and rename `gazonk/foo` to `gazonk/foo.html`, Roxen will instead look for `fubar` the directory `gazonk/`.

- `fromword=toword`; Replace `fromword` with `to`word in the macro or file.
- `name=macroname`; Insert this macro, which should have been defined by `<define>` before it is used. If it resides in another file, you have to `<insert file=filename>` before you can insert the macro.
- `nocache;` File includes are normally cached. If the nocache flag is specified, that cache won't be used. Useful when including dynamic documents, like Pike-scripts.
- `variable=name`; Insert the value of the variable of name `name`.
- `variables[=full]`; Insert all variables, more or less verbose.

Example 7.7)

First we define a macro called `foo`:

```
<define name=foo>This is a foo</define>
```

Then we insert this macro somewhere:

```
<insert name=foo>
```

The text sent to the client is:

```
This is a foo
```

Example 7.8)

If we insert the above macro but tell Roxen to replace the word `foo` with the word `cat`:

```
<insert name=foo foo=cat>
```

we will get:

```
This is a cat
```

Example 7.9)

```
<insert name=foo a=the foo=”green door.”>
```

Result:

```
This is the green door.
```

Example 7.10)

```
<insert name=foo a=some foo=cats is=are>
```

Result:

```
Thare are some cats
```

Note that even parts of words become exchanged; ”This” becomes ”Thare”.

Example 7.11)

```
<insert file=/includes>
```

The result of this is that the contents of the file /includes is inserted here. Note that macros defined in this file are not inserted until they are called by `<insert name=macro>`. This is very useful for making site-wide defines (like heads, titles etc.), used in all files in a server, thus simplifying the code generation a lot.

<DOC>

This is useful for writing HTML-examples. It replaces &, < and > with `&amp;`, `&lt;` and `&gt;`

Attributes

- pre; Enclose the section with `<pre>...</pre>` as well.

<DTHOUGHT>

This tag is only available if you have installed the deepthought module. The tag is replaced by a random ”deep thought”.

<ENDTABLE>

Only allowed inside the `<lysator>` container. It ends the table created by all the `<item>`:s.

<FOT>

Only allowed inside the <lysator> container. This creates a footer. The only attribute is sv which gives the footer text in swedish. If left out, the text is in english. The RXML that is sent is hard-coded in the module;

```
<hr noshade>Feedback and Comments to: <user na-
me="some_user's_id">
<!--This page has been accessed <accessed> times
since <accessed since>, and it was last modified
<modified> by <modified by nolink realname>.-->
```

If you want to change it, read the source and change it.

<H>

Only allowed inside the <lysator> container. Creates a subheader using <FONT>. Quite dirty and thus works only for clients who can handle <FONT>.

Attributes

- level=number; How big should the heading be? An integer from 1 to 7.
- hr; use a horisontal ruler in the construction of the header.

<HEADER>

Add a header to the head of the response. When a browser sends a request for a file, the server returns a few lines of text that tells something about the result of the request and what kind of file the browser requested. By using this tag you can extend and / or modify this header, telling the browser, for example that the file demands authorization in order to be viewed.

Attributes

- Add »header name«:»value« to the response. See table C.2 for suggestions of headers to add.

See also "<RETURN>" on page 64.

If you wish to see what the server returns try the following example.

Example 7.12)

```
telnet myserver.com 80
```

and you will see:

```
Trying 123.4.56.789... Connected to myserver.com.
Escape character is '^]'.
```

Next, make an http request. For example, enter:

```
HEAD /Mycompany/logo.gif HTTP/1.0
```

and see what you get back; these lines are the headers.

<ICON>

This tag is only available when the Lysator module is enabled. Works like <picture> below, except that the images are assumed to be in the directory given by the variable Icon pre-URL.

<ICONS>

This tag is only available when the Lysator module is enabled. The enclosed area will only be sent if the browser supports inline images.

<IF>, <ELSE> AND <OTHERWISE>

Use <if> to only show the enclosed section when certain conditions are met. You can also use <else> or <otherwise> in order to suggest alternative actions if the conditions do not evaluate to a true value.

These are perhaps the most useful tags in RXML. Among other things, it allows you to write HTML-code that is only showed to people with a certain client. You can for example make a table, that if viewed on a non-table compliant client uses preformatted text, or maybe even a completely different text instead.

Note that the part(s) that should not be seen according to the conditions, are not even sent which means that the person looking at your page won't even know that he / she isn't seeing everything. This also makes it possible to entertain the illusion of dynamic pages, without using scripts, through clever use of RXML tags.

Below are the possible attributes to this tag. There are a lot of them and they have been divided into *conditionals* (the de facto checks) and *modifiers* (slight change in behaviour of the checks).

N.B.: If you don't use <if> before <else>, the result is unpredictable.

Example 7.13)

```
<if somecondition>
somecondition occurred
</if>
<else>
something else happened
</else>
```

is equivalent to

```
<if somecondition>
somecondition occurred
<otherwise>
something else happened
</if>
```

CONDITIONALS

- `accept=type1[,type2,...]`; The `type` refers to content-type, e.g. `image/jpeg` or `text/html`. The values can contain * (for several arbitrary characters) and ? (for any one character). Every client tells the server what it thinks it can deal with so this is one way of ensuring that nothing is sent that the browser can't handle.
- `cookie="name[ is value]"`; Returns true if the cookie named `name` exists. If you also include the part about value the expression is of course only true if the cookie holds that value. The value can contain * (for several arbitrary characters) and ? (for any one character).
- `date=yymmdd`; Show the enclosed section if the date is yymmdd.

  Modifiers: `before`, `after` and `inclusive`.
- `defined=definedmacro`; Show the enclosed section if the macro `definedmacro` is defined. Wildcards work as for `cookie`.
- `domain=pattern[,pattern...]`; Show the enclosed section only to hosts whose DNS name match these pattern(s). Note that domain names are resolved asynchronously. This means that the first time someone accesses this page, hostname will be the same as the IP number.
- `host=pattern[,pattern...]`; Show the enclosed section only to hosts whose IP number matches one of these pattern(s).

- `language=language1[,lang2, ...]`; True if the client prefers the language(s) listed. * and ? may be used and work as for `cookie`. Available languages are Swedish (se), Finnish (fi), German (de), Catala (ca or es_CA), Dutch (du), Spanish (es), French (fr), English (en, default) and Norwegian (no).

  The language codes are the international two-letter codes used for example in domain names. Roxen determines this by looking at the request-header `Accept-Languages`.
- `name=pattern[,pattern,...]`; If the full name of the client matches the given pattern, show the enclosed text.
- `prestate=state1[,state2, ...]`; Show the enclosed text, only if all the specified prestates are present. The prestates are prepended to the URL with this syntax: `http://www.whatever.domain/(pre,state)/my/nice/page.html`. See "<APRE>" on page 49 for more information.
- `referer`; Show the enclosed text, only if the *referer header* is supplied by the client.

  If you add a pattern (`referer=pattern[,pattern,...]`) then send the enclosed text only if the referer header matches the pattern(s). See "<REFERER>" on page 64 for more information.
- `supports=feature`; If the client supports the given feature, include the enclosed section. This is configurable (Global Variables / Client supports regexps). These are the available features:
  - `backgrounds`; The client supports backgrounds according to the HTML3 specifications, cf. page 34.
  - `bigsmall`; client understands the `<big>` and `<small>` tags.
  - `center`; The `<center>` tag for centering HTML objects is supported.
  - `cookies`; client can receive cookies.
  - `divisions`; the client can at least handle `<div align=...>`
  - `font`; The client supports at least `<font size=foo>`, à la Netscape Navigator.
  - `fontcolor`; You can change the colour of individual characters.
  - `fonttype`; The browser can set the font, cf. Microsoft Internet Explorer.

- `forms`; Forms according to the HTML 2.0 and 3.0 specification are supported.
- `frames`; frames should work.
- `gifinline`; The client can show GIF images inline.
- `imagealign`; The client supports `align=left` and `align=right` in images, à la Netscape.
- `images`; the client can display images.
- `java`; the client supports Java applets.
- `javascript`; client supports java scripts.
- `jpeginline`; The client can show JPEG images inline.
- `mailto`; The mailto function can be used.
- `math`; the `<math>` tag is correctly displayed by the browser.
- `perl`; supports Perl applets
- `pjpeginline`; can handle progressive JPEG images (`.pjpeg`) inline.
- `pnginline`; client can handle `.png` images inline.
- `pull`; the client handles Client Pull
- `push`; the clint can handle Server Push
- `python`; supports Python applets
- `robot`; the request really comes from a search robot, not an actual browser.
- `stylesheets`; client handles stylesheets (a là Arena)
- `supsub`; handles `<sup>` and `<sub>` (superscript and subscript, respectively).
- `tables`; tables according to the HTML 3.0 specification are supported.
- `tcl`; supports TCL applets.
- `vrml`; the client supports VRML.

This list is refreshed automatically every week directly from our site (infovav.se) unless you explicitly tell Roxen not to do this by setting `Update the supports database automatically` under GLOBAL VARIABLES to `No`. The list is a list of browsers and what features they can handle, not something the browser sends.

- `user=name[,name,...]|any`; Show the enclosed section only to the user `name`, or, if `any` is specified, to any valid user on the system. Unless the modifier `file=X` is specified, the default user data base is used.

  Modifiers: `file=X`, `wwwfile`.
- `variable`; works exactly like `cookie`.
- `config=configuration`; If you have set a user's configuration through the use of one or several `<aconf +/-configuration>` you can use this conditional to check it. It works like prestates but is instead saved on the client side through the use of cookies.

MODIFIERS

Used in conjunction with some of the conditionals.

- `file=X`; Modifies the `user=userid` value. If this is specified, the `user:password` pairs will be taken from an external file of this format:

  ```
  username:encrypted-password
  username:encrypted-password
  ....
  ```

  Unless `wwwfile` is present, it is assumed that the file "X" is an ABSOLUTE pathname in the real filesystem, like `/usr/www/security/localpasswd`.
- `wwwfile`; Indicates that the file "X" is a file in the virtual filesystem of the server. This might be a security problem, since everyone can read it via WWW.
- `before`; Used together with `<if date=yymmdd>`. Show the enclosed section if the current date is before yymmdd.
- `after`; Used together with `<if date=yymmdd>`. Show the enclosed section if the current date is after yymmdd.
- `inclusive`; Used together with `<if date=yymmdd>` and before or after. Show the enclosed section if the current date is the same as or before/after yymmdd.
- `not`; Invert the results of *all* tests.
- `and`; Show the enclosed text only if all tests are true (default).
- `or`; Show the enclosed text if one or more of the tests are true.

Example 7.14)

This example shows how to make part of a page (in this case a list item) available to locals only, using the `host` attribute:

```
<if host="130.236.25?.*">
<li><a href="/local/">Local info</a>, only availa-
ble to local clients
</if>
```

A useful use of the `user` option might be:

Example 7.15)

```
<if not user=any>
<header name=WWW-Authenticate value="Basic; Re-
alm=Pers">
<return code=401>
<h1>Access denied</h1> You may not see this dou-
ment without a valid user and password.
</if>
<else>
.. The secret document ..
</else>
```

This will force a user entry. But please note that if the user presses the `Cancel` button, or refuses to enter an authentification, the parts of the document that is outside `<else>...</else>` will be shown to him or her.

<ITEM>

Only allowed inside the `<lysator>` container. This is a container that defines a link item with a short description and possibly an icon. It creates a part of a table.

Attributes

- `linkto`; The URL of the link.
- `icon`; the name of the image to use. It is assumed to be in the icon directory as stated in the variable `Icon pre-URL`. Leave out the file extension since it is assumed to be a gif image. Make sure that is the case.
- `title`; The linked text.

<LANGUAGE>

This tag was implemented for debugging purposes. It sends a list of all languages supported by the client. Helps you find out if you're doing something wrong or if the client doesn't do it's thing properly. The only attribute is `full`, which when included gives you a somewhat more verbose list.

<LYSATOR>

This container is only available when the Lysator module is enabled. This tag formats the display properly. Within the container, several additional tags are available. It uses tables for its output if the browser can handle it.

Attributes

- `pretxt="First text"`; text to be placed in front of the main title.
- `title="Nifty title"`; the title.
- `txt="Nice text"`; subheader.

<MODIFIED>

Insert the date when the page was last modified or by whom it was modified.

Attributes

- `by`; This tag will insert `<user name=user>`, where `user` is the last one to modify the file.
- `realfile=file`; Insert the modification date of the file `file` in the real filesystem. This tag can also be used together with `by`.
- `file=virtual`; Insert the modification date of the file `virtual` in the virtual filesystem. This tag can also be used together with `by`.

See also: "<DATE>" on page 51 and "<ACCESSED>" on page 46.

<PICTURE>

This tag is only available when the Lysator module is enabled. It works like the `<IMG>` tag except for the fact that you do not provide everything. The picture that you insert is supposed to be in `/pictures/` and considered a gif picture. Do not provide the `.gif` extension in the `src` attribute

because <picture> will add it. Otherwise it behaves like <IMG>.

<QUOTE>

Sometimes you wish to use the quotation marks for something apart from quotes. In that case you should, in order to avoid confusing yourself or the server, define other characters as starting and ending quotation marks, respectively.

Attributes

- start=start_char; The character beginning a quotation.
- end=end_char; The character that ends a quotation.

Example 7.16)

```
<quote start='{' end='}'>
<insert name=foo bar={"foo" bar gazonk 'elefant'
snabel}>
```

Since we have redefined quotation marks to being curly brackets, bar gets the value equalling the sequence of characters between the curly brackets. If we had not made this redefinition bar would have been just "{" or something unpredictable due to the placement of the ordinary quotation marks.

<RANDOM>

Randomly select a part of the document.

The text between <random> and </random> will be split on the specified separator, and one of the resulting parts will be returned, which one it will be is randomly selected.

Attributes

- separator=string; The separator to be used. If none is specified, newline will be used.

Example 7.17)

```
<random separator=+>
This is a test+This is not a test+What is this?
</random>
```

will cause Roxen to send one, and only one, of the three sentences separated by the plus signs.

<REFERER>

The Referer field allows the client to specify, for the server's benefit, the address, URL, of the document (or element within the document) from which the Request-URL was obtained. Insert the referer! It's good practice because it can help you track down faulty links.

FIGURE 7.16 *A nice Back button*

This tag can also be used for making a "back button", a link back to the page the user previously visited. We recommend that you use it together with `<if referer>`.

<REMOVE_COOKIE>

Removes a cookie. The attribute `name=cookiename` must of course be present.

See also "<SET_COOKIE>" on page 65.

<RETURN>

Return an HTTP result code other than 200, which is the regular, "no problem", return code.

Attributes

- `<return code=c>`; This will return the response `c`. Note that most of them are quite odd to have in a document, especially the server errors. See the listing in table C.3 on page 204.

See also "<HEADER>" on page 55.

<RIGHT>

This tag is essentially obsolete and is only here for sentimental reasons. Do not use it, it does not give any good results.

This tag tries to align the enclosed text to the right. Today it just makes a table, with `align=right`, which means that you can do it yourself by defining a macro.

Example 7.18)

```
<right>This text is aligned to the right</right>
```

will send the following to the client:

```
<tt>
<table width=100%>
<tr>
<td align=right>This text is aligned to the
right</td>
</tr></table></tt>
```

Sometime in the future, this tag may be developed further.

<SET_COOKIE>

Sets the `cookie` called name to `value`.

Attributes

- `name=name`; Gives the cookie a name.
- `value=value`; Assigns a value to the cookie.
- `persistent`; Tells the client to save the cookie forever.

See also "<REMOVE_COOKIE>" on page 64.

<SIGNATURE>

Like `<user>`, but with some more bells and wistles. The result will in fact be `<p align=right><address><user name=username></address></p>`. Like the `<right>` tag it is not recommended for use since we do not actively intend to develop this tag.

Attributes are the same as those of "<USER>" on page 67.

<SMALLCAPS>

This tag takes the enclosed string and turns it into a string of smallcaps (as you might have already guessed).

Attributes

- `size=x`; sets the base font size to x. x can be between 1 and 7. This is what is used for the capitals.
- `small=x`; sets the font size for the small capitals.
- `space`; inserts a space between every character in the string.

<SOURCE>

Show both the source and the parsed result of the enclosed section.

Attributes

- `separator="Separator string"`; Use this separator instead of the default "Result".

<TABLIFY>

You can let Roxen magically conjure up tables for you by to installing the Tablify module. This module lets you do the following:

```
<tablify>
tab separated text
</tablify>
```

and then Roxen sends an HTML 2.0 table to the browser instead of the tab separated text. Rows are separated by newlines and cells are separated by tab stops.

<TABLIST>

If you have installed the Tablist module you can let Roxen automatically generate tab lists like the one you find at the top of the configuration interface (Servers / Global Variables / Status / Debug). It also simulates tab lists in text-only browsers by using slashes and backslashes in combinations.

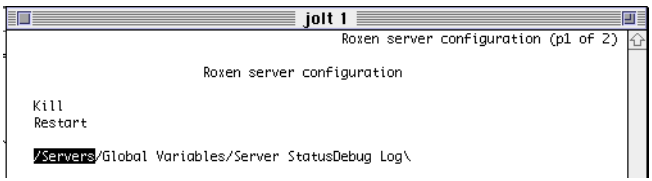

FIGURE 7.17 *An example of tablists in a text-only browser.*

Attributes

- `names=name1;name2;name3 ...`; The text that should be seen in each of the tabs.
- `1=url1 2=url2 3=url3 ...`; The URL:s corresponding to the different tabs. The list of URL:s must be in the same order as the list of names.
- `selected=[number]`; initially selected tab number.

- bg=#rrggbb; background color.
- tc=#rrggbb; tab color.
- fc=#rrggbb; font color.
- font=<font name>; font file name.
- scale=<float number>; Image scaling factor, default is 1.0.

Example 7.19)

```
<tablist selected=2 names=Orange;Banana;Cucumber
1=orange.html 2=banana.html 3=cucumber.html>
```

<USER>

Insert the real name and email address of a user.

Attributes

- name=username; Insert data about this user, modified by realname, email and nolink, see below.
- realname; Insert only the real name of the user, modified by link, see below.
- email; Insert only the email address of the user, modified by link, see below.
- link; Link the text consisting of the user's name, Real Name, to the home page of the user, and the email to a mailto: link. This is the default for the default action (i.e. with no email or realname modifiers present)
- nolink; Add no links whatsoever. This is the default when any of the email and realname modifiers are present.

The default output is "Real Name <user@your.domain>".

ATTRIBUTES RELATED TO TIME AND DATES

These attributes can be added to RXML tags related to dates, like <modified> and <accessed>.

- type=discordian|stardate; These attributes only make a difference when *not* using part (see below). Note that stardate has a separate companion attribute, prec, which sets the precision.
- type=number|string|roman

- `lang=language`; When using `type=string`, return the equivalent in the given language. This also affects the `part` attribute, see below. Available languages are Swedish (se), Finnish (fi), German (de), Catala (ca or es_CA), Dutch (du), Spanish (es), French (fr), English (en, default) and Norwegian (no).
- `part=year|month|day|date|hour|minute|second|yday`
  - `year`; The year
  - `month`; The month
  - `day`; The weekday, starting with Sunday.
  - `date`; The number of days since the first this month.
  - `hour`; The number of hours since midnight.
  - `minute`; The number of minutes since the last full hour.
  - `second`; The number of seconds since the last full minute.
  - `yday`; The day since the first of january.

  The return value of these parts are modified by both `type` and `lang`.
- `time`: Only show the time.
- `date`: Only show the date.
- `capitalize`: Capitalize the string.
- `lower`: Lowercase the string.
- `upper`: Uppercase the string.

EXAMPLES OF HTML AND RXML

In order to illustrate how one can use these tags, this section contains a couple of pages, both source and screenshots of the results when sent to the Netscape Navigator. If you wish to see the resulting HTML type them in and try them, then choose to view source from the browser. The examples are sparsely commented but they are quite simple and should give you a rough idea about what you can do.

Screenshots are from the Macintosh and besides the Netscape Navigator, I used the Alpha programmer's editor to build the code, NSCA Telnet to connect to our server, so that I could run Lynx, and Fetch to upload the

files.

I decided to build an ego-boosting homepage and some nice examples of how the HTML tags <FONT> and <Hn> look. Since I want to use a simple, consistent layout on all pages I first build a macro file for use at the top and at the bottom of the pages, see figure 7.18.

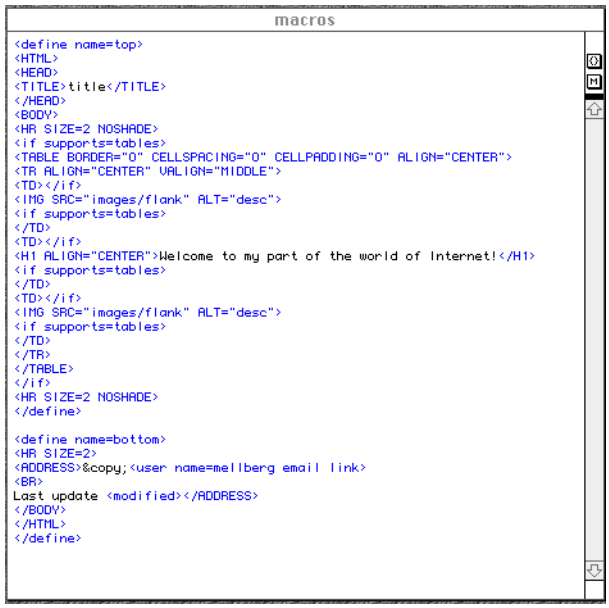

FIGURE 7.18 *The macro file containing only the definitions of header and footer. N.B.: There's no extension on the filename.*

Then I use the macros to build a simple homepage, figure 7.19.

The result when using Netscape Navigator is in figure 7.20. If you use Lynx, the result will look like figure 7.21.

Ok, then we have the two example pages in figure 7.22 and you see the results in figure 7.23 and in figure 7.24.

Now there are a few things that are not obvious;

The file macros does not have an extension. The reason for this is that if you put .html or even .rxml (due to the configuration of our server) at the end of the name, the file will be parsed before being inserted and we

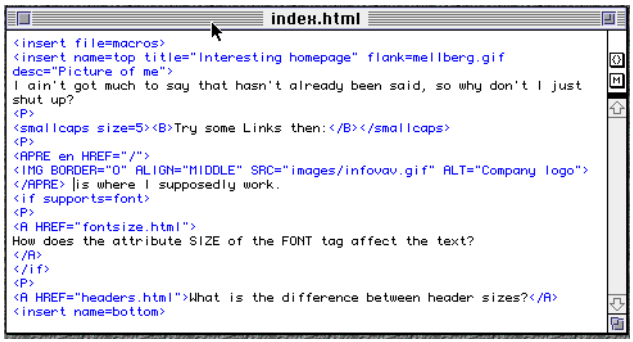

FIGURE 7.19 *Let's write a nice homepage!*

don't want that in this case. Sometimes it might be useful but not here. Which extensions that should result in Roxen parsing the file is configurable.

You can be quite mean to Roxen, putting RXML tags almost anywhere as you can see in the return link (TURN BACK!) where I have used `<referer>`.

Well, as for other strange things, look at the examples and play around with them to see what they do. Good luck!

FIGURE 7.20 *Looks nice, doesn't it?*

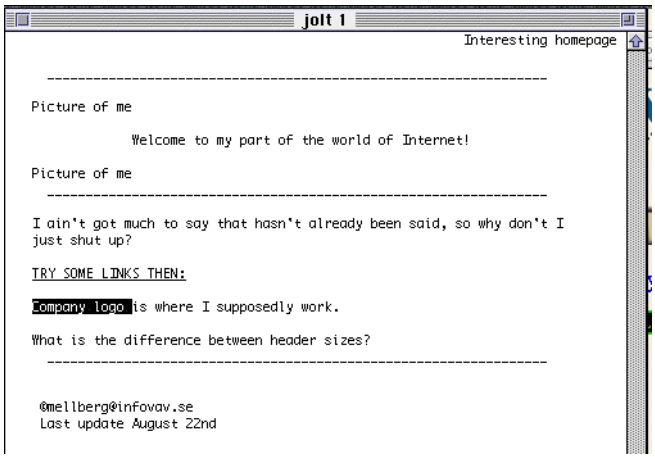

FIGURE 7.21 *The Lynx version. See how one of the links is not present? This is due to the fact that the link is not interesting for the user of Lynx.*

headers.html

```
<insert file=macros>
<insert name=top title="Header test" flank=infovav.gif desc="Company logo">
<H1>Example of header of size 1</H1>
<H2>Example of header of size 2</H2>
<H3>Example of header of size 3</H3>
<H4>Example of header of size 4</H4>
<H5>Example of header of size 5</H5>
<H6>Example of header of size 6</H6>
<if referer>
<P>
<A HREF="<referer>">TURN BACK!</A>
</if>
<insert name=bottom email=realname>
```

fontsize.html

```
<insert name=top title="Font size test" flank=infovav.gif desc="Company logo">
<P>
<FONT SIZE="1">Example of setting fontsize to 1</FONT>
<P>
<FONT SIZE="2">Example of setting fontsize to 2</FONT>
<P>
<FONT SIZE="3">Example of setting fontsize to 3</FONT>
<P>
<FONT SIZE="4">Example of setting fontsize to 4</FONT>
<P>
<FONT SIZE="5">Example of setting fontsize to 5</FONT>
<P>
<FONT SIZE="6">Example of setting fontsize to 6</FONT>
<P>
<FONT SIZE="7">Example of setting fontsize to 7</FONT>
<if referer>
<P><A HREF="<referer>">TURN BACK!</A>
</if>
<insert name=bottom email=realname>
```

FIGURE 7.22 *The example pages. The results are in the next two figures.*

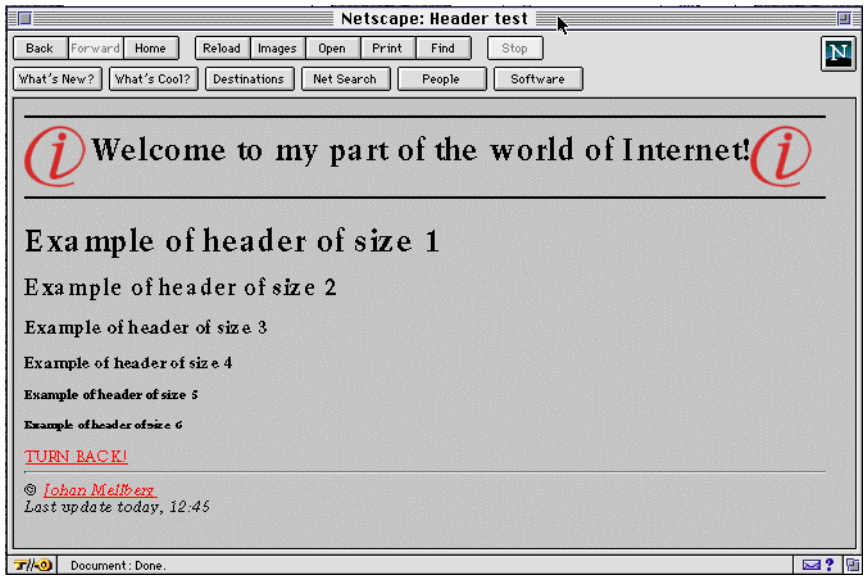

FIGURE 7.23 *The header test page.*

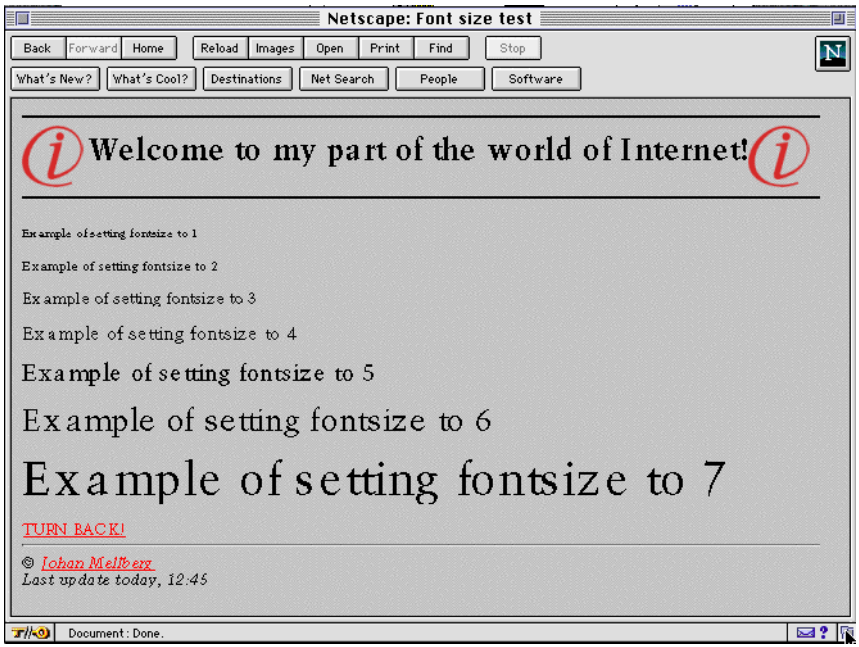

FIGURE 7.24 *And the font size test page.*

CHAPTER 8

IMAGE MAPS

INTRODUCTION TO IMAGE MAPS

An image map is an image on which has been defined several sub-areas, often called *hot spots*, each one associated with a certain URL. Clicking on one of these hot spots takes you to the associated URL as if it had been an ordinary text link.

An image map ordinarily consists of two files; the image file itself and the associated *map file* in which should be defined which area should be mapped to which URL.

The two most common map file formats are the NCSA and CERN formats. The Netscape Navigator (starting with version 2.0) and the Microsoft Internet Explorer can handle the mapping definitions as part of the HTML-file, so-called client side image maps, thus eliminating the need to separately load a map file.

HOW TO INSERT AN IMAGE MAP ON A PAGE

You insert an image by writing something like

```
<IMG SRC="/images/choices.gif">
```

Making this image a link is, as for text, done by enclosing the image in the <A> container, like this:

```
<A HREF="choice1.html"><IMG SRC="/images/choi-
```

```
ces.gif"></A>
```

If you wish this image to have several hot spots you have to tell the browser that the image is an image map by adding the attribute `ISMAP` to the `<IMG>` tag. Then, to associate certain areas with certain URL:s, you have to provide the URL to the file that contains the definitions of these areas in the `<A>` tag:

```
<A HREF="/images/choices.map"><IMG SRC="/ima-
ges/choices.gif" ISMAP></A>
```

The image map file is a text file whose contents have a special structure as we will discuss below.

ROXEN AND IMAGE MAPS

Roxen has support for ISMAP images in the form of a separate module, the *ISMAP Image map* module. Roxen has its own map file format but this module also has full built-in support for NCSA and CERN mapfiles. You can, if you want to, mix these three mapfile formats in the same file. Accordingly, there are often three (very similar) ways of doing things.

If you come across any other mapfile formats, feel free to send us an example file, so we can add support for that format as well. Most servers can use one or both of the first two formats (NCSA and CERN).

The notable variable in the image map module is the mapfile extension All files ending with this extension, will be parsed as map-files. The default is `.map`.

Note that with Roxen you can put your `.map` files anywhere in the virtual filesystem, you are not limited to using a separate directory, since the image map module is part of Roxen, not a separate CGI-script as for other servers.

IMAGE MAP FILE FORMATS

In defining a map file you define areas of the image by providing pixel coordinates of suitable corners, vertices. You can define rectangles, circles and even polygons. After having defined an area you associate it with a

URL. Finally, you should define some kind of default URL which is used if the user clicks on a part of the image that is not covered by any defined area(s).

Whitespace does not have any meaning whatsoever, and can be inserted at will. All lines starting with a "#" are considered comments. The following are the possible methods of creating hot spots. Coordinates are given in pixels and the origin of both the X-axis and the Y-axis is the upper left corner.

Note that areas are checked in the order they appear appear in the map file so in the case of overlapping, the first to appear takes precedence.

The URL may be full or relative, but be aware that the relative URL would be relative to the directory of the map file. The advantage is that when you don't provide a full URL the server does not have to send a redirect to the browser, thus saving time and bandwidth.

CERN

One of the first WWW servers to emerge, the CERN httpd, implements image maps using `.map` files and four keywords on the following form:

- `default URL`

  The url `URL` will be returned if nothing else matched. Don't forget to set it.

- `circle (X,Y) R URL`

  Any point inside the circle centered in (X,Y) and with the radius R will return the url `URL`.

- `rectangle (X1, Y1) (X2, Y2) URL`

  (X1, Y1) are the coordinates of the upper left corner of arectangular area whose lower right corner has the coordinates (X2, Y2). Any point inside the box will return the url `URL`.

- `polygon (X1, Y1) (X2, Y2) ... (Xn, Yn) URL`

  Every value pair is a vertice. The first and last should be the same thus closing the area, but if they are not Roxen will (as `htimage` in httpd does) complete the series of adjacent vertices.

The keywords may be abbreviated as `def`, `circ`, `rect` and `poly` respectively.

NCSA/APACHE

This format is almost identical but with one change; the URL should come before the coordinates. There is also one addition to the methods available:

- `point URL (X,Y)`

  This just specifies a single point and ties a URL to it. If more than one point is specified in the file, the one closest to the position on which the user clicks will be used.

ROXEN

As you can see, the ”method” of the NCSA and CERN formats need not be present, Roxen will deduce the area type from the given coordinates. You can also use colours as representatives of URL:s.

- `(X1,Y1)-(X2,Y2) URL`

  (X1, Y1) are the coordinates of the upper left corner of arectangular area whose lower right corner has the coordinates (X2, Y2). Any point inside the box will return the url URL.

- `(X,Y),R URL`

  Any point inside the circle centered in (X,Y) and with the radius R will return the url URL.

- `(X,Y) URL`

  This just specifies a single point and ties a URL to it. If more than one point is specified in the file, the one closest to the position on which the user clicks will be used.

There are methods of describing an image file in a big, but easily decoded, binary format. These files can be used to map colours to URL:s

- `ppm:PPM_file`

  Use the PPM file referred to by the absolute filesystem path PPM file. Each colour in that file may give a different URL.

- `pgm:PGM_file`

  As PPM, but the file is a greyscale file.

- `color:(r,g,b):URL`

  In all PPM files referenced, this colour will point to the document URL. r,g and b are decimal integers between 0 and 255 and the colour defined is the combination of the red (r), green (g) and blue (b) intensities. If the file searched is a PGM (greyscale) picture, the greyscale will be (r+g+b) / 3.
- `color:(r,g,b)-(r,g,b):URL`

  All colours in the range will point to URL. If the file searched is a PGM (greyscale) picture, the greyscale will be (r+g+b) / 3.
- `color:greyscale-greyscale:URL`

  All colors with an intensity falling within the range will point to URL.
- `color:greyscale:URL`

  All colours with the intensity `greyscale` will point to URL.
- `default:URL`

  The url URL will be returned if nothing else matched. Don't forget to set it.
- `void URL`

  The url URL will be returned if the client doesn't support imagemaps or if the mapfile is accessed without coordinates.

CLIENT-SIDE IMAGE MAPS

This method is faster because it is server independent. The first implementation was made by Netscape and now the Microsoft Internet Explorer also supports this style of image maps. What makes this different from server-side image maps is that the map information resides in an HTML file, usually in the same file as the <IMG> tag, instead of in a separate file.

HOW TO INCLUDE CLIENT-SIDE IMAGE MAPS

By adding the USEMAP attribute to the <IMG> tag you tell the browser that this is a client-side image map. With USEMAP you also tell where the mapping definitions reside:

```
<IMG SRC="/images/choices.gif" USEMAP="/image-
```

```
maps.html#choicesmap">
```

This line will only work in a browser that supports client-side image maps. The USEMAP=... tells the browser to look in a file called **imagemaps.html** (in the root of the web server) for an image map definition called choicesmap. If you leave out the path (what is *before* the "#"), the browser assumes that the map definition resides in the current file. Cf. "Links" on page 39.

To make an image map that works regardless of the browser used, include provisions for both client-side and server-side image maps, respectively:

```
<A HREF="/images/choices.map"><IMG SRC="/ima-
ges/choices.gif" USEMAP="/imagemaps.html#choices-
map" ISMAP></A>
```

HOW TO DESCRIBE THE DIFFERENT AREAS

This is done through use of the <MAP> and the <AREA> tags:

```
<MAP NAME="name">
<AREA [SHAPE="shape"] COORDS="x,y,..."
[HREF="URL"] [NOHREF]>
.
.
.
</MAP>
```

Within <MAP>...</MAP> there can be as many <AREA> as you like but the first of two areas that overlap each other will take precedence.

- name is the name referenced in the USEMAP attribute after the "#". Since you can have several map definitions in a file, make sure you give each one a unique name.
- shape is either RECT (rectangle), POLY (polygon), CIRCLE (circle) or DEFAULT (areas not covered by other shapes) and tells how the coordinates in the COORDS attribute should be interpreted. If you do not supply the shape, SHAPE="RECT" is assumed.
- The coordinates are given in pixels and in the upper, left corner x=y=0. The x-axis is the horisontal axis and the y-axis is the vertical axis.
  - Rectangular coordinates are given as COORDS="xleft, yupper, xright, ylower".

  - Circular areas are defined by `COORDS="xcenter, ycenter, radius"`.
  - Polygons are defined by providing a series of (x, y) pairs that, when connected by straight lines enclose the desired area, like this: `COORDS="x1,y1,x2,y2,x3,y4,..."`.
- The `URL` of course indicates the associated URL.
- If `NOHREF` is included, the area is not associated with any URL. You did figure that out yourself, didn't you?

N.B.: Relative URL:s are relative to the location of the file that contains the map description, i.e. not necessarily the file containing the image reference itself.

More information can be obtained through Netscape's WWW server, http://home.netscape.com/.

CHAPTER 9

SCRIPTING WITH ROXEN

Using Roxen you can use both the well-known CGI as well as make scripts in Pike, the native language of Roxen. In this chapter we will explain how to write Pike scripts and also take little look at CGI scripts and Roxen.

PIKE SCRIPTS

We assume that you have some knowledge of programming in general and of Pike in particular, refer to "The Pike Quick Guide" on page 143 for a short introduction.

A Pike script is a piece of code (in fact a Pike object) that is run when the URL of the script is requested, i.e. instead of returning the script itself, the result of the script is returned. The definition of a script is very simple. Since Pike is an interpreted language it is sufficient to create a file with your script in it, and then refer to it with a URL.

Example 9.1)A well known test case

```
string|mapping parse(object request_id)
{
   return "Hello world!\n";
}
```

Create a file with the above content, call it test.pike and put it some-

where in the virtual file system of your web server. When you've done that, access the file with your favourite browser, e.g. http://www.whatever.domain/path/test.pike. You will see a page with nothing but the plain text "Hello world".

Now, since you've seen a lot of interesting things that Roxen can do in the previous pages, how do you go about using them in your Pike scripts, i.e. taking care of variables from forms, prestates etc? All this information resides in the object `request_id`. This object is the single most important thing for Roxen's taking care of different requests and thus for your scripts too! Below we shortly describe what you can obtain from this object.

CONTENTS OF REQUEST_ID

You can reference the contents of this object through:

```
request_id->variable_name.
```

You'll find examples below that illustrate this.

- `int time;`

  When the connection was established measured in seconds since 00:00:00, the first of January 1970.

- `object conf;`

  This object contains the current configuration, a pointer to the virtual server to which the request was sent, i.e. the server in which your module or script resides. You will probably not need this variable if you don't intend to perform some very odd operations.

- `string raw_url;`

  This string contains the URL completely unparsed as it was sent from the client.

- `mapping (string:string) variables;`

  This mapping contains all those nice variables you can send over the HTTP protocol. Most of the time they come from an HTML form. In this mapping they are already parsed and ready to use.

  Example 9.2)

A request for the file `/goo/bar?hmm=hej&foo=%20` will set `variables` to `([ "hmm":"hej", "foo":" "])`.

- `mapping (string:array (string)) misc`

  As the name implies this mapping contains a little bit of this and a little bit of that. Usually you don't have to care but to see the contents you can do a test script with the following line:

  ```
  return sprintf("%O", request_id->misc);
  ```

  However, this is also where you define your own variables in case you against all odds should need any. Make sure that all variables have unique names to avoid nasty surprises. Variables are local for every request.

- `multiset (string) prestate;`

  Prestates provide variables that can be used to affect Roxen's treatment of a request. The prestate variable is a list of strings. They can for example, be used like this,

  ```
  if(request_id->prestate->nobg) no_background = 1;
  ```

  which checks if the prestate for no background is active and in that case sets the `no_background` variable to 1, i.e. true.

  Prestates are included in the URL before the name of the file requested, like this, `/(foo,bar)/goo/bar`.

- `multiset (string) config;`

  Configurations works just like *prestates* except they aren't kept in the URL, but in client-side cookies, which entails several advantages:

  - The config is not in the URL, where it perhaps confuses people.
  - The config will not disappear when the user turns off his or her browser, but will reside in a cookie sent with the request to the URL where the cookie came from the next time.
  - Links in bookmark files will not include the configuration.

  Not to forget the disadvantages:

  - There cannot be more than one list of configurations for every server.
  - It is hard for the user to get rid of a configuration.
  - Cookies are not widely implemented in browsers.

  In order to add something to `config` you let the user access a file on the following form;

  ```
  http://your.server:port/<+config,-config,...>/re-
  ```

al/URL.

As you can see, you add to `config` by using the ”+” and remove by using ”-”.

What happens is that the new and updated config will be added to the cookie *Roxen-Config* and the client will recieve a redirect from Roxen.

- `multiset (string) supports;`

  The string that contains a list of what the client can handle. The contents originate from etc/supports by default, which can be changed in the configuration interface. Use it, for example, to conditionally generate different code for different clients.

  Example 9.3)

```
if(request_id->supports->tables)
   return make_table();
else
   return make_pre();
```

- `string remoteaddr;`

  This string contains the IP-number of the computer on the other end of the connection, e.g. ”158.126.90.157”.

- `array (string) client;`

  This is the client that requests the page. The reason for it being an array is that the HTTP/1.* protocol allows multiple User-Agent: header rows. You never know...

  Ordinarily you should use the string operator in order to obtain a string (`request_id->client*””`), but sometimes it may be better to use the `supports` list.

- `array (string) referer;`

  The (one or several) page(s) that referred to the current page.

- `multiset (string) pragma;`

  This list contains all the pragma headers the client has sent. What is interesting for the programmer here is the no-cache pragma header that is generated when the browser’s RELOAD button is pressed. Read more about pragma headers in the HTTP specifications.

- `string prot;`

  The protocol which was used to generate the request, most probably it will contain ”HTTP/1.0” or ”HTTP/1.1”, but you may also encounter things like ”FTP”, ”GOPHER” and ”HTTP/0.9”.

- `string method;`

  The method used to send all the data from a form to the server. It is either ”GET” or ”POST”.

- `string rest_query;`

  Everything in the URL behind a ”?” which is not a variable.

- `string raw;`

  The complete request in its raw, unparsed, form.

- `string query;`

  Everything in the URL behind the ”?”.

- `string not_query;`

  Contains everything before the ”?” in the URL, excluding prestates. This variable is used by Roxen to calculate through which modules the request should be mapped.

- `string data;`

  This contains everything in a body of a request. It isn’t used very often but when form data is sent using `method=POST` it is used for all the form variables.

- `array (int|string) auth;`

  This one is either,

  - 0; No authentication sent by the client
  - `({ 0, ”username:password” })`; authentication sent but the Auth module doesn’t consider it to be correct and the user does not exist
  - `({ 0, ”username” })`; authentication sent but the Auth module doesn’t consider it to be correct although the user does exist
  - `({ 1, ”username” })`; authentication sent and is considered correct

- `mapping (string:string) cookies;`

  All the cookies sent by the client.

Now that we can obtain all this data we can do something with it and then, most probably, we return some other data. How this is done? Read on!

RETURNING DATA

The simplest case is to return a string. This string is sent through the parser of Roxen. You can also return a mapping, but it can be a bit difficult to put together the correct kind of mapping. In order to help with this, there are several help functions defined in pike/http.pike, which is inherited by roxenlib, which you in turn should inherit in all your scripts by including the line `inherit "roxenlib"`, if you intend to use any of these functions.

By the way, `some_type|void` means that the value does not have to be present when calling the function.

- `mapping http_string_answer(string data, string|void type);`

  Return a string as the result whose type is `text/html` if nothing else is set. The string will not be parsed by Roxen before being returned to the client.

- `mapping http_file_answer(object fp, string|void type, void|int len);`

  Return a file as the result whose type is `text/html` if nothing else is set. If you do not provide the length, Roxen will calculate it anyway. The object fp has to be an instance of precompiled/file, or an object implementing the same methods, which is something you definitely do not want to try. Use precompiled/file.

- `mapping http_redirect( string url, object|void request_id );`

  Return a redirect to the given URL. Sending along request_id in addition to providing a relative URL causes both `prestate` and `state` to be added to the URL.

- `mapping http_auth_failed(string realm);`

  Return a demand for a password (authentication) within the namespace `realm` on this server. The Netscape Navigator retains one username and password pair for every server in memory. This mapping is, as might be inferred from the name, used when the login failed.

- `mapping http_auth_required(string realm, string message);`

  This returns almost the same as `http_auth_failed` but you can send along a small error message which is shown if the user chooses Cancel. In addition, the code used is somewhat different. As the name implies, `http_auth_required` indicates that this should be used when the user has not tried to log in before.

  No web browser we have seen distinguishes between the above two results.

- `mapping http_low_answer(int error_code, string message);`

  This function returns a message with `error_code` as the error code (unexpected, eh?). Take a look in **server / include / variables.h**, in figure C.3 on page 204 or in the HTTP specification for a list of these error codes.

  Example 9.4)

  ```
  if(search(lower_case(request_id->client*""),
  "aol") != -1) return http_low_answer(402, "Please
  enter your VISA number and expiration date be-
  low:\n");
  ```

  If you enjoy details, below is the returned mapping. All fields can be left out but the resulting response wouldn't amount to much.

  ```
  ([ "file":file_object, "data":"string",
  "len":int, "type":"main/sub", "raw":0|1,
  "leave_me":0|1, "extra_heads":([ "name":"value,
  ... ]), "error":int, ])
  ```

  - `file`; A file object that should be sent back to the client. If there is `data` present, `file` is sent after that.
  - `data`; This is a string that should be sent before `file`.
  - `len`; The sum of the lengths of `data` and `file`. If you do not specify it, Roxen will calculate it anyway.
  - `type`; This is the mime type of what you return. If it is not defined, `text/plain` will be used.
  - `raw`; If you set this variable you will have to henerate all the headers and such by yourself because nothing at all will be sent to the client.

- `leave_me`; Setting this variable will tell Roxen not to send anything to the client, and so you are expected to take care of all the socket communication and clean-up by yourself. This approach is used in the proxy modules.
- `extra_heads`; Extra header fields that will be added to the response. One such that could be useful is
  `([ "Expire":http_date(time()), ])`
  which will cause the return value of the the script to expire immediately.
- `error HTTP`; The response code that should be used, usually 200. All the codes are defines in server/include/variables.h, probably the most unintuitively named file in all of the Roxen distribution. You can also find them in figure C.3 on page 204.

IMPORTANT NOTES

Scripts are not allowed to execute for longer than four seconds. If this limit is transgressed, the interpreter returns an error, which it of course is possible to trap, analyse or possibly ignore. The reason for this is that the script blocks the server during this time. If you don't mind blocking the server, it is possible to set a longer execution time limit in the Pike script module.

When the script is running it blocks the server, i.e. nothing else is done during this time.

If you run several servers, take note of the fact that the script can run on all of them at once.

If you hit RELOAD the script is reloaded. It is possible to hinder this by implementing a method, for example like this, in your script:

Example 9.5)

```
int no_reload(object request_id)
{
   if(!request_id->variables->reload)
     return 1;
   return 0;
}
```

Comments are included by preceding the comment with double slashes, "//". You can also use C-style comments, see the "Example Script" below.

EXAMPLE SCRIPT

This is an example of a short script that reads a file with names of images, separated by a newline. It returns a random image.

```
inherit "roxenlib";
// Has some nice functions, notably the
// http_* functions.

array (string)files;
// An array of strings; the image filenames

#define FILELIST "/the/full/path/to/the/filelist"
// Absolute path to the image list.

#define BASEDIR "/"
// Path to be prepended to the filenames in the
// list

/* Create() is called when the script loads. Reads
* the file from the disk and explodes it into an
* array
*/
void create()
{
   files = read_bytes(FILELIST)/"\n";
}

/* Don't reload the module from file, unless the
* creator wants to. Call the script with
* '/random.pike?reload=whatever', to reload it.
*/
int no_reload(object id)
{
   if(!id->variables->reload)
     return 1;
   return 0;
}
```

```
/* parse() is called every time someone
* requests the URL of the script. We simply use
* a function defined in roxenlib (really in a
* file inherited from the above mentioned
* library), http_redirect, to send a redirect to
*one of the files from the list.
*/
mapping parse(object id)
{
   // Return a redirect to a random file
   return http_redirect(BASEDIR +
   files[ random(sizeof(files)) ], id);
}
```

THE `parse()` FUNCTION

When someone accesses the URL of a Pike script, the function `parse()` has to be called in it, i.e. you have to put it there in your scripts, with one argument; `object request_id`. Read about `request_id` in "Contents of request_id" on page 86 and you'll understand that this is where all the interesting information about the request resides.

CGI

Of course we'd prefer you to use Pike scripts since it is more efficient and also more secure as everything is handled internally in Roxen. Nevertheless, we see the need for use of CGI with Roxen so if you've installed the CGI module in the virtual server you can make CGI scripts. We've also included support for FastCGI.

Many servers offer small extensions of CGI so that certain scripts are incompatible with other servers. Thanks to all beta testers we have endeavoured to make Roxen as compatible as possible with as many homebrew scripts as possible. If a CGI script that you wish to use follows the original NSCA standard it should work like a charm, otherwise please send us a report on roxen-bugs@roxen.com.

ROXEN ADMINISTRATOR'S GUIDE

Detta dokument har framställts med FrameMaker 4.0.4

CHAPTER 10

GENERAL INFORMATION

As opposed to merely using Roxen's features when putting your information together, someone has to be responsible for the actual running of the Roxen server. This person, the administrator, must be more intimately familiar with all aspects of Roxen than the ordinary user. This part of the manual aims to explain some of the details of Roxen. Note, though, that there is a lot of information in this part of the manual that is probably very useful for any user of Roxen.

THE ROXEN CONCEPT

In short, Roxen is built upon the concept of modularity. The core of Roxen cannot, by itself, do anything. The only thing the Roxen core can do is talk to modules. This means that before you can start using your server you have tell it how to treat files, how to do all the things a normal web server does. You do this by adding modules that take care of different aspects of Roxen's functionality. This means that you can make your web server as complex or as simple as you wish, all depending on needs and wants of the people using it to provide information on the Internet.

In "Configuring Roxen – the first steps" we showed how to set up a basic WWW server. The modules that are necessary for basic operation are the *User database* module and the *Filesystem* module in addition to the de-

fault modules *Contenttypes* and *Main RXML Parser*. Actually, the RXML parser and the user database are not strictly essential to the web server but they are necessary for many of Roxen's special features. In the next chapter you will find information specifically pertaining to each and every module.

Remember that in order to access all information pertaining to a certain node, be it a virtual server or a module, it is necessary to focus on it, cf. "The fold / unfold principle and focusing" on page 17. If we focus on a virtual server it looks something like figure 10.1.

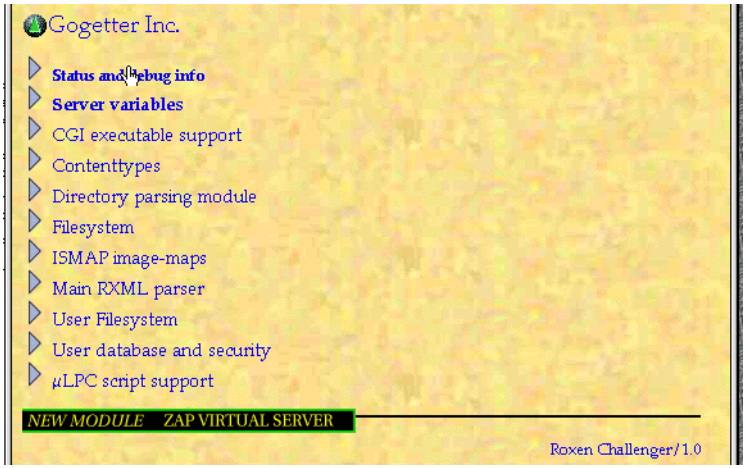

FIGURE 10.1 *Focus on a virtual server.*

Now we will first take a look at variables.

VARIABLES

Roxen has a number of global variables, namely, *Configuration variables*, *Disk Cache Variables* and *Logging Variables*.

Apart from these, there are also variables that are independently settable in every virtual server. These variables are of a general nature, but they also concern security and the customisation of logging.

CONFIGURATION INTERFACE VARIABLES

All these variables reside under the *Configuration interface...* header under the GLOBAL VARIABLES tab.

- `IP pattern`

  Only clients running on computers with IP numbers matching this pattern will be able to use the configuration interface. This is a way to enhance the security of the configuration interface.

- `Ports`

  This is the ports via which you can configure the server. Initially there should be one, but you can of course change this. *Do not delete the last one!* Change the port number if you want to but *do not delete the last one!* You may want to tattoo this onto the inside of your eyelids.

- `Password`

  Here you can change the configuration password. To prevent errors there is also a form available.

- `User`

  Set the user name that should accompany the password.

- `URL`

  The URL of the configuration interface. The default is `http://{Configuration interface IP}:{Configuration port}/`. This value will be used for all redirects generated during configuration.

- `Background`

  If set to `No`, the background color will be set to standard grey.

- `Compact layout`

  When set removes some of the images from the configuration interface and replaces image maps with text links instead.

- `Help texts`

  Shows descriptions of each and every possible variable. When you know what you are doing, you might want to turn them off.

PROXY DISK CACHE VARIABLES

The disk cache is used by the HTTP proxy, the Gopher gateway and the

FTP gateway.

- `Enabled`

  If set, caching will be enabled. This will speed up most accesses outside your domain quite alot, especially if you have a slow Internet connection.

The following are only visible if the cache is enabled.

- `Base Cache Dir`

  This is the base directory where cached files will reside.

  To avoid mishaps, the cache is saved in the directory roxen_cache/ in this directory.

- `Bytes per second`

  How file size should be treated during garbage collect. Larger files will be removed first. It has to be an integer.

- `Clean size`

  Minimum number of megabytes removed when a garbage collect is done.It has to be an integer.

- `Size`

  How many Mbytes may the cache grow to before a garbage collect is done?

- `File name method`

  You can choose any of the methods Hash, Flat and Hierarchy. If you change method, the old cache is rendered unusable and may as well be trashed. The Hash method creates a hash table and directories corresponding to the table entries and cached files are saved in the corresponding directories. The Flat method just saves files in a directory and Hierarchy saves the cached files in a hierarchy of directories that looks like the hierarchy of the web server.

LOGGING VARIABLES

- `Logging Method`

  The method to use for logging. Default logging to file is used, but it's also possible to enable syslog logging. Try `man syslog` if you don’t know what it is. `Syslog` is quite slow, though.

The variables below are only present if you have chosen `syslog` above.

The syslog variables reside under a separate node.

- `Log PID`

  If enabled, the PID (Process ID) will be included in the syslog.

  `Roxen[4711]: Error: Flep flop`

  instead of just

  `Roxen: Error: Flep flop`

- `Log as`

  When syslog is used, this is the identification that Roxen uses.

- `Log to system console`

  If set and if syslog is used, the error / debug messages will be printed on the system console, as well as to the system log.

- `Log Type`

  When using syslog, which log type should be used?

- `Log what`

  When using syslog, how much information should be sent to it?

  - `Fatal`: Only messages about fatal errors
  - `Errors`: Only error or fatal messages
  - `Warning`: Warning messages as well
  - `Debug`: Debug messager as well
  - `All`: Everything

GENERAL GLOBAL VARIABLES

- `Change UID and GID to`

  See “Your first changes” on page 18 for a discussion of this.

- `Client supports regexps`

  This is a list of client names in the form of regular expressions, followed by a commaseparated list of features supported by all clients matching that regular expression. All lines beginning with ”#” are treated as comments. One special case is `default` which is used if

nothing else matches. If a client matches more than one regular expression, the supported features are ”summed” together. You can also include files;

`#include <relative/exact filepath>`

There is a default list included which you can study. If you have additions for this list, please send them to us. The file etc/supports is automatically updated now and then from Infovav, unless you turn it off.

The other case is `#section` which begins and ends individual browser sections. This simplifies the construction of the regular expressions pertaining to each and every browser. Take a look at the examples in etc/supports.

- `Update supports database automatically`

  If set to `Yes` the list of what browser(s) support which features will be updated automatically from our site.

- `Documentation URL`

  The URL to prepend to all documentation URLs throughout the server. This URL should not end with a ”/”.

- `Honor If-Modified-Since headers`

  If set, Roxen sends a `Not modified` response in reply to `if-modified-since` headers.

- `Identify as`

  What Roxen will call itself when talking to clients.

  N.B.: Revealing the specific software version of the server may allow the server machine to become more vulnerable to attacks against software that is known to contain security holes.

- `Module directories`

  A comma separated list of directories, where Roxen should look for modules. They can for example be paths relative from the server/ directory. If you install Roxen and decide to make your own modules, it might be a good idea having those in a special directory.

  By default there are three module directories in the server/ directory; modules/ (containing tried and tested modules), more_modules/ (containing modules that we use but haven’t thoroughly tested. Some of them were written by other people than us)

and the non_maintained_modules/ directory (with modules that lack documentation and aren't supported. Use them at your own peril).

- Number of accepts to attempt

  The maximum number of accepts to attempt for each read callback from the main socket. Increasing this will make the server faster for users making many simultaneous connections to it, or if you have a very busy server. It won't work on some systems, though, e.g. IBM AIX 3.2 To see if it works, change this variable, but don't press SAVE, and then try connecting to your server. If it works, go back and press the save button. If it doesn't work, just restart the server and be happy with having "1" in this field.

  If you have many virtual servers, it is not a good idea to have a high value in this field since it will place a great load on your machine.

- Number of hostname lookup processes

  The number of simultaneous hostname lookup processes that Roxen should run. The default value is 2, which should be more than enough on a normally loaded server. Consider raising the number of processes, if your server is heavily loaded.

- Number of copies to run

  The number of forked copies of Roxen to run simultaneously. This is quite useful if you have more than one CPU in your machine, or if you have a lot of slow NFS accesses. This must be an integer.

- PID file

  In this file, the server will write out its PID, and the PID of the start script. $pid will be replaced with the pid, and $uid with the uid of the user running the process.

- Set unique user id cookies

  If this is set, every client that visits your server and supports cookies will receive a unique cookie. This cookie can then be used to track individual users through the log files.

- Show the internals

  If set, the Internal server error messages will be relayed to the client. This can be very helpful when debugging your own modules or scripts.

SERVER SPECIFIC LOGGING VARIABLES

- `Enabled`

  Setting this to `No` turns off all logging for the server.

- `Format`

  The format to use when logging accesses. The syntax is:

  `response-code:Log format for that response code`

  or

  `*:Log format`

  The log format is normal characters, or one or more of the variables below:

  - `$host`; The remote host name, or IP number.
  - `$ip_number`; The remote ip number.
  - `$bin-ip_number`; The remote host id as a binary integer number.
  - `$cern_date`; Cern Common Log file format date.
  - `$bin-date`; Time, but as a 32 bit integer in network byte order.
  - `$method`; Request method.
  - `$resource`; Resource identifier.
  - `$protocol`; The protocol used (normally HTTP/1.0).
  - `$response`; The response code sent.
  - `$bin-response`; The response code sent as a binary short number.
  - `$length`; The length of the data section of the reply.
  - `$bin-length`; Same, but as an 32 bit integer in network byte order.
  - `$referer`; the header "REFERER" from the request, or "-" if the browser does not supply the referring URL.
  - `$user_agent`; the header "USER-AGENT" from the request, or "-".
  - `$user`; the name of the authentication that the user used, if any was used.

- `Log file`

  This is the name of the access log file. It can be stdout for standard output, stderr for standard error or a filename, whose path is relative to the server directory if it doesn't begin with a "/". It defaults to ../logs/Virtual_server_name/log, i.e. relative to Roxen's server/ directory. If left empty, no logging will take place.

- `No Logging for`

  Don't log requests from hosts with IP numbers matching any of the patterns in this comma separated list. This also affects the access counter log.

SERVER MESSAGES

- `FTP Welcome`

  Since Roxen can act as an FTP server this is where you put what Roxen should say to new connections in case the file welcome.msg doesn't exist.

  This variable may not be present in your configuration interface since it will be moved to the FTP protocol module.

- `No such file`

  This is the page that Roxen returns if someone tries to access a file or resource that doesn't exist. As any other page, this one gets parsed by Roxen, unless you have removed the html-parser in this virtual server.

  You can insert two variables on this page; `$File` will be replaced with the name of the resource requested and `$Me` with the URL of the server. If you want to use a file, you can use the `<insert file=XXX>` tag, see "<insert>" on page 52. Doing this makes it possible to change the page without entering the configuration interface.

GENERAL VIRTUAL SERVER VARIABLES

- `Configuration interface comment`

  This text will be visible in the configuration interface, it can be quite useful to use as a memory helper. Write whatever you want to be sure that those who administer Roxen read.

- `Configuration interface name`

  This is the name that will be used in the configuration interface instead of the actual name you entered when first creating the virtual server. If this is left empty, the actual name of the virtual server will be used.

- `Domain`

  Your domain name should be set automatically here. If it isn't, enter the real domainname and send a bug report to roxen-bugs@infovav.se.

- `Server URL`

  This is the location of your server. It's not necessarily correct, so if you change port and/or network interface, you should change this variable to the correct one. If someone accesses the directory /foobar, it is redirected to {Server URL}/foobar/. If you forget to change here before you save, Roxen will redirect your users to the wrong place.

- `Listen ports`

  The port(s) Roxen should bind this virtual server to. If you want Roxen to run on port 80 (the standard WWW port), Roxen must start as root.

  If your server has many ethernet interfaces or virtual interfaces, you can tell Roxen which interface this configuration should be bound to. The default is ANY which means that the virtual server should bind to all interfaces. See figure 10.2.

BUILTIN MODULE VARIABLES

- `Comment`

  An optional comment. It has no effect on the module and is only a text field for comments that the administrator might have (why the module is there, etc.)

- `Module name`

  An optional name. You can set it to something that reminds you of what the module really does, e.g. `WWW site main filesystem`. It has no effect on the module itself.

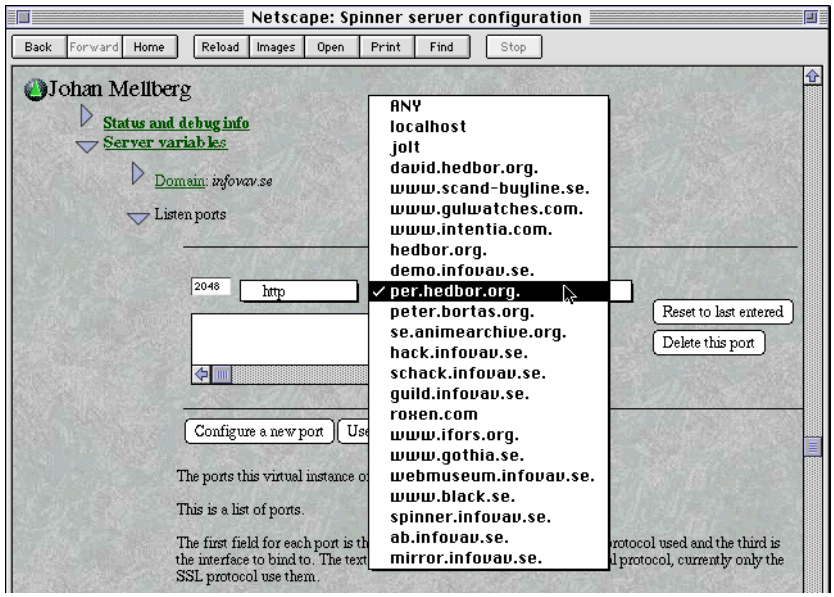

FIGURE 10.2 *Telling Roxen to which virtual interface to bind this virtual server.*

- `Priority`

  The priority of the module, i.e. in which order the module is called among modules of the same type. 9 is highest and 0 is lowest. Modules with the same priority are assumed to be called in random order.

MODULE SECURITY VARIABLES

- `Patterns`

  This is a list where you can set up very precise patterns for who will be allowed to use the module. It is a list where each entry is on the form

  ```
  security level=value
  ```

  The entries can be one or more from this list:

  - `allow ip=pattern`
  - `allow user=username,...`

- `deny ip=pattern`

  In patterns, ”*” is one or more characters and ”?” is one character.

  Using `any` as username stands for any valid account as indicated by .htaccess or other authentication type module. The default used when *no* entries are present is `allow ip=*`, allowing everyone to access the module.

- `Trust level`

  When a location module finds a file, that file will get a ”Trust level” that equals the level of the module. This file will then only be sent to modules with a *lower or equal* Trust level.

  As an example: If the trust level of a User filesystem is 1, and the CGI module has trust level 2, files from that file system will never get passed on to the CGI module.

  A trust level of 0 means that any file can be passed to the module; ”free access”.

SERVER STATUS

There are a couple of ways to obtain status about Roxen. On the configuration top page, there is a panel called *Server Status* and another called *Debug*. The debug log shows problems that have occurred. If you click on the server status, you will see the following nodes:

- `Access/request status`

  Nothing exciting, just a report on how much data has been sent out.

- `Process Status`

  Status about the Roxen server process, collected using the `getrusage()` system call or via `/proc/`, the process file system.

  NB: If you use Solaris and change `UID`, the server process won't be allowed to receive this information!

- `Pipe system status`

  Here you can see the number of open outputs and inputs, the number of `mmap`:ped files (and their total size) and more.

- `Host names`

  The number of hostname lookup processes and the size of the hostname lookup queue.

- `Memory cache system`

  Information about Roxen's memory cache system.

- `Open files`

  Information about the active file descriptors.

VIRTUAL SERVER STATUS

If you focus on or unfold a virtual server, you will see a node called *Status and debug info*. If you unfold it, you will see the status information for this virtual server, like sent data and number of requests. This is the server's part of the access / request count as seen in the status node for the server as a whole.

CHAPTER 11

MODULES

A module is an addition to the server, which adds to or modifies its functionality. There are a few different *types* of modules. For a description of the types see the first section of "How to make your own Roxen module" on page 161.

AVAILABLE MODULES

For a complete listing of all the basic modules that come with Roxen see table C.4.

Of these, there are of course a few required modules without which the server won't work as expected;

1. *Content-types module*

   This is the module that handles the mapping of extensions to content types. All other modules may also set the content type to whatever they want to, so this is a fallback for those that don't, like the default filesystem module.

   There must be a content types module present, but it can be replaced.

2. *Main RXML Parser*

   This module takes care of all RXML parsing. If there are other parser modules enabled, they are called from this module.

   If there is no Main RXML Parse module present, there will be no parsing at all.

3. *User database and security*

   The user database and security module manages the security. All modules use this module . User-made modules may also use it to get information about users or verify logins. The same goes for CGI and Pike scripts. See page 131 and page 133 for more information.

BOFH MODULE

This is a very simple module that only adds the tag `<bofh>`. What the tag does is that it inserts a random ”bofh” excuse, for whatever reason.

Usable when sending pages as a result of an unsuccessful request.

BOFH is an acronym for Bastard Operator From Hell.

CGI EXECUTABLE SUPPORT

This module can execute CGI-scripts both from a special directory and on extension basis. It supports the CGI/1.1 interface. Read more about this at http://hoohoo.ncsa.uiuc.edu/docs/cgi/interface.html.

Variables

- `Allow listing of /cgi-bin/ directory`

  If set, users can get a listing of all the files in the CGI-bin directory.

- `CGI-bin path`

  The location of this module in the namespace of the server. Usually, this is /cgi-bin/ for compatibility reasons. By default, the module will also service one or more extensions from anywhere in the name space of the server.

- `CGI-script extensions`

  All files ending with these extensions, will be parsed as CGI-scripts. For example, if you would like to run perl scripts, add `pl` to this comma separated list.

- `Extra environment variables`

  These are extras that can be sent to the script. The normal CGI variables will override these. The format is `NAME=value`.

- `Pass environment variables`

  If this is set, all environment variables seen by Roxen will be passed to CGI scripts, and not only those defined in the CGI/1.1 standard. Roxen also adds the CGI enhancements if they are defined, see below. This includes `LOGNAME` and all the others. For a quick test, you can try this script with and without this variable set, respectively:

  ```
  #!/bin/sh

  echo Content-type: text/plain
  echo ''
  env
  ```

- `Raw user info`

  If set, the raw, unparsed, user information will be sent to the script, in the `HTTP_AUTHORIZATION` environment variable. This is not recommended, but some popular ready-to-run scripts seem to need it.

  N.B.: This will give the scripts access to the password used, if any.

- `Search path`

  This is where the module should expect to find files in the real filesystem.

- `Search path`

  This is where the module should look for the CGI-binaries in the real file system.

- `Send decoded password`

  Setting this will cause the environment variable `REMOTE_PASSWORD` to be set to the decoded password value.

- `Send stderr to client`

  It you set this, standard error from the scripts will be redirected to the client instead of the `../logs/debug/»config_dir_name«.1` log.

- `Roxen CGI Enhancements`

  If set, the module will add a few extra environment variables on it's own, namely:

- ° VAR_variable_name or QUERY_variable_name

  Parsed form variable, like CGI parse. The parsed value of the form variable variable_name. That is, if you have an input field in an HTML form on the form <input name=name>, and the user types "J. Random" in that field, the evironment variable QUERY_name will be set to "J. Random".

- ° VARIABLES

  A space separated list of all variables in the form request, if any. This list consists only of the variable names.

- ° STATE_variable_name

  The parsed value of the state variable_name. A state is prepended to a URL with the add_state() function in the server. If the state variable module is set to config.pike, the environment variable STATE_module will be set to config.pike as well.

- ° STATES

  A space separated list of all state variables if any.

See also "Pike script support" on page 129.

CLIENT LOGGER

This module simply logs the user-agent field in the log file.

Variables

- Client log file

  This is the file into which all client names will be put.

CONFIGURATION INTERFACE

This module can be used to access the configuration interface from location in the normal file system, i.e. acting like any other file system. This can be very useful if you wish to do remote configuration of a server that sits behind a firewall.

Variables

- Mount point

  Where the configuration interface should exist in the virtual file system.

- `Allow anonymous read-only access`

  If set, anyone will be able to connect to, and read the configuration interface settings. This can be useful if you wish people to be able to learn from your work. Normally it should be off though.

CONNECT METHOD IMPLEMENTATION

This module implements the `CONNECT` method, useful for "tunneling" SSL connections. This is used in the Secure proxy server by Netscape Communications. Read more on this subject in the draft at http://www1.netscape.com/newsref/std/tunneling_ssl.html.

Variables

- `Allowed Ports`

  Connections will only be made to ports within the range given here. The syntax is `firstport-lastport` or just plain `port`. It might be desireable to disallow access to some ports, see the `Forbidden Ports` variable below. It is a comma separated list of strings

- `Connection refused message`

  The message to send when the requested host denies the connection.

- `Forbidden Ports`

  The syntax is as for `Allowed Ports`. This is a comma separated list of strings too.

- `No such host message`

  The message to send when the requested host cannot be found.

CONTENTTYPES

This module takes care of all the normal file extension to Content-type mapping.

Example 11.6)

Given the file foo.html, this module will set the content type to `text/html`.

Variables

- `Extensions`

  A list with extensions and their corresponding Content-types. The format is as follows:

```
Extension    type    encoding
```

Example 11.7)

```
gif                  image/gif
gz           STRIP   application/gnuzip
```

STRIP means that Roxen should strip this extension, and try again. Thus, a file named roxen.tar.gz would get the Content-encoding x-gzip and in addition the Content-type application/unix-tar instead of just the Content-encoding.

In Roxen, you can include files containing more mappings by typing:

```
#include <etc/extensions>
#include <etc/more-ext>
```

etc/extensions is included by default.

The complete list of types can be found at ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/media-types.

DEEP THOUGHT

This is only an example of a parser module. It simply adds a new tag, <dthought>. The main reason for including it is to show the interested programmer how to build a working module in Pike.

There are no variables to set in this module.

DIRECTORY PARSING

This is a directory parsing module, with a Machintosh lookalike directory tree.

N.B.: To get any directory parsing at all, you have to have a directory parsing module enabled, either this one or the simple directory module.

Features with this directory module includes folding/unfolding of directories, module virtual locations shown in the directory tree. Also, if you have overlapping modules (i.e. two filesystems mounted on the same location) the content of all of them will be shown.

Variables

- Separate hosts

  If this is set, the fold/unfold status will be different for each host accessing the server. Beware that this uses quite a lot of memory.

- `Index files`

  If any one of the files listed here is present in the directory requested, it will be sent instead of the directory listing.

- `Allow directory index file overrides`

  If `Yes`, you can force Roxen to send the directory listing even if there is an index file present by adding a dot to the request. This is very useful for ”debugging” whil building the site or trying out new scripts. However, it may be seen as a security hole and you can therefore turn this feature off if you wish.

  Example 11.8)

  `http://www.roxen.com/.`

- `Include file size`

  If `Yes`, file sizes will be included in the directory listings.

- `Include readme files`

  If `Yes`, README-files (i.e. README, README.html) will be inserted before the listing if they exist. In figure 11.1 there’s an example of the result.

EXPLICIT CLOCK

This module is only here as an example of a very simple location module. It shows what time it is or perhaps approximately, since time shown can be modified.

Variables

- `Mount Point`

  The location of the clock in Roxen’s virtual filesystem.

- `Time Modification`

  Time difference, in seconds, from the system clock.

FASTCGI

This module has support for the Fast-CGI interface. More information can be found on http://www.fastcgi.com/.

Variables

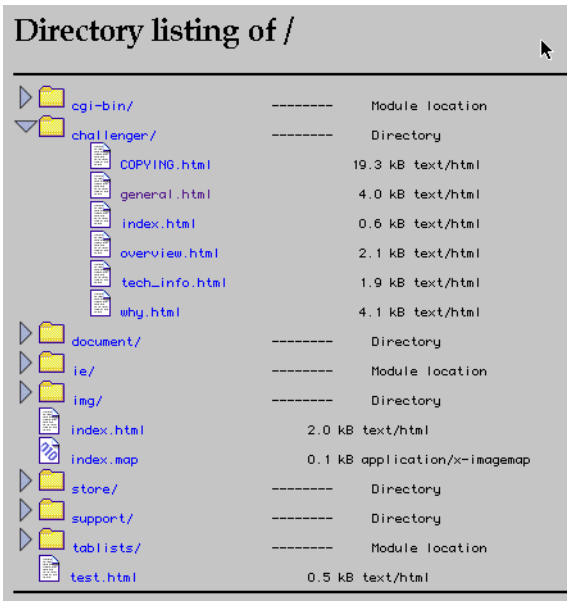

FIGURE 11.1 *Example of the pretty directory listing.*

- `Handle *.fcgi`

  This flag tells Roxen to treat all files with the extension .fcgi as well as the files in the cgi-bin / directory as fast-CGI scripts. This is a way of emulating the behaviour of the NCSA server. In the CGI module you can set what file extensions are to be handled.

For the other variables, see “CGI executable support” on page 112.

FAST DIRECTORY PARSING

This is a simple and fast directory parsing module. It is very fast since there are no fancy details as in the ordinary directory parser.

N.B.: To get any directory parsing at all, you must enable a directory parsing module.

Variables

- `Include readme files`

  If set to Yes, README-files (i.e. README, README.html) will be included in the listing.

- `Index files`

  If any one of the files listed here is present in the directory requested, it will be sent instead of the "No such file" response.

FILESYSTEM

The filesystem module is placed on a *Mount point* in the name space of the server, e.g. /doc/. This mount point is a "mapped to" location in the real file system, e.g. /usr/spider/doc/. The module makes files from the real file system available in the virtual filesystem of your web server.

The module tries to map all requests to files, e.g.:

```
/doc/s2.gif
```

maps to

```
/usr/spider/doc/s2.gif
```

and

```
/doc/tmp/test.html
```

maps to

```
/usr/spider/doc/tmp/test.html
```

If more than one module have the same mount point, the server will call them in priority order, and the first one that finds a file "wins". The end result is that directory listing will be the union of the files in all location modules that match the directory.

For example, if the (virtual) directory /foo/ is accessed, and one module is mounted on /foo/, and gets its files from /usr/www/foocustomer/, and another module is mounted on /, and gets its files from /usr/www/html, and there is a directory foo in /usr/www/html, the resulting list of files will be the union of all files in /usr/www/foocustomer and /usr/www/html/foo.

Variables

- `Mount point`

  This is where the module will be inserted in the name space of the server.

- `Search path`

  This is where the module will search for files in the real filesystem.

- Handle DELETE

  The DELETE action can be used to delete files in the filesystem.

- Handle PUT

  PUT can be used to upload files to the filesystem.

- Enable directory listings by default

  If set, you have to create a file named .www_not_browsable (or .nodiraccess) in a directory to *disable* directory listings. If unset, a file named .www_browsable in a directory will *enable* directory listings.

- Require authentication for modification

  Only allow authenticated users to use methods other than GET and POST. If unset, this filesystem will be a *very* public one allowing anyone editing access to files located on it.

- Show backup files

  If set to No, all files ending with "~" or "#" or ".bak" will be excluded from directory listings, since they are considered backups.

- Show hidden files

  If set to Yes, all hidden files will be included in directory listings, and become retrievable.

FTP GATEWAY

An FTP gateway, with support for remote proxies. It keeps connections alive to the FTP sites, for improved speed.

Variables

- Connection timeout

  The time in seconds that a connection to a ftp server is kept, after the last usage of it. When the time is up, the connection is closed.

- Data connection timeout

  Time in seconds before a data connection has timed out and cancelled.

- Connection timeout

  This is the time in seconds before a connection attempt is retried.

- `FTP transfer method`

  The method used to transfer files, active or passive. Both should work, but if there is a problem with a site, try switching method.

- `Hold until response`

  Hold data transfer until response from server; if the server sends file size, size will be sent to the http client. This may slow down access some, but not noticeably.

- `Icons`

  If set, icons are used in the directory listings.

- `Location`

  This is the location in the virtual filesystem, and the default value is `ftp:/`. If set to anything else, all normal WWW-clients will fail using it.

  The useful case where it would work with something else is `/ftp/`. If you set this location, a link formed like this: `<a href="/ftp/my.ftp.server/foo">myftpserver</a>` will enable accesses to local FTP servers through a firewall. Consider the security issues first though.

- `Logfile`

  This is the file name of the logfile. If left empty, no FTP logging will take place.

- `Port timeout`

  How long tim,e in seconds, a data port is kept open without usage, before it's closed.

- `Remote gateway regular expressions`

  Here you can add redirects to remote gateways. If a file is requested from a host matching a pattern, the gateway will query the FTP gateway server at the host and port specified. Hopefully, that gateway will then connect to the remote ftp server. Currently, the remote gateway has to be an http-ftp gateway like this one.

  Example 11.1)

```
# All hosts inside *.rydnet.lysator.liu.se have to
# be accessed through lysator.liu.se
.*\.rydnet\.lysator\.liu\.se 130.236.253.11 80
```

  Please note that these must be regular expressions.

- `Save dataports`

  Some FTP daemons have problems when the same port is reused. Try this out on your own, it might help.

  Show server information

  Set this if you want the gateway to show the information that the server gives when the gateway logs in.

GOPHER GATEWAY

This is a caching gopher gateway, useful for sites with firewalls and for those who desire faster ”surfing speed”.

Variables

- `Location`

  The location of the Gopher gateway in the virtual filesystem.

.HTACCESS SUPPORT

Previously, this used to be part of the Roxen core. Now, to further modularise Roxen, support for `.htaccess` files has been placed in a separate module, meaning that you can turn this functionality on and off at will. Read some more on .htaccess in “.htaccess” on page 137.

Variables

- `Cache the failures`

  If this is set, all attempts to find a `.htaccess` file are logged, failures as well as successful attempts. If you run a busy site it is advantageous to set it so that problems can be quickly traced. The problem with this is that users has to press RELOAD in their browser in order to get an updated `.htaccess` file parsed.

HTTP-PROXY

This is a caching HTTP-proxy, which is useful for sites using firewalls. It’s also useful as a site-wide cache, allowing for faster ”surfing” on the Internet.

Variables

- `Location`

  The location of the proxy in the virtual filesystem. If set to any other value than the default (`http:/`), all WWW clients will fail using it as a proxy.

  The useful case that might be useful and works is `/http/`. If you set this location, a link formed like this: `<a href="/http/my.www.server/foo">mywwwserver</a>` will enable accesses to local WWW-servers through a firewall. Please consider security issues first though.

- `Logfile`

  This is the filename of the log to be used for proxy-accesses. If left empty, no logging will take place.

- `Remote proxy regular expressions`

  Here you can add redirects to remote proxy servers. If a file is requested from a host matching one of the patterns, the proxy will query the proxy server at the host and port specified. Hopefully, that proxy will then connect to the remote computer.

```
# All hosts inside *.rydnet.lysator.liu.se have to
# be accessed through lysator.liu.se
.*\.rydnet\.lysator\.liu\.se 130.236.253.11 80
```

- `External filter regular expressions`

  If the request matches one of these regular expressions, these are the external filters to use.

HTTP-RELAY

This module relays requests which the server cannot resolve to another server. This can for example be useful when you have moved information to another server.

Variables

- `Module priority`

  This tells Roxen whether or not to immediately redirect the request. If set to `last`, Roxen first tries to find the file in the ordinary way. If set to `first`, the redirection will be immediate.

- `Relay host`

  The IP number of the host to relay to.

- `Relay port`

  The port number on the remote host to relay to.

- `Always redirect`

  All URL:s that match any of the patterns in this list will always be redirected to the remote server, even if they match a pattern in the `Don't redirect` list.

  The format for the list is very straightforward:

  `pattern pattern pattern`

  where each pattern is a string wich will be matched against the requested URL. '*' denotes zero or more arbitrary character(s), and '?' is any single character.

- `Don't redirect`

  Don't relay requests for any of the patterns in this list, unless they match one of the patterns in the `Always redirect` list.

INDEX FILES

This is a directory module which you can use if you absolutely do not wish to have Roxen send directory listings. If no index file is present in the requested directory, the "No such file" message will be sent back to the client.

Variables

- `Index files`

  This is a list of file names that should be sent when a directory is requested. If a file with any of these names is present, it will be sent to the client upon request of the directory.

INDIRECT HREF

This is a database with URL aliases mapped to real URL:s. The module adds a new tag and by using this new tag (whose name is defined in the variable Tagname) you can use the symbolic names instead of the real URL. Using these definitions you only need to change in one place, here, when a URL changes.

Variables

- `Indirect HREFs`

  The database itself. The syntax is as follows:

  `name=URL`

  Example 11.2)

  `infovav=http://www.infovav.se/`

- `Tagname`

  The name of the tag used when inserting a URL from the database cf. above). `<Tagname name=indirectname>foo</Tagname>` will be replaced with `<a href=URL>foo</a>`.

ISMAP IMAGE-MAPS

This module gives Roxen the power to handle image maps. For further discussion on image maps see “Image Maps” on page 77.

There is only one variable to set, the map file extension. This is by default set to `.map` but may of course be changed to anything you deem appropriate. All files named with this extension are then parsed as map files.

LANGUAGE MODULE

This module enables you to have pages in several languages and easily manage them. It also manages nice flags that represent the languages. It enables the user to choose a preferred language and the module will automatically send pages in that language, without your having to care about separate link farms for every language.

To make the language module tick you create separate files for every language and name them as usual but with the addition of a language code at the end, e.g. index.html.en, index.html.se, index.html.fr and so on.

Links on these pages are identical in every language version. The language module sends the correct language version of requested pages if they exist. This means that the module is also a simple directory parser module.

To see an example of an older version of the language module in action visit http://www.lio.se/.

Variables

- `Default language`

  Well, if the user hasn't selected any preferred language, this is what will be used instead, of course. You did figure that out for yourself, didn't you? Also, if a file doesn't have a language extension, this is the language that Roxen will consider should be used on the page.

- `Directory index files`

  If any of these files are present in a directory, they will be returned instead of the actual directory. It is a commaseparated list of strings.

- `Flag directory`

  ° language-code.selected.gif; The image to use to indicate the language of the page. It could for example be slightly larger image.

  ° language-code.available.gif; The image that should be shown as a link to the page in the language that the image represents. It will of course only be shown if the page exists in that language.

  ° language-code.unavailable.gif; This image indicates that the page doesn't exist in the chosen language.

- `Languages`

  These are the languages that are supported by your web site. Each row defines one language's settings in this format:

  ```
  lang-code lang-name [list-of-next-lang-codes]
  ```

  (Note that the last field is optional.)

  For example:

  ```
  se Svenska en de
  en English de
  de Deutch en
  ```

  The optional list of language codes is used when there is no page present in the desired language. In the above example, if no swedish page is found, Roxen will first try to find an english page. If one doesn't exist Roxen will search for a page in german. Not until this last item has failed to appear will the default language be used. (And if that fails, well, enter the "No such resource" message!)

  The language search is done in the following order:

  ° The selected language, stored as a prestate.

  ° The user's client's accept-headers, i.e. the languages that the user has set in the preferences of his or her client.

- The selected language's `list-of-next-languages` if any exist.
- The default language. If no language has been selected and no page exists in the default language, Roxen will try the `list-of-next-languages` for the default language.
- If there were no selected language, the default language's `next-language-codes`.
- All the languages in this list, in the order they appear in the text field.

Empty lines and lines beginning with "#" or "//" will be ignored.

- `Use config (uses prestate otherwise).`

  If set, Roxen will store the user's chosen language in a client-side Cookie, if possible. Unfortunately, Netscape may not reload the page when the language is changed using Cookies, which means that users may have to manually reload to see the page in the new language. Prestates do not have this problem, but on the other hand they will not be remembered between sessions.

LOGGING DISABLER

This module can be used for turning off logging for certain resources, based on regular expressions.

Variables

- `No logging for`

  Requests for any file whose virtual file name matches this pattern will not be logged.

- `Logging for`

  Files matching this regular expression will be logged unless they also match the pattern in the `No logging for` field.

LYSATOR SPECIFIC PARSING

This is a very special module that reflects needs and also some opinions of the Lysator Computer Society. For example it can be set up to filter away the horrible `<blink>` tag from user's pages...

The module adds the tags `<icons>`, `<blink>`, and `<lysator>`. It also adds the containers `<icon>` and `<picture>`.

Look at http://www.lysator.liu.se/ to see this module working. The module is, among other things, used to create the first page where there are lots of small icons.

Note that this module might not work on your system without some tweaking of the code but you can try it and maybe learn something from working with it. The tag is mainly included as an example of how one can make very site-specific tags.

Variables

- `Icon size`

  The icons used at lysator are all squares and this number tells Roxen how big an icon should be, no matter the original size.

- `Icon has borders`

  If set, it gives borders to the icons.

- `Icon pre-url`

  This is the icon directory. It is prepended to icon URL:s.

- `Blinking enabled`

  If you set this, Roxen will not filter `<blink>` tags. This is of course only valid when this module is enabled. Roxen without the Lysator module does not filter `<blink>` tags.

MAIN RXML PARSER

The main module for parsing RXML.

Variables

- `Access log`

  If unset, the `<accessed>` tag will not work and no access log will be needed. This will save three file descriptors.

- `Access log file`

  When a file is accessed, a counter in this file is increased. This will then be used in the `<accessed>` tag.

- `Extensions to parse`

  Parse all files ending with these extensions. You can add as many as you want. The returned type will always be `text/html`.

- `Maximum file size`

  Files that is larger than this (size in kilobytes) won't be parsed, to save time.

- `SSI support: NCSA and Apache SSI support`

  If set, Roxen will parse NCSA / Apache server side includes.

- `SSI support: execute command`

  If set and if server side include support is enabled, Roxen will accept the NCSA / Apache `<!--#exec cmd=\"XXX\"-->` server side include.

  N.B.: Inserting command- and CGI script results block the server.

PIKE SCRIPT SUPPORT

This module takes care of users' Pike scripts. Scripting with Pike works somewhat like CGI, with the exception that scripts are handled internally in the server. Pike scripts are thus much faster, but it blocks the server while executing.

Note that though the module is still called µLPC script module, the scripting language is Pike really.

Variables

- `Extensions to parse`

  Files ending with any of these extensions, will be handled as Pike-scripts. It's a comma separated list.

- `Maximum evaluation time`

  The maximum execution time, in seconds, for a script. This may be changed in the script, but it is good for stopping stupid programming errors like

```
for i=0;
   while (i<=0)
      i--;
```

  This option might not be available to you.

REDIRECT MODULE V2.0

This module redirects all accesses from one path in the virtual filesystem to another server or path. This might for example be useful if you move a directory tree to another server or path.

The module allows you to redirect requests for one file to another by using regular expressions. The syntax has a three different forms; *regexp to_URL*, *prefix to_URL* or *exact file_name to_URL*.

A few examples:

Example 11.3)

Redirect requests in a certain directory to another

```
/from/.*      http://to.infovav.se/to/%f
```

Example 11.4)

Redirect all requests ending with .cgi

```
.*\.cgi       http://cgi.foo.bar/cgi-bin/%p
```

Example 11.5)

Request in /thb/ is answered by one certain file

```
/thb/.*       %u/thb_gone.html
```

Example 11.6)

Redirect requests to /roxen/ to another WWW server

```
/roxen/       http://www.roxen.com/
```

Example 11.7)

```
exact /       /main/index.html
```

%f in the to-field will be replaced with the filename of the matched file, %p will be replaced with the full path, and %u will be replaced with this server's URL, useful if you for some reason wish to send a redirect instead of doing it internally. The last example is a special case. If the first string on the line is "exact", the filename following must match *exactly*.

You can use "(" and ")" in the regular expression to separate parts of the from-pattern when using regular expressions. The parenthesised parts can then be insterted into the to-string with $1, $2 etc.

More examples:

Example 11.8)

When requesting a file ending in .class somewhere in a directory containing the partial path /SE/liu/lysator/, redirect the request to the same file, but under /java/classes/.

```
.*/SE/liu/lysator/(.*)\.class     /java/clas-
ses/SE/liu/lysator/$1.class
```

Example 11.9)

Redirect requests for files ending in .en.html to a prestate-relative URL

instead.

```
/(.*).en.html                    /(en)/$1.html
```

Example 11.10)

Make sure that all ”ugly” requests for index files get redirected to the directory itself. This will cause Roxen to always send the index file you have defined should be sent when the URL doesn’t end with a file.

```
(.*)/index.html                  %u/$1/
```

If the ”to-file” isn’t a full URL, the redirect will always be handled internally, so add `%u` to generate an actual redirect, i.e. to return a new URL to the browser.

N.B.: For reasons of speed; if the *from* pattern does not contain any ”*” characters, it will not be treated like a regular expression, but as a prefix that must match exactly.

SECURE FILESYSTEM MODULE

The secure file system module works like the ordinary filesystem module. It is a bit more secure since it allows regular expression security. There is only one additional variable in this module. For a description of all the other variables that you can set refer to “FileSystem” on page 119.

Variables

- `Security patterns`

  This is a list with entries on the form `filepattern: security level=value`. Each security level can be one or more from this list:

  - `allow ip=pattern`
  - `deny ip=pattern`
  - `allow user=pattern`

  In patterns: * is one or more characters, ? is one character.

  Please note the the expressions are tested from top and downwards, so if you have `*: allow host = *` as the first line, it won’t matter whatever you add further down, everything will still be allowed.

STATUS MONITOR

The status monitor simply shows the number of open connections and how much data there is to send. Note that this module only works with Netscape and probably the Microsoft Internet Explorer because it uses

dynamic document loading.

Variables

- `Mountpoint`

  The file to access to get the status information.

- `Maximum bar length`

  The maximum length of the bar graph bars (in pixels).

- `Minimum scale factor`

  The minimum scaling factor to use. The actual maximum value of the graph will be at least (`minfact * maxlength`).

- `Scale step`

  The scaling factor will increment with 'Scale step' each time the max value has gone off the scale. The actual algorithm for the scaling factor is:

```
float factor, tmp_factor;
factor = minfact;

while((value/maxlen) > factor)
   factor *= scalestep;

barlength = value / factor;
```

- `Delay between updates`

  The delay between each update of the information (in seconds).

TABLIFY

This is parser module that can generate HTML2.0 tables from, for example, a set of tab separated fields. It defines a tag; `<tablify>` and there are no variables to set. The tag is a container, i.e. text between `<tablify>...<tablify>` is parsed and put into a table.

TAB LIST

This module is used to automatically generate tab lists, like the one on the top of the Roxen configuration interface.

Variables

- `Mount point`

  This is the place where the module resides in the virtual file system.

- `Font path`

  Is where the fonts reside on your system. This path is relative the mount point.

- `Default font`

  The font that should be used when the module generates the tablists.

TIMESTAMP

A sample extension type module. If you open a file with the extension .timestamp, the time stamp, the last modification date (`mtime`) of the file without that extension will be shown.

Variables

- `Mount Point`

  The location in Roxens virtual file system.

- `Time Modification`

  Time difference in seconds from the system clock.

USER DATABASE AND SECURITY

The user database and security module manages the security in Roxen. It uses the normal system password and user database to validate users. The module also maintains the user database for all other modules in Roxen, e.g. the user filesystem module.

Variables

- `Password database request method`

  What method to use to maintain the passwd database. `getpwent` is very slow but it should work on all systems and it will work with `/etc/shadow`, that is if Roxen is allowed to read it. It will also enable automatic password information updates. Every ten seconds the information about one user from the passwd database will be updated. A call will also be performed if a user is not in the in-memory copy of the passwd database. This choice may not be available on your system.

  Other methods are `ypcat`, `niscat` (on Solaris 2.x systems) and `file`. If you choose `none`, all authentication requests will succeed regardless of use name and password.

- `Password database file`

  The password file that is used for authentication checks if the method is set to `file`.

- `Password command arguments`

  If you wish to send extra arguments to either `ypcat` or `niscat`. For `ypcat` the full command line will be;

  ```
  ypcat »arguments« passwd
  ```

  and for `niscat;`

  ```
  niscat »arguments« passwd.org_dir
  ```

  If you do not want the `passwd` part, you can end your arguments with "#".

- `Turn }{| into ääö.`

  If set, }{| will become ääö in the Real Name field of the userinfo database. This is quite useful in Sweden.

- `Strip finger information from fullname`

  If enabled, this will strip everything after the first "," character from the GECOS field of the user database.

USER FILESYSTEM

The user file systems works more or less like a file system, with the exception that it uses the user database to get information about the home directories of users. This is then used to fetch the files by appending a public directory path.

This means that you have to have the user database enabled or this module won't work as expected.

Variables

- `Banish list`

  None of the users in this comma separated list is considered valid. This can be used to selectively shut off access for certain users, or to disable stupid loops, like if the home dir of the user `www` is `/usr/www/`, and most html files are located in the `html/` directory, which also happens to be the public directory. This would make `/~www/` the same as `/`.

- `Password users only`

  Only users who have a valid password on the system are allowed to have public directories.

- `Public directory`

  This is where the public directory is located. If the directory is set to .public, the module has the mountpoint /~, and the file /~per/foo is accessed, and the home directory of Per is /home/per, the module will try to find the file or directory /home/per/.public/foo.

- `Only owned files`

  If set, only files whom the user really owns can be sent. This enhances security, but may be a pain when several users are working in one user's directory on a project.

All variables except `Search path` are inherited from the file system module. Refer to "FileSystem" on page 119 for the other variables.

USER LOGGER

This is a module that logs the accesses for each user in their home directories if, and only if, they create a file named Accesslog in that directory and also set write permissions on this file so that Roxen may peruse it. This can save time for logging, especially when there is a large amount of users on your system.

Variables

- `Maximum number of open log files`

  Since any one user's pages are typically accessed several times in a row, it is inefficient to close the files after every logging. This number tells Roxen how many user's log files should be allowed to be open at the same time.

- `Log file garb timeout`

  This should be set to an integer number and is the number of seconds after which the file should be closed.

- `Only log in user log`

  If this one is set, no logging in the normal logs will be done.

- `Private logs`

  These directories want their own log files. Use either a specific path, or a pattern. `/foo/` will check if there is a `/foo/AccessLog`, while `/users/%s/` will check for the file `AccessLog` in all subdirectories of `/users/`. All paths are in the virtual filesystem, not the real one.

WAIS GATEWAY

This module allows Roxen to act as a caching WAIS Gateway. It can be useful for sites with firewall installations and as well as for everyone who wishes to experience faster ”surfing”.

Variables

- `Location`

  The mountpoint of the gateway in the virtual filesystem.

- `Cache wais files`

  Enables the caching of wais files

- `Connection refused message`

  Set this variable to the path of the file that should be sent to the user upon a ”connection refused” error.

SECURE SOCKETS LAYER, SSL

Using SSL is a quite delicate operation. To use SSL you have to obtain a verification certificate and an SSLeay implementation. The latter is included with Roxen, so we urge you to read that documentation before configuring Roxen to use an SSL port. Note that the we do not provide any printed documentation for this, we only distribute it in its electronical form.

To obtain the certificate, you have to contact a verification institute, for example Verisign.

Follow these steps to set up Roxen-SSL on your machine;

1. *Install the SSLeay package.*

   This must be done before you start installing Roxen or it won't work. Follow the instructions in the SSLeay documentation and leave the default paths alone. This is because Roxen expects the files to be in these default locations. If you start with a clean, new Roxen this is done during start-up and compilation.

2. *Create a certificate and a private key.*

   This can be done either by contacting one of the verification institutes or by temporarily creating a dummy certificate. A dummy certificate is a valid certificate but it's worthless since it hasn't been verified by the proper authorities. It is necessary to have something with which to start though.

   Follow the instructions for creating certificates and private keys in the SSLeay documentation.

3. *Install and start up Roxen.*

   Configure port(s) for one or more virtual servers to use SSL. Use the instructions in *Listen Ports...* under the node *Server Variables*.

.HTACCESS

If you use .htaccess you will place a special file in every directory detailing the access rights for that directory. The name of the file should be `.htaccess`. .htaccess was originally implemented by NCSA for use with their WWW server, NCSA Httpd.

SECURE TRANSMISSION

.htaccess files reside in directories that are open for reading but since the filename begins with a dot, Roxen considers it to be a hidden file and thus doesn't send it, at least as long as you configure the directory parsing module in the right way.

The transmission of login name and user id is uuencoded and according to NCSA is roughly as secure as your average telnet connection.

HOW TO RESTRICT ACCESS

Create a file named .htaccess in the directory that you wish to protect.

The file should look something like this;

```
AuthUserFile /fullpath/.htpasswd
AuthGroupFile /fullpath/.htgroup
AuthName ByPassword
AuthType Basic

<Limit GET PUT POST>
require user »username«
</Limit>
```

The first four lines contain the definitions of how we should protect and the last three tell us what to check when someone tries to request the protected resources.

`AuthUserFile` and `AuthGroupFile` indicate where the password file and the group file are respectively. They must both be fully qualified Unix paths. If the group file doesn't exist put `/dev/null` there instead to sigify that it doesn't exist. You can place these files anywhere in the real filesystem.

`AuthName` can be anything you want. It gives the *Realm*[1] for which the protection is provided. This name is usually given when a browser prompts for a password, cf. "<RETURN>" on page 64. Browsers save this information along with the URL so that passwords entered for a certain realm are used when that realm once again demands authentication. If you don't set this to something it will default to `ByPassword`.

`AuthType` should be set to Basic, since we are using Basic HTTP Authentication.

In the limiting section we can set up protection against all or a few of the available methods for information transfer. For normal HTTP traffic the method GET is used. Other methods include PUT and DELETE.

What about the password file then? It is very easy to create:

```
htpasswd -c /fullpath/.htpasswd username
```

You will then be asked to enter a password for that user twice and that's it. Now look in the newly created file `.htpasswd` and you will see

```
username:encryptedpassword
```

Adding more user/password pairs is easy; just do:

1. A realm is a group of files and directories that are of the same security level.

```
htpasswd /fullpath/.htpasswd otherusername
```

once for every new user you wish to add. Do not include `-c` since it clears the file (unless you wish to do so of course).

When many users are allowed different access rights in different parts of the WWW tree it can be useful to group users together. These groups are set up in the file `.htgroup` which you create in a proper place, looking something like this:

```
GenerationX: spike shaq martin rod
```

You can of course add as many groups as you like to this file. Now anyone in a group can use his or her individual username and password to get the same access rights as the others when entering a group limited realm.

Read all about .htaccess at http://hoohoo.ncsa.uiuc.edu/docs/tutorials/user.html.

PROGRAMMER'S REFERENCE

Detta dokument har framställts med FrameMaker 4.0.4

CHAPTER 12

THE PIKE QUICK GUIDE

INTRODUCTION

This is an extremely brief introduction to Pike, Roxen's native language. It is not a programmer's handbook as such, as that would take up far too much space, but a reference for the budding Roxen enhancer.

This is not an introduction to programming, we assume that you know how to program. Later we will publish a nice Pike manual, with tutorials and stuff.

Pike is an object-oriented language and it has a quite C-like syntax, and many of you C / C++ programmers will feel right at home. However, do look out for the differences!

Pike is not a compiled language, it is interpreted, a bit like Perl. Like Perl it has very good string handling capabilities and quite not like Perl it is a more orthogonal language, i.e. you cannot write cryptic programs as easily as in Perl.

PRINTING TEXT

Let's begin by writing a small Pike program:

```
int main()
{
   write("hello world\n");
   return 0;
}
```

Let's call this file **hello_world.pike**, and then we try to run it:

```
$ pike hello_world.pike
hello world
$
```

Pretty simple, eh? Now let's see what everything means:

```
int main()
```

`main()` is a function identifier. Before the function name we have placed the declaration of the type of value it returns, in this case `int` which is the integer type in Pike. The empty space between the parenthesises indicates that this function takes no arguments.

A Pike program has to contain at least one function, the `main()` function. This function is where program execution starts and thus the function from which every other function is called, directly or indirectly. We can say that this function is called by the operating system.

Pike is, as many other programming languages, built upon the concept of functions, i.e. what the program does is separated into small portions, or functions, each performing one (perhaps very complex) task. A function declaration consists of certain essential components; the *type* of the value it will return, the *name* of the function, the *parameters*, if any, it takes and the body of the function. A function is also a part of something greater; an object. You can program in Pike without caring about objects, but the programs you write will in fact be objects themselves anyway.

Now let's examine the body of `main()`;

```
{
   write("hello world\n");
   return 0;
}
```

Within the function body, programming instructions, statements, are grouped together in blocks. A block is as series of statements placed between curly brackets. Every statement has to end in a semicolon.

```
write("hello world\n");
```

The first statement is a call to the builtin function `write()`. This will execute the code in the function `write()` with the arguments as input da-

ta. In this case, the constant string `hello world\n` is sent. Well, not quite. The "\n" combination corresponds to the newline character.

`write()` then writes this string to `stdout` when executed. `Stdout` is the standard Unix output channel, usually the screen.

```
return 0;
```

This statement exits the function and returns the value zero. Any statements following the return statements will not be executed.

IMPROVING OUR PROGRAM

Typing `pike hello_world.pike` to run our program may seem a bit unpractical. Fortunately, Unix provides us with a way of automating this somewhat. If we modify hello_world.pike to look like this:

```
#!/usr/local/bin/pike

int main()
{
   write("hello world\n");
}
```

And then we tell Unix that hello_world.pike is executable:

```
$ chmod +x hello_world.pike
```

Now we can run hello_world.pike without having to bother with running pike;

```
$ ./hello_world.pike
hello world
$
```

N.B.: The hash bang (#!) must be first in the file, not even whitespace is allowed to precede it! The file after the hash bang must also be the complete filename to the Pike binary, and it may not exceed 30 characters.

CHOICES AREN'T HARD TO MAKE

FURTHER IMPROVEMENTS

Now, wouldn't it be nice if it said "Hello world!" instead of "hello world" ? But of course we don't want to make our program "incompatible" with the old version. Someone might need the program to work

like it used to. Therefore we'll add a *command line option* that will make it print the old "hello world". We have to give the program the ability to choose what it should output based on the command line option. This is what it could look like;

```
#!/usr/local/bin/pike

int main(int argc, array (string) argv)
{
   if(argc > 1 && argv[1]=="--traditional")
   {
     write("hello world\n"); // old stype
   }
   else
   {
     write("Hello world!\n"); // new style
   }
   return 0;
}
```

Let's run it;

```
$ chmod +x hello_world.pike
$ ./hello_world.pike
Hello world!
$ ./hello_world.pike --traditional
hello world
$
```

What is new in this version, then?

```
int main(int argc, string *argv)
```

In this version, the space between the parenthesises has been filled. What it means is that `main()` now takes two arguments. One is called `argc`, and is type `int`. The other is called `argv` and is an array of strings. This could also be represented as `string *argv`, but that is harder to read and often confuses programmers used to other languages. The asterisk means that the argument is an array, in this case an array of strings.

The arguments to `main()` are taken from the command line when the Pike program is executed. The first argument, `argc`, is how many words were written on the command line (including the command itself) and `argv` is an array formed by these words.

```
if(argc > 1 && argv[1] == "--traditional")
{
   write("hello world\n"); // old stype
}
else
{
   write("Hello world!\n"); // new style
}
```

This is an if-else statement that will execute what's between the first set of brackets if the expression between the parentheses evaluates to `true`, i.e. not zero. Otherwise the block after `else` will be executed. Let's look at that expression;

```
argc > 1 && argv[1] == "--traditional"
```

Loosely translated, this means: `argc` is greater than one, i.e. there was something in addition to the program invocation on the command line, and the second element in the array `argv` is equal to the string `--traditional`.

Also note the comments:

```
write("hello world\n"); // old stype
```

The // begins a comment which continues to the end of the line. Comments lets you type in text in the program which will be ignored by the computer. This is to inform whoever might read your code (like yourself) of what the program does to make it easier to understand. Comments are also allowed to look like C-style comments, i.e. /*....*/, which can extend over several lines. The // comment only extends to the end of the line.

DATA TYPES

As you remember from the first examples we have to indicate the type of value returned by a functions or contained in a variable. We used integers (int), strings (string), and arrays (with the * notation). The others are `mapping`, `mixed`, `void`, `float`, `multiset`, `function`, `object`, `program`.

Neither `mixed` nor `void` are really types, `void` signifies that no value should be returned and `mixed` that the return value can be of any type, or that the variable can contain any type of value.

Function, object and program are all types related to object orientation. We will not discuss the last three in any great detail in this short Pike overview. You can, however, read more at http://pike.infovav.se/. There you'll also find more details on the other data types and the possible operations that are possible to perform on each of them.

INT

The integer type stores an integer.

FLOAT

This variable type stores a floating point number.

ARRAY

Arrays are basically a place to store a number of other values. Arrays in Pike are allocated blocks of values. They are dynamically allocated and do not need to be declared as in C. The values in the array can be set when creating the array like this,

```
arr=({1,2,3});
```

or anytime afterwards like this,

```
arr[index]=data;
```

where `index` is an integer, i.e. entry number `index` is set to `data`. The first index of an array is 0 (zero). A negative index will count from the end of the array rather than from the beginning, -1 being the last element.

Note that arrays are shared and use reference counts to keep track of their references. This will have the effect that you can have two variables pointing at the same array, and when you change an index in it, both variables will reflect the change.

To indicate an array of a certain type of value you can use the * operator, e.g.

```
string *i;
```

which tells us that i is an array of strings. The * binds to the variable name, not to the type, so writing

```
string *i, j;
```

will declare one array of strings and one string. However, it's much clea-

rer to write

```
array (string) i;
```

STRING

A string contains a sequence of characters, a text, i.e. a word, a sentence, or a book. Note that this is not simply the letters A to Z; special characters, null characters, newlines and so on can all be stored in a string. Any 8-bit character is allowed. String is a basic type in Pike, as opposed to C where strings are represented by an array of char. This means that you cannot assign new values to individual characters in a string.

Also, all strings are ”shared”, i.e. if the same string is used in several places, only one will be stored in memory.

When writing a string in a program, you enclose it in doublequotes. To write special characters you need to use the following syntax;

- `\n newline`
- `\r carriage return`
- `\t tab`
- `\b backspace`
- `\" " (quotation character)`
- `\\ \ (literal backslash)`

MAPPING

A mapping is basically an array that can be indexed on any type, not just integers. It can also be seen as a way of linking data (usaully strings) together. It consists of a lot of index-data pairs which are linked togeter in such a way that `map[index1]` returns `data1`.

A mapping can be created in a way similar to arrays;

```
map=([five:good,ten:excellent]);
```

You can also set that data by writing `map[five]=good`.

If you try to set an index in a mapping that isn't already present in the mapping it will be added as well.

MULTISET

A multiset is basically a mapping without data values. When referring to

an index of the multiset a 1 (one) will be returned if the index is present, 0 (zero) otherwise.

A MORE ELABORATE EXAMPLE

To illustrate several of the fundamental points of Pike we will now introduce an example program, that will be extended as we go. We will build a database program that keeps track of a record collection and the songs on the records.

In the first version we hard-code our ”database” into the program. The database is a mapping where the index is the record name and the data is an array of strings. The strings are of course the song names. The default register consists of one record.

```
#!/usr/local/bin/pike

mapping (string:array(string)) records =
([
   ”Star Wars Trilogy” :
   ({”Fox Fanfare”,
   ”Main Title”,
   ”Princess leia’s Theme”,
   ”Here They Come”,
   ”The Asteriod Field”,
   ”Yoda’s Theme”,
   ”The Imperial March”,
   ”Parade of th Ewoks”,
   ”Luke and Leia”,
   ”Fight with Tie Fighters”,
   ”Jabba the Hut”,
   ”Darth Vader’s Death”,
   ”The Forest Battle”,
   ”Finale”})
]);
```

We want to be able to get a simple list of the records in our database. The function `list_records` just goes through the mapping `records` and puts the indices, i.e. the record names, in an array of strings, `record_names`. By using the builtin function `sort` we put the record names into the array in alphabetical order which might be a nice touch.

For the printout we just print a header, ”Records:”, followed by a newline. Then we use the loop control structure `for` to traverse the array and print every item in it, including the number of the record, by counting up from zero to the last item of the array. The builtin function `sizeof` gives the number of items in an array. The printout is formatted through the use of `sprintf` which works more or less like the C function of the same name.

```
void list_records()
{
   int i;
   array (string) record_names=sort(indices
     (records));

   write(”Records:\n”);
   for(i=0;i<sizeof(record_names);i++)
     write(sprintf(”%3d: %s\n”, i+1,
          record_names[i]));
}
```

If the command line contained a number our program will find the record of that number and print its name along with the songs of this record. First we create the same array of record names as in the previous function, then we find the name of the record whose number (`num`) we gave as an argument to this function. Next we put the songs of this record in the array `songs` and print the record name followed by the songs, each song on a separate line.

```
void show_record(int num)
{
   int i;
   array (string) record_names =
     sort(indices (records));
   string name=record_names[num-1];
   array (string) songs=records[name];

   write(sprintf(”Record %d, %s\n”,num,name));
   for(i=0;i<sizeof(songs);i++)
     write(sprintf(”%3d: %s\n”, i+1, songs[i]));
}
```

The main function doesn’t do much; it checks whether there was anything on the command line after the invocation. If this is not the case it calls the `list_records` function, otherwise it sends the given argument to the `show_record` function. When either one of those functions is done

the program just quits.

```
int main(int argc, array (string) argv)
{
   if(argc <= 1)
   {
      list_records();
   } else {
      show_record((int) argv[1]);
   }
}
```

TAKING CARE OF INPUT

Now, it would be better and more general if we could enter more records into our database. Let's add such a function and modify the `main()` function to accept "commands".

ADDRECORD()

Using the builtin function `readline()` we wait for input which will be put into the variable `record_name`. The argument to `readline()` is printed as a prompt in front of the user's input. Readline takes everything up to a newline character.

Now we use the control structure `while` to check whether we should continue inputting songs. The `while(1)` can be interpreted as "while everything is ok". When something has been read into the variable `song` it is checked. If it is a "." we return a null value that will be used in the while statement to indicate that it is not ok to continue asking for song names. If it is not a dot, the string will be added to the array of songs for this record, unless it's an empty string.

Note the "+=" operator. It is the same as saying `records[record_name]=records[record_name]+({song})`.

```
void add_record()
{
   string record_name=readline("Record name: ");
   records[record_name]=({});
   write("Input song names, one per line. End with
   '.' on it's own line.\n");
   while(1)
   {
      string song;
      song=readline(sprintf("Song %2d: ",
           sizeof(records[record_name])+1));
      if(song==".")
        return;
      if (strlen(song))
        records[record_name]+=({song});
   }
}
```

MAIN()

The main function now does not care about any command line arguments. Instead we use `readline()` to prompt the user for instructions and arguments. The available instructions are "add", "list" and "quit". What you enter into the variables `cmd` and `args` is checked in the `switch()` block. If you enter something that is not covered in any of the `case` statements the program just silently ignores it and asks for a new command.

In a `switch()` the argument (in this case `cmd`) is checked in the `case` statements. The first `case` where the expression equals `cmd` (the argument) then executes the statement after the colon. If no expression is equal, we just fall through without any action.

The only command that takes an argument is "list" which works like the first version of the program; if there is an argument that record is shown along with its songs, and if there isn't the program sends a list of the records in the database. When the program returns from either of the listing functions, the `break` instruction tells the program to jump out of the `switch()` block.

"Add" of course turns control over to the function descibed above.

If the command given is "quit" the `exit(0)` statement stops the execution of the program and returns 0 (zero) to the operating systems, telling

it that everything was ok.

```
int main(int argc, string * argv)
{
   string cmd;
   while(cmd=readline("Command: "))
   {
      string args;
      sscanf(cmd,"%s %s",cmd,args);

      switch(cmd)
      {
        case "list":
        if((int)args)
        {
          show_record((int)args);
        } else {
          list_records();
        }
        break;

        case "quit":exit(0);

        case "add":add_record();
        break;
      }
   }
}
```

COMMUNICATING WITH FILES

Now if we want to save the database and also be able to retrieve previously stored data we have to communicate with the environment, i.e. with files on disk.

Now we have to introduce objects. To open a file, be it for writing or for reading, we need to use the builtin program /precompiled/file/. A program is an object. An object can be *cloned*, i.e. we can create an object just like it and associate it with the physical file in question. The methods and variables in the file object enables us to perform actions on the associated file. You can find more on this at http://pike.infovav.se/. The methods we need to use are open, read, write and close.

SAVE()

First we clone the program /precompiled/file/ to the object `o`. Then we use it to open the file named `file` for writing, using the fact that if there's an error during opening, `o` will return a false value which we can detect and act upon by exiting. The arrow operator is what you use to access methods and variables in an object.

Note that it is perhaps easier to understand if we write

```
o=new(File); // Roxen addition
```

instead of using the somewhat bulky `clone()` expression. `File` is a constant that can be used almost anywhere in cases like this.

N.B.: when inheriting you must use /precompiled/file.

If there's no error we use yet another control structure, `foreach`, to go through the mapping `records` one record at a time. We precede record names with the string "Record: " and song names with "Song: ". We also put every entry, be it song or record, on its own line by adding a newline to everything we write to the file.

Finally, remember to close the file.

```
void save(string file)
{
   string name, song;
   object o;
   o=clone((program)"/precompiled/file");

   if(!o->open(file,"wct"))
   {
     write("Failed to open file.\n");
     return;
   }

   foreach(indices(records),name)
   {
     o->write("Record: "+name+"\n");
     foreach(records[name],song)
       o->write("Song: "+song+"\n");
   }

   o->close();
}
```

LOAD()

The `load` function begins much the same, except we open the file named `file` for reading instead. When receiving data from the file we put it in the string `file_contents`. The somewhat cryptic argument given to the method `o->read` means that the reading should not end until the end of the file.

After having closed the file we initialise our database, i.e. the mapping `records`. Then we have to put `file_contents` into the mapping and we do this by splitting the string on newlines (cf. the `split` operator in Perl) using the division operator. Yes, that's right: by dividing one string with another we can obtain an array consisting of parts from the first. And by using a `foreach` statement we can take the string `file_contents` apart piece by piece, putting each piece back in its proper place in the mapping `records`.

```
void load(string file)
{
   object o;
   string name="ERROR";
   string file_contents,line;

   o=clone((program)"/precompiled/file");
   //or o=new(File);
   if(!o->open(file,"r"))
   {
     write("Failed to open file.\n");
     return;
   }

   file_contents=o->read(0x7fffffff);
   o->close();

   records=([]);

   foreach(file_contents/"\n",line)
   {
     string cmd, arg;
     if(sscanf(line,"%s: %s",cmd,arg))
     {
       switch(lower_case(cmd))
       {
         case "record":
         name=arg;
         records[name]=({});
         break;

         case "song":
         records[name]+=({arg});
         break;
       }
     }
   }
}
```

MAIN()

`main()` remains almost unchanged, except for the addition of two `case` statements with which we now can call the load and save functions. Note

that you must provide a filename to load and save, respectively, otherwise they will return an error which will crash the program.

```
case "save":save(args);
break;

case "load":load(args);
break;
```

COMPLETING THE PROGRAM

DELETE()

If you sell one of your records it might be nice to able to delete that entry from the database. The delete function is quite simple; first we set up an array of record names (cf. the `list_records` function). Then we find the name of the record of the number `num` and use the builtin function `m_delete()` to remove that entry from `records`.

```
void delete_record(int num)
{
   string *record_names=sort(indices(records));
   string name=record_names[num-1];

   m_delete(records,name);
}
```

SEARCH()

Searching for songs is quite easy too. To count the number of hits we declare the variable `hits`. Note that it's not necessary to initialise variables, that is done automatically when the variable is declared if you do not do it explicitly. To be able to use the builtin function `search()`, which searches for the presence of a given string inside another, we put the search string in lowercase and compare it with the lowercase version of every song. The use of `search()` enables us to search for partial song titles as well.

When a match is found it is immediately written to standard output with the record name followed by the name of the song where the search string was found and a newline.

If there were no hits at all, the function prints out a message saying just that.

```
void find_song(string title)
{
   string name, song;
   int hits;

   title=lower_case(title);

   foreach(indices(records),name)
   {
      foreach(records[name],song)
      {
         if(search(lower_case(song), title) != -1)
         {
            write(name+"; "+song+"\n");
            hits++;
         }
      }
   }

   if(!hits) write("Not found.\n");
}
```

MAIN()

Once again `main()` is left unchanged, except for yet another two `case` statements used to call the search and delete functions, respectively. Note that you must provide an argument to `delete` or it will not work properly.

```
case "delete":delete_record((int)args);
break;

case "search":find_song(args);
break;
```

FINAL NOTES

Well that's it! The example is now a complete working example of a Pike program. But of course there are plenty of details that we haven't attended to. Error checking is for example extremely sparse in our program. This is the next step and it is left as an execise to the student. Good luck! And once again: read the on-line documentation at http://pike.infovav.se/ for all the technical reference you can stomach!

By the way; the complete listing can be found in APPENDIX B on page 191. Read it, study it and enjoy!

CHAPTER 13

HOW TO MAKE YOUR OWN ROXEN MODULE

In order to really take advantage of Roxen as a flexible management information engine, we provide you with an easy way of programming new features into Roxen - and you add them easily while the server is running!

This text assumes that you have a working knowledge of programming in general and of Pike in particular. In "The Pike Quick Guide" on page 143 you can find more information on Pike. You can also check out the web pages at http://pike.infovav.se/. Good luck!

MODULE TYPES

The three most common types of modules are *file extension*, *location* and *parser* modules.

- A location module handles everything under a certain directory in the virtual filesystem, for instance /cgi-bin/. It is usually this type of module that finds files and passes them on to the different extension modules.
- A file extension module handles files of one or several extension types, e.g. .cgi, after that a location module has located the file.

Detta dokument har framställts med FrameMaker 4.0.4

- A parser module defines one or more RXML tags. These are later handled by a module of the type *main parser*. There can only be one *main parser* in each virtual server, but there can be any number of *parser* modules. The main parser gets all modules of type parser sent to it, but nothing else. To be able to do any parsing, a main parser module also has to be an extension module or a location module, or at least an *extension without a file* module, see below.

There are also several module types which are used less frequently;

- *Authentication*; A module type that handles authentification of users, and also keeps a database of users for for instance the *user filesystem* module (which is an example of a location module).
- *Directory*; This type of module handles file listings. If you don't have one such module, Roxen won't be able to generate file listings of directories, and the automatic use of index files, like `index.html`, won't work. There can only be one instance of this type of module present in each virtual server.
- *Extension*; A kind of extension module that is called *before* the location module is called, and thus it does not receive any file pointers. Note the difference; this is not a *file extension* module.
- *First try*; A module type that is called before all other module types, except other modules of the type first try and the Authentication type.
- *Last try*; A module that is called after all other modules, if none of the other modules found anything to do.
- *Filter*; This type of module does something after all the other modules have had their way with a file. It differs from last try modules in that it is run even if other modules managed to resolve a request. It is also the only module type to talk to Last try modules.
- *Types*; This module type quite simply handles extension to content-type mapping in those cases where the modules that have already run haven't told the system what content-type the file has. For example, the file `things.html` should have content-type `text/html`.
- *URL*; This type of module receives a URL and returns another. This module type should be used if you wish to implement redirecting modules of any kind.

- *Logger*; A module type that should be used for implementing logging modules. Logger modules are called at the same time as the response is sent out.

A module can have one or more module types, cf. the main parser module type above. Modules can even lack type entirely. This is quite pointless, though, since such a module would never be called by Roxen.

So, in a Roxen context, how are these modules related to each other? We have made a nice little flowchart in figure 13.1 on page 164 that shows how the modules are called during the treatment of a request. In written terms it goes something like in the following section. Do you need to know this then in order to program Roxen? No, actually not, since if you know what kind of module you intend to make and register it properly (see the beginning of "How to write a module" on page 165), Roxen will call it when it's time and take care of the output it generates. But it might give you some insight!

ROXEN MODULE TYPE CALLING SEQUENCE

If there is no module at a certain level, or if the modules there are cannot handle the request, it is sent on to the the next module level.

When a request first comes in it is sent through the protocol modules to find out what protocol is used, and to parse the request accordingly.

Requests are then authenticated. Authentication modules basically just set a flag and sends the request on its way to be handled.

It is then sent through the first try modules if there are any. If the request is successfully handled by a first try module now sent through the filter module level.

If the request was not successfully handled by a first try module it is sent on to the URL module level. A successful result will send the resulting, rewritten request back to the beginning of the URL level, i.e. Roxen will send it through all URL modules again. When there's no possible (further) treatment the request is sent on to the extension module level.

Extension modules handle "imaginary" files. It means that it looks att the ending of the request and based on this it might do something about the request. Since no location module has yet been run, there is no file associated with the request, and thus no file will be involved in the response. The only example that we have of this is the Timestamp module, see "Timestamp" on page 133. There is also an as yet unsupported module that

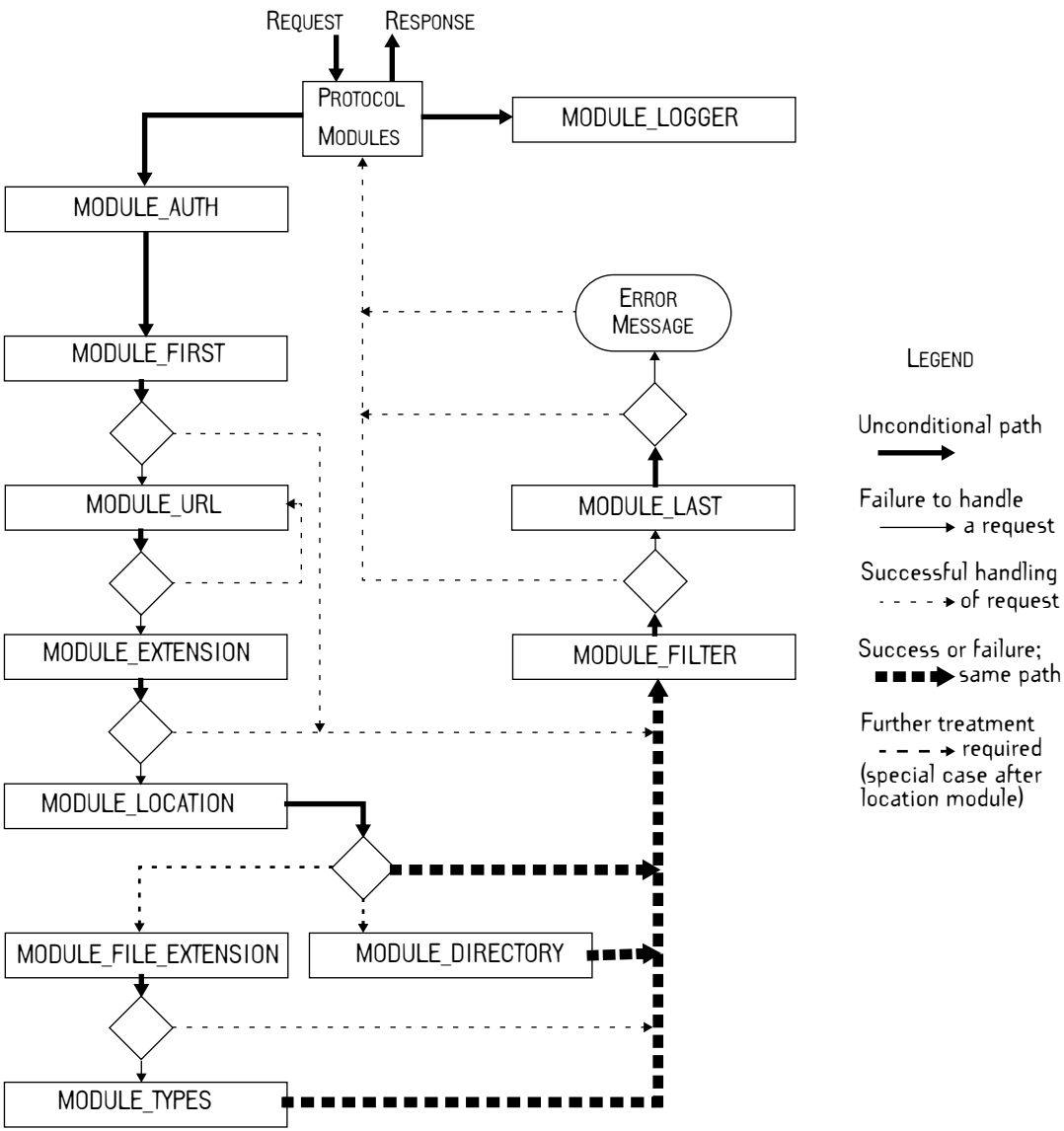

FIGURE 13.1 *The calling sequence of the module types of Roxen. At each type level, all modules of that type are called in order of priority.*

can take care of client side image maps and remake them into server side image maps by creating an imaginary map file on the fly in case the client does not handle client side image maps. Success here sends the request on to the filter modules.

Next contestant is the location module level where Roxen finally tries to find something in the filesystem. There are four possible results here;

- Nothing could be found.
- There was actually a location module that could take care of the request and send something back.
- A directory was found.
- A file was found.

In the first two cases the request is sent on to the filter level. The third reults in a try at the directory module level. You probably also guessed that if a file was found it is sent on to the file extension modules.

Whether there is either failure or success at the directory modules it's time to try the filter modules.

Success at the file extension modules also sends the request on to the filter modules and failure sends it the content types modules. Content type modules sends the reult to the filter modules.

Success at a filter module sends the result back to the protocol modules. If there is a failure Roxen will try the last try modules. If there aren't any last try modules or there is a failure here too, an error message will be the result.

Now the result ends up with the proper protocol module which logs the request through the use of a logger module and also sends the result back to the client.

Is everybody still awake? Good, let's go on with more practical details!

HOW TO WRITE A MODULE

First of all, configure a test server. This way you will have your own copy of Roxen with which you can play.

One variable that is useful to change is *Module directory* under GLOBAL VARIABLES, which tells the server where to look for modules. This is a

comma separated list of directories. The value for this could, for example, be: local/modules/, /home/john/roxen_modules/. They can be relative to Roxen's server/ directory.

Now you're all set to write your own module. Below is a step-by-step instruction on how to accomplish this. We assume that you have basic knowledge of Pike programming.

Many times we do not tell you everything. If you wonder about something, read the source. That will help you learn too.

THE FUNDAMENTALS OF A MODULE

We start by introducing a module skeleton, containing all necessary functions and inheritances for a module.

```
#include <module.h>
inherit "module";
inherit "roxenlib";
array register_module()
{
   return ({
     MODULE_TYPE,
       "module name",
       "module documentation",
       0,
       0|1,
   });
}
```

First the file module.h is included. It contains a lot of constant definitions, e.g. all module types. It also contains some macros that can be used when you want the value of a certain module variable.

After that, the file module is inherited. In this file, functions like defvar(), set() and query() are defined. It also contains several usable default values and values to use for checks in your modules.

The file roxenlib includes a lot of useful functions. Though it is not necessary to include it, it can be very useful. For ideas on how to use this file, please take look at "Returning values" on page 176.

The function register_module() is supposed to return an array that defines the function of the module, the name of the module and if it can have more than one concurrent active instance in each virtual server.

The first element of the array is the module type. This is a bitwise-or (|)

of module types. Ordinarily each module only has one type, but for instance htmlparse.pike in the Roxen distrubution is a combination of a `MODULE_MAIN_PARSER`, a `MODULE_PARSER` and a `MODULE_FILE_EXTENSION`.

MODULE TYPE NAME	MODULE TYPE
MODULE_AUTH	Authentication module and user data base. There can only be one active instance at a time.
MODULE_DIRECTORIES	Directory parsing module. Only one active at any one time.
MODULE_EXTENSION	Extension module, called before location modules.
MODULE_FILE_EXTENSION	Extension module called after location modules.
MODULE_FILTER	Used for ”post-treatment” of files.
MODULE_FIRST	First try modules.
MODULE_LAST	Last resort modules.
MODULE_LOCATION	Location module, finds files.
MODULE_LOGGER	Logger module, used when implementing different logging schemes.
MODULE_MAIN_PARSER	Main parser. Only one active at a time.
MODULE_PARSER	Normal parser, handles one or more tags.
MODULE_TYPES	Type module that handles extension to content type mappings.
MODULE_URL	URL modules can internally rebuild requests.

TABLE 13.1 *Module type identifiers and their meanings.*

The second element is the module name, and the third element is a short description of what the module does. The fourth element is reserved for future use. The fifth element is ”0” if there can be multiple copies of the module in each virtual server (e.g. the filesystem module) or ”1” if there can only be one active copy in each virtual server (e.g. the Main RXML Parser).

Since this example is a module which adds tags to RXML we choose the following look of the `register_module()` function:

```
array register_module()
{
   return({
     MODULE_PARSER,
        "Gazonk",
        "The gazonk module. Adds a container and "
        "a non-container: "
        "&lt;foo&gt;...&lt;/foo&gt; and "
        "&lt;bar&gt;",
        0,
        1,
   });
}
```

Now the module can register itself, but that's not quite enough, how do we get it to really handle the tags `<foo>...</foo>` and `<bar>`?

CALLBACK FUNCTIONS

This brings us to the module specific callback functions, which follow. See table 13.2 on page 169 for a compact listing. In several of them you can see the argument `object request_id`. This object is described in the section on Pike scripts, see "Pike scripts" on page 85.

- `void create();`

  Called automatically by Pike when the object (instance of your module) is created. Here you define the variables that are local in the module with `defvar()`.

- `void start();`

  Called by Roxen just before the module should be ready to receive a request (requests are handled with the module specific callback functions).

- `void stop();`

  Called by Roxen just after the module is done with a request (requests are handled with the module specific callback functions).

- `string status();`

  Called by the configuration interface at any time to get the status of the module. Should return a string, containing HTML code that can be fitted in the `<dd>` part of a definition list.

MODULE TYPE	INITIALISATION ROUTINES	CALLBACK ROUTINES
All	create, start	status, info, check_variable, query_name
MODULE_AUTH		auth, userinfo, userlist, user_from_id
MODULE_DIRECTORIES		parse_directory
MODULE_EXTENSION	query_extensions	handle_extension
MODULE_FILE_EXTENSION	query_file_extensions	handle_file_extension
MODULE_FILTER		filter
MODULE_FIRST		first_try
MODULE_LAST		last_resort
MODULE_LOGGER		log
MODULE_LOCATION	query_location	find_file, find_dir, stat_file, real_file
MODULE_MAIN_PARSER		add_parse_module, remove_parse_module
MODULE_PARSER	query_tag_callers, query_container_callers	Provided by the initialisation routines
MODULE_TYPES		type_from_extension
MODULE_URL		remap_url

TABLE 13.2 *Module type specific callback functions*

- `string info();`

  If you define this function, its result is used instead of the third element in the return value of `register_module()`. If you look at the beginning of the example, you'll recall that this string was used to describe the module in the configuration interface.

- `string|void check_variable(string variable, mixed will_be_set_to);`

  If you need to check the value of your module variables before they are ”permanently” set, this function is where you should do it. Usually you don’t have to check on the validity of the values of variables, since all values that the administrator can set them to are valid.

  Writing `string|void` means that the function does not necessarily return a value.

- `string query_name();`

  Returns the name of the module, and is used instead of the second element in the array returned by `register_module()`. This can be very useful if you allow more than one copy of a module.

  The user can, however, use any module name he/she wishes in the configuration interface.

- `array auth(array from);`
  `array userinfo(string username);`
  `array userlist();`
  `array user_from_uid(int uid);`

  See the **userdb.pike** module. Usually you don’t have to implement your own user database and authentification module, and it is not part of the fundamentals of Roxen programming.

- `mapping parse_directory(object request_id);`

  Given an internal url (in `request_id->not_query`) this function returns an HTML-coded directory listing or an index file depending on how the virtual server making use of the module is set up.

- `array (string) query_extensions();`
  `array (string) query_file_extensions();`

  Return an array with the extensions that this module is intended to handle. It is suitable to implement this as follows:

```
array (string) query_extensions()
{
   return query(”extensions”);
}
```

  This requires that the variable `extensions` is defined, as a TYPE_STRING_LIST, see below.

- `mapping handle_extension(string extension, object request_id);`

  `mapping handle_file_extension(object file, string ext, object request_id);`

  Return either a result or 0 (zero). Called by Roxen when the module should handle an extension.

  The case `handle_extension` is hardly ever used and the only place where you can find it in the distribution is in the timestamp module.

  The ususal case, though, is to make file extension modules. In that case it's the second definition that applies. The object *file* is a clone of `/precompiled/file`, taken from Pike. One useful method in this class is `read()`.

- `mapping first_try(object request_id);`

  If a a response mapping is returned, this will be used as the return value.

  This function is never called by internal requests, i.e. if you use `<insert>` and similar things.

- `mapping last_resort(object request_id);`

  Used by the relay module, if priority is set to `last`.

  Called only if no other module has found a matching document, but never by internal requests.

- `string query_location();`

  Returns which position in the virtual filesystem the module should have. Use a variable here since this lets the user choose where the module should end up.

- `object|mapping find_file(string file_name, object request_id);`

  Returns an open fileobject of the class `/precompiled/file` if there is a file matching `file_name` that this module handles, or a response mapping (usually generated with the `http_*` functions) if you do

not wish that extension modules should be used. The mapping can be very practical if you want the user to enter a password, or in redirecting to a new file.

Instead of using the `clone()` function you can use the much more intuitive `new(File)` in Roxen. `File` is a constant that is usable everywhere except when inheriting when you have to use `/precompiled/file`.

A useful convenience function is `open(filename, mode)`, which returns one of these open fileobjects.

N.B.: If the module has a mount point (the value that `query_location()` returns) `/foo/` and the file `/foo/bar/` is asked for, the location module will get `bar/` as the first argument, not `/foo/bar/`, i.e. the mount point is stripped from the filename.

- `array find_dir(string dir_name, object request_id);`

  Returns an array with filenames if there is a directory matching `dir_name` or 0 (zero).

  There is a default `find_dir` in `module.pike` that returns 0. So if you don't want to return any directory listings you don't have to define this function.

- `array stat_file(string file_name, object request_id);`

  Return the result or zero if there is a file matching `file_name`.

  If you don't want to, you don't have to define this function, but the directory module and some other modules might require this in order to function exactly as "regular" filesystem.

- `string real_file(string file_name, object request_id);`

  Return what the file `file_name` really is called, e.g. if the module *test* looks for it's files in `/usr/www/foo/`, and the file `bar` is requested, it should return `/usr/www/foo/bar`.

  This method isn't necessary, it might for instance happen that your files really dont exist on the disk. However, it speeds up things like CGI and Pike scripts. Both of these work perfectly fine with only "virtual" files, though.

- `void add_parse_module(object module);`
  `void remove_parse_module(object module);`

  Register and unregister a parse module in the main_parse module. Only used by the main parser module. We recommend you to study **htmlparse.pike** for real working examples.

- `mapping (string:function(string, mapping(string:string), object, object, mapping(string:string):string))`
  `query_tag_callers();`
  `mapping (string:function(string, mapping(string:string), string, object, mapping(string:string):string))`
  `query_container_callers();`

  `query_tag_callers()` and `query_container_callers` are called from the main parser to register a parse module. The return value should be a mapping on the following form:

  `(["tag":function, ... ]),`

  or an empty mapping ([]).

- `array (string|int) type_from_extension(string ext);`

  Called from Roxen in the content-type module if no other module returned what kind of file type this file was. The result is `({content-type, content-encoding})` or `0` (zero). There can, as noted earlier, only be one content-type module in each virtual server.

- `object|mapping remap_url(object request_id);`

  Returns `request_id` (with some variables modified), in which case it is used to make a new request, meaning that the module will be called again, so be careful not to create an indefinite recursion. It can also return a mapping with the help of some of the `http_*()` functions.

  Finally you could return zero, in which case the program just continues.

  N.B.: You can change variables in `request_id` and then return zero. This is not allowed in any other module type than MODULE_PARSER.

THE COMPLETE MODULE

Now, back to the example! As you can see in table 13.2 on page 169 this type of module (MODULE_PARSER in case you'd forgotten) must have the initialisation functions `query_container_callers` and `query_tag_callers`. As you can see above, they are quite hard on the fingers, and since we do not wish to torture our poor digits unnecessarily we write:

```
mapping query_tag_callers()
{

}
mapping query_container_callers()
{

}
```

Now that the function definitions exist, what should they return? They are supposed to define the two tags that we wish our module to provide. Let us complete the entire module and fill in the details.

```
/* A simple parse type module for Roxen.
* This module defines two new tags, foo and bar.
* The first is a container and the second is a
* stand-alone tag.
*/
#include <module.h>
inherit "module";
inherit "roxenlib";
array register_module()
{
   return({
     MODULE_PARSER,
       "Gazonk",
       "The gazonk module. Adds a container and "
       "a non-container: "
       "&lt;foo&gt;...&lt;/foo&gt; and "
       "&lt;bar&gt;",
       0,
       1,
   });
}
/* A container gets the contents as the third
* argument. Example: <foo Bar=Gazonk>Hi!</foo> -->
* container_foo("foo", (["bar":"Gazonk"]),
* "Hi!", ...);
* Note the lowercasing of Bar.
*/
string container_foo(string tag_name, mapping
                     arguments string contents,
                     object request_id, mapping
                     defines)
{
   if (arguments->lower)
     return lower_case(contents);
   if (arguments->upper)
     return upper_case(contents);
   if (arguments->reverse)
     return reverse(contents);
}
string tag_bar(string tag_name, mapping arguments,
               object request_id, object file,
               mapping defines)
{
```

```
    int i;
    string res="";
    if (arguments->num)
      i=(int)arguments->num;
    else
      i=30;
  #define LETTERS (("abcdefghijklmnopqrstuvwxyzå-
  äö")/"")
    while(i--)
      res += LETTERS[random(sizeof(LETTERS))];
  #undef LETTERS
    return res;
  }
  mapping query_tag_callers()
  {
    return (["bar":tag_bar, ]);
  }
  mapping query_container_callers()
  {
    return (["foo":container_foo, ]);
  }
}
```

Now we actually have a working parse module. If we start using the module (see "Start using your new module" on page 185) you can enter `<foo lower|upper|reverse>Some string</foo>` in your pages which should give you the text "Some string", changed as indicated by the attribute; `lower` changes the string to lowercase, `upper` to uppercase letters and `reverse` reverses the string, printing it backwards.

If you put `<bar>` on a page, you'll get a randomly selected string of 30 characters. If you include an integer attribute, e.g. `<bar num=17>`, you'll get a randomly selected string of , in this case 17, characters.

RETURNING VALUES

Putting together a mapping of the kind that should be returned in most cases (extension, last, first and file extension) when you've found a file might be a bit tricky. Therefore there are several convenient helper functions available.

All these functions are defined in `pike/http.pike` which is inherited by `roxenlib`, which it is, in turn, a good idea to inherit in your own modules. So that you won't have to study the source code you'll find the interes-

ting contents, the library functions, of roxenlib and pike/http.pike documented below.

HTTP.PIKE

- `mapping http_low_answer(int errno, string data);`

  Return a filled out structure with the error and data specified. The error is in fact the status response, so ”200” is ”OK”, and ”500” is ”Internal Server Error”, etc.

  Mostly used by the other functions in http.pike.

- `mapping http_pipe_in_progress();`

  Returns a structure that indicates to Roxen that it should leave this socket alone. This is not to be used normally, since it might cause a socket leak. You have to know what you are doing to use it.

  Study one of the proxy modules to the point where you can recite it in your sleep for more information.

- `mapping http_string_answer(string text, string|void type);`

  Convenience function to use in Roxen modules. When you just want to return a string of data, with an optional type, this is the easiest way to do it if you don’t want to worry about the internal Roxen structures.

- `mapping http_file_answer(object file, string|void type, void|int len);`

  Like `http_string_answer()`, but it returns a file object instead.

- `mapping http_redirect(string URL, object|void request_id)`

  Send a redirect to the file URL, which can be either relative or absolute.

  If relative, `request_id` *must* also be supplied to resolve the path correctly.

- `mapping http_auth_required(string realm, string message)`

  Send an Auth-challenge to the client, from the realm `realm`. A default message that will be shown if the user choose not to send any password can be supplied.

- `mapping http_proxy_auth_required(string realm, string message);`

  Like `http_auth_required()`, but for proxies.
- `string cern_http_date(int t);`

  Format the timestamp ”t” according to the CERN common log-file date format.
- `string http_date(int t);`

  Returns an http_date, as specified by the HTTP-protocol standard. This is used for the Last-Modified and Time headers in the reply.
- `string http_encode_string(string f);`

  Encode the string ”f” so that it can safely be used as a URL.
- `string http_encode_cookie(string f);`

  Encode the string ’f’ so that it can safely be used as a cookie value or name.
- `string http_decode_string(string f);`

  This one is of course the opposite of `http_encode_string()` but is seldom needed.

ROXENLIB

Roxenlib inherits http.pike and so all those convenience functions are at your command if you inherit roxenlib in your programs.

- `string decode_mode(int m);`

  Returns an `ls -l` style file permission mode from a numerical one, and also prepends the type of the file.

  Example 13.11)

  ```
  File, <tt>rw-r-----</tt>
  ```

  Example 13.12)

  ```
  Dir, <tt>rwxr-xr-x</tt>
  ```

  Should probably take a format string as well.

  This function is here more or less only for historical reasons.
- `int _match(string w, array (string) a);`

  Try to find a pattern from `a` that matches `w`. If one is found, return 1, otherwise 0. This function ought to be be renamed.

- `int is_modified(string a, int t, void|int len);`

  Is the time described in `a` before the integer time in `t`, as gotten from `time()`.

  `a` is as given by the `If-Modified:` header, and if `len` is included in the header, and in the call as well, it is used as a first quick check to see if the file has indeed been modified.

  You will probably not have to use this unless you are writing a proxy module or a protocol module.

- `string short_name(string long_name);`

  Return a short name, useful as a filename, from the given long name.

  The short name is not really all that much shorter, but it is often easier to handle with a shell.

- `string parse_rxml(string what, object|void id);`

  Parse the string `what`, as if it was generated as a reply from the request described in `id`, and then sent from the the HTML parser.

  If no `id` is supplied, a dummy will be generated. Tags like `<clientname>` will not work then.

  Can be quite useful in modules, but not in scripts, since if a script returns a string (i.e. does *not* use any of the `http_*()` functions to return a value, but instead simply returns a string) it will be parsed automatically.

- `string dirname( string file );`

  Return the directory name from the file `file`, just like the `dirname` program available in most Unix flavours.

- `string add_pre_state(string url, multiset state);`

  Add the prestate described in the multiset `state` to the URL `url`.

  Example 13.13)

  `add_pre_state(”/index.html”, (< ”foo”, ”bar” >))`

  returns

  `/(bar,foo)/index.html`

- `string add_config( string url, array config, multiset prestate )`

  Add a config modification *and* a prestate to a URL.

Example 13.14)

```
add_config( "/index.html", ({ "+foo", "-bar" }),
(< "foo", "bar" >));
```

gives

```
/<+foo,-bar>/(bar,foo)/index.html
```

When this request is sent to Roxen, a redirect will be generated immediately to `/(bar,foo)/index.html`, and the cookie `config` database will be modified.

- `string msectos(int t);`

  This one should really be named `describe_msec` or something

  Returns a description of the time `t`, given in milliseconds.

- `string extension( string f );`

  Return the extension of the file `f`.

  Example 13.15)

  ```
  extension("foo.gif");
  ```

  gives

  ```
  "gif"
  ```

  Example 13.16)

  ```
  extension("foo.html~");
  ```

  gives

  ```
  "html"
  ```

  Example 13.17)

  ```
  extension("foo.html.old");
  ```

  gives

  ```
  "html"
  ```

- `int backup_extension( string f )`

  Return `1` if filename of `f` indicates that it might be an backup copy of an original file.

- `int get_size(mixed x);`

  The arguments can be anything and it returns the memory usage of the argument.

- `int ipow(int what, int how);`

  `What` to the power of `how`. There is, of course, also a `pow(float what, float to);`

  These functions do not really come from **roxenlib** but from Pike.

- `string simplify_path(string file);`

  This one will remove `../`, `./` etc. in the path.

  Example 13.1)

  `simplify_path("foo/../bar/../")`

  gives

  `"/"`

  Example 13.2)

  `simplify_path("foo/../bar/././")`

  gives

  `"/bar/"`

- `string short_date(int timestamp);`

  Returns a reasonably short date string from a time integer.

- `string int2roman(int m)`

  Return the integer `m` as a roman number.

  That is, `int2roman(10)` returns `"X"`, etc.

- `string number2string(int n,mapping m,mixed names);`

  Convert the number `n` to a string, using the conversion indicated by `m->type`, and optionally the conversion function `names`. This is used in all tags in the main RXML parser that converts a number to a string. Look at the **htmlparse.pike** for live examples.

- `string image_from_type( string t );`

  Return a suitable `internal-gopher-*` image from the mimetype specified by `t`. The function is used by the directory modules.

- `string sizetostring( int size );`

  Describe the size `size` of anything, e.g. a file, an amount of data or whatever, as a string. Size is measured in bytes and the unit is reasonably intelligently scaled using kB, MB, etc.

MODULE VARIABLES

If you have studied the configuration interface, you have probably noticed that almost all modules contain a plethora of variables that can be set by the administrator.

How do you do if you want variables of this kind in your module, since it is nice to let the user himself configure as much as possible?

You use the function `defvar()` in `create()` and then `query()` if you want to know the value of one of the variables.

```
int defvar(string name, mixed value, string
long_name, int type, string documentation_string[,
array choices, function|int hidden]
```

where `name` is the name, of the variable, used internally in your program to obtain its value through `query("name")` or `QUERY(name)`; `value` is the value of the variable; `long_name` is the variable name seen in the configuration interface; and `type` is, of course, the type of variable, see table 13.3 on page 183.

Now to illustrate this let's write an example of `create()`.

VARIABLE TYPE	EXPLANATION
TYPE_STRING	The variable is a string like ”Informative text”
TYPE_FILE	A file variable. It is in fact like a string variable, but the help text seen by the user differs.
TYPE_LOCATION	A position in the virtual file system. Once again you treat this like a string, but the help text is different.
TYPE_INT	An integer
TYPE_DIR	A directory. The user can only enter existing directories and they always end with a ”/”, so you won’t have to care about that in your modules.
TYPE_FLOAT	A floating point number.
TYPE_TEXT_FIELD	A field that can contain several lines of text.
TYPE_FLAG	A variable that is either true or false is of this type.
TYPE_COLOR	A colour variable, a number between 0 and 16777215=2^{24}-1. This number is a three-byte value where each of the R, G and B colours is represented by one of the bytes. This was earlier used in the configuration interface.
TYPE_PASSWORD	A password, the variable is automatically encrypted when set.
TYPE_STRING_LIST	An array of strings, like {”foo”, ”bar”, ”gazonk”}.
TYPE_MULTIPLE_STRING	One of several strings. The strings are chosen from the array sent with the optional field `choices`.
TYPE_INT_LIST	An array of integers.

TABLE 13.3 *Available variable types.*

VARIABLE TYPE	EXPLANATION
TYPE_MULTIPLE_INT	One of several integers. Cf. TYPE_MULTIPLE_STRING above.
TYPE_DIR_LIST	An array of directories.
TYPE_FILE_LIST	An array of files, treated just like variables of the TYPE_STRING_LIST type but the help text differs.
TYPE_MODULE	The variable is a module object.

TABLE 13.3 *(Cont.)Available variable types.*

```
void create()
{
   defvar("BG", 1, "Configuration Interface "
          "Background", TYPE_FLAG, "Should the "
          "background be set by the "
          "configuration interface?");
   defvar("NumAccept", 1, "Number of accepts to "
          "attempt", TYPE_MULTIPLE_INT,
          "The maximum number of accepts to "
          "attempt for each read callback from "
          "the main socket. <p>Increasing this "
          "will make the server faster for "
          "users making many simultaneous "
          "connections to it, or if you have a "
          "very busy server.", ({1, 2, 4, 8, 16,
          32, 64, 128, 256, 512, 1024}));
   defvar("ConfigurationPort", 22202,
          "Configuration port", TYPE_INT, "The "
          "port number of the configuration "
          "interface. Anything will do, but you "
          "will have to remember it to be able "
          "to configure the server.");
}
```

In this piece of code three variables are defined (actually, this was cut out of the central parts of Roxen, and is not from any module, but it works in the same way).

The first is a flag (`BG`), whose value is `on`, then an integer (`NumAccept`)chosen from a list of integers (2^n, $0 \le n \le 10$) and finally a single integer (`ConfigurationPort`).

START USING YOUR NEW MODULE

When you have written your module it is time make it accessible for Roxen. Do this by placing the file in the module directory of your server, i.e. in the directory that you have chosen for your module directory, cf. “The fundamentals of a module” on page 166. Then all you have to do is to access the configuration interface and choose ADD MODULE as usual; you will now be able to choose your newly added module! As simple as that.

FIGURE 13.2 *The Reload button.*

And if you change the code of a module, you can focus on that module and hit RELOAD, and voilà! Your new code is immediately loaded and starts working and this without ever stopping the server.

If, or perhaps when, your module does not show up on the module selection page you should check the debug log for errors.

ROXEN MANUAL
APPENDICES

Detta dokument har framställts med FrameMaker 4.0.4

APPENDIX A

REGULAR EXPRESSIONS

INTRODUCTION

A regular expression specifies a set of character strings. A member of this set of strings is said to be matched by the regular expression. Some characters have special meaning when used in a regular expression; other characters stand for themselves.

The regular expressions available for use with the regexp functions are constructed as follows.

EXPRESSION MEANING

- c; the character c where c is not a special character.
- \c; the character c where c is any character, except a digit in the range 1-9.
- ^; the beginning of the line being compared.
- $; the end of the line being compared.
- .; any character in the input.
- [s]; any character in the set s, where s is a sequence of characters and / or a range of characters, for example, [a-z].

- `[^s]`; any character not in the set s, where s is defined as above.
- `r*`; zero or more successive occurrences of the regular expression r. The longest leftmost match is chosen.
- `rx`; the occurrence of regular expression `r` followed by the occurrence of regular expression `x`. It is a *concatenation* of expressions.
- `r\{m,n\}`; any number of `m` through `n` successive occurrences of the regular expression `r`. The regular expression `r\{m\}` matches *exactly* `m` occurrences while `r\{m,\}` matches *at least* `m` occurrences.
- `(r\)`; the regular expression `r`. When `\n` (where `n` is a number greater than zero) appears in a constructed regular expression, it stands for the regular expression `x` where `x` is the nth regular expression enclosed in `\(` and `\)` that appeared earlier in the constructed regular expression. For example, `\(r\)x\(y\)z\2` is the concatenation of regular expressions `rxyzy`.

Characters that have special meaning except when they appear within square brackets (`[]`) or are preceded by `\` are

```
. (dot)
* (asterisk)
[ (left bracket)
\ (backslash)
```

Other special characters, such as "`$`" have special meaning in more restricted contexts.

The character "`^`" at the beginning of an expression permits a successful match only immediately after a newline, and the character "`$`" at the end of an expression requires a trailing newline.

Two characters have special meaning only when used within square brackets. The character "`-`" denotes a range, `[c-c]`, unless it is just after the open bracket or before the closing bracket, `[ -c]` or `[c-]` in which case it has no special meaning. When used within brackets, the character "`^`" has the meaning complement of if it immediately follows the open bracket (example: `[^c]`); elsewhere between brackets (example: `[c^]`) it stands for the ordinary character "`^`".

The special meaning of the "`\`" operator can be escaped only by preceding it with another "`\`", i.e. "`\\`".

APPENDIX B

A PIKE RECORD DATABASE

This is the complete listing of the big example of "The Pike Quick Guide". There are no comments in the code so please refer to that chapter for explanations of what the various functions do.

For more information on Pike, point your favourite web browser at http://pike.infovav.se/ where you'll find not only all the reference information you can wish for, but also a few examples.

But of course there's nothing like practice. Get your hands dirty and hack as much as you can. That is the only way to really understand what's going on.

The listing begins on the next page.

```
#!/usr/local/bin/pike

mapping (string:array(string)) records =
([
”Star Wars Trilogy” :
   ({
   ”Fox Fanfare”,
   ”Main Title”,
   ”Princess leia’s Theme”,
   ”Here They Come”,
   ”The Asteriod Field”,
   ”Yoda’s Theme”,
   ”The Imperial March”,
   ”Parade of th Ewoks”,
   ”Luke and Leia”,
   ”Fight with Tie Fighters”,
   ”Jabba the Hut”,
   ”Darth Vader’s Death”,
   ”The Forest Battle”,
   ”Finale”,
   })
]);


void list_records()
{
   int i;
   array (string) record_names=sort(indices
        (records));

   write(”Records:\n”);
   for(i=0;i<sizeof(record_names);i++)
     write(sprintf(”%3d: %s\n”, i+1,
                   record_names[i]));
}
```

```
void show_record(int num)
{
   int i;
   array (string) record_names=sort(indices
          (records));
   string name=record_names[num-1];
   array (string) songs=records[name];

   write(sprintf("Record %d, %s\n",num,name));
   for(i=0;i<sizeof(songs);i++)
     write(sprintf("%3d: %s\n", i+1, songs[i]));
}


void add_record()
{
   string record_name=readline("Record name: ");
   records[record_name]=({});
   write("Input song names, one per line. End with
        '.' on it's own line.\n");
   while(1)
   {
     string song;
     song=readline(sprintf("Song %2d: ",
          sizeof(records[record_name])+1));
     if(song==".") return;
     if(strlen(song))
       records[record_name]+=({song});
   }
}
```

```
void save(string file)
{
   string name, song;
   object o;
   o=clone((program)"/precompiled/file");

   if(!o->open(file,"wct"))
   {
     write("Failed to open file.\n");
     return;
   }

   foreach(indices(records),name)
   {
     o->write("Record: "+name+"\n");
     foreach(records[name],song)
       o->write("Song: "+song+"\n");
   }

   o->close();
}
```

```
void load(string file)
{
   object o;
   string name="ERROR";
   string file_contents,line;

   o=clone((program)"/precompiled/file");
   if(!o->open(file,"r"))
   {
     write("Failed to open file.\n");
     return;
   }

   file_contents=o->read(0x7fffffff);
   o->close();

   records=([]);

   foreach(file_contents/"\n",line)
   {
     string cmd, arg;
     if(sscanf(line,"%s: %s",cmd,arg))
     {
       switch(lower_case(cmd))
       {
         case "record":name=arg;
         records[name]=({});
         break;

         case "song":records[name]+=({arg});
         break;
       }
     }
   }
}
```

```
void delete_record(int num)
{
   array (string) record_names=sort(indices
        (records));
   string name=record_names[num-1];

   m_delete(records,name);
}


void find_song(string title)
{
   string name, song;
   int hits;
   title=lower_case(title);

   foreach(indices(records),name)
   {
     foreach(records[name],song)
     {
       if(search(lower_case(song), title) != -1)
       {
         write(name+"; "+song+"\n");
         hits++;
       }
     }
   }

   if(!hits) write("Not found.\n");
}
```

```
int main(int argc, string * argv)
{
   string cmd;
   while(cmd=readline("Command: "))
   {
      string args;
      sscanf(cmd,"%s %s",cmd,args);

      switch(cmd)
      {
         case "list":
         if((int)args)
         {
            show_record((int)args);
         } else {
            list_records();
         }
         break;

         case "quit":exit(0);

         case "add":add_record();
         break;

         case "save":save(args);
         break;

         case "load":load(args);
         break;

         case "delete":delete_record((int)args);
         break;

         case "search":find_song(args);
         break;
      }
   }
}
```

APPENDIX C

TABLES

RXML TAGS

All the RXML tags that are present in the basic distribution of Roxen are listed here in table format. For a complete description of every tag refer to the full documentation on the page referred to in the table.

TAGNAME	CONTAINER	ATTRIBUTES	PAGE REFERENCE
<accessed>	No	file, reset, silent, since, cheat, factor, per, precision, type, addreal	page 46
<aconf>	Yes	N / A	page 48
<apre>	Yes	href, prestates	page 49
<blink>	Yes	N / A	page 50
<bofh>	No	N / A	page 50
<clientname>	No	full	page 51
<comment>	Yes	N / A	page 51

TABLE C.1 *The Tags of the RoXen Macro Language (RXML).*

TAGNAME	CONTAINER	ATTRIBUTES	PAGE REFERENCE
<comment> (inside the <lysator> container.)	Yes	N / A	page 51
<date>	No	day, hour, minute, second and the date related attributes	page 51
<define>	Yes	name	page 51
<doc>	Yes	pre	page 54
<dthought>	No	N / A	page 54
<else>	Yes	N / A	page 56
<endtable>	No	N / A	page 54
<fot>	No	N / A	page 55
<h>	Yes	level, hr	page 55
<header>	No	name, value	page 55
<icon>	No	as for <IMG> of HTML	page 56
<icons>	Yes	N / A	page 56
<if>	Yes	name, supports, prestate, defined,referer, date, host, user, file, wwwfile, before, after, inclusive, and, not, or	page 56

TABLE C.1 *(Cont.)The Tags of the RoXen Macro Language (RXML).*

TAGNAME	CONTAINER	ATTRIBUTES	PAGE REFERENCE
<insert>	No	cookie, cookies, file, name, nocache, variable, variables	page 51
<item>	Yes	linkto, icon, title	page 61
<language>	No	full	page 62
<lysator>	Yes	pretxt, title, text	page 62
<modified>	No	by and the date related attributes	page 62
<otherwise>	No	N/A	page 56
<picture>	No	as for <IMG> of HTML	page 62
<quote>	No	start, end	page 63
<random>	Yes	separator	page 63
<referer>	No	N/A	page 64
<remove_cookie>	No	name	page 64
<return>	No	code	page 64
<right>	Yes	N/A	page 64
<set_cookie>	No	name, value, persistent	page 65
<signature>	No	name, realname, email, link, nolink	page 65
<smallcaps>	Yes	size, small, space	page 65
<source>	No	N/A	page 65
<tablify>	Yes	N/A	page 66
<tablist>	No	names, 1, [2,..], selected, bg, fc, tc, font, scale	page 66

TABLE C.1 *(Cont.)The Tags of the RoXen Macro Language (RXML).*

TAGNAME	CONTAINER	ATTRIBUTES	PAGE REFERENCE
<user>	No	name, realname, email, link, nolink	page 67
The date related attributes		type, lang, part, time, date, capitalize, lower, upper	page 67

TABLE C.1 *(Cont.)The Tags of the RoXen Macro Language (RXML).*

HEADER RESPONSE LINES

When sending headers, using the <header> RXML tag, back to the client after a request you can send many things. In figure C.2 on page 203 are some suggestions on what to send. Probably the most interesting header is the WWW-Authenticate header, used when restricting access.

For more information on the use of the header tag see "<HEADER>" on page 55. For in-depth information on headers you'll find several texts on this subject if you browse around the Internet.

HEADER NAME	DESCRIPTION
Retry-After	The Retry-After header field can be used with ”502 Unavailable” to indicate how long the service is expected to be unavailable
WWW-Authenticate	Must be included with a ”401 Unauthorized” response, like this: `<header name=WWW-Authenticate value=”Basic; Realm=any”>`. The realm attribute is required for all access authentication schemes which issue a challenge (i.e. tells the browser to obtain a user name and a password). The realm value, in combination with the root URL of the server being accessed, will be presented to the user.
Location	Must be included with a redirect of any kind, like this: `<header name=Location value=URL>`.
Content-Language	The language of the document
Content-Encoding	The encoding of the document
Content-Type	The content type, usually text / html.
Derived-From	Indicates which document this document is derived from.
Expires	After the date indicated by this header, the document must be refetched from the server.

TABLE C.2 *Some possible heads in the <header> tag.*

HTTP RESULT CODES

When a request has been received and treated by an http server it sends the result back to the client. Figure C.3 on page 204 contains the possible responses, some of which are more than a little bit cryptic. Those that you understand are probably the ones you are interested in.

CODE	MEANING
200 Document follows	The request has been fulfilled and an entity corresponding to the requested resource is being sent in the response.
201 Created	The request has been fulfilled and resulted in a new resource being created.
202 Accepted	The request has been accepted for processing, but the processing has not been completed.
203 Provisional information	The returned metainformation in the Entity-Header is not the definitive set as available from the origin server, but is gathered from a local or a third-party copy.
204 No Content	The server has fulfilled the request but there is no new information to send back. If the client is a user agent, it should not change its document view.
300 Moved	The requested resource is available at one or more locations and a preferred location could not be determined via content negotiation.
301 Moved permanently	Requires the `Location' header, see the <head> tag below. The requested resource has been assigned a new permanent URI and any future references to this resource must be done using the returned URI.
302 Moved temporarily	Requires the `Location' header, see the <head> tag below. The requested resource resides temporarily under a different URI. Since the redirection may be altered on occasion, the client should on future requests from the user continue to use the original Request-URI and not the URI returned in the URI-header field and Location fields.
304 Not modified	The document has not been modified
400 Bad Request	The request had bad syntax or was inherently impossible to satisfy. The client is discouraged from repeating the request without modifications.

TABLE C.3 *HTTP result codes.*

CODE	MEANING
401 Access denied	The request requires user authentication. The response must include a WWW-Authenticate header field containing a challenge applicable to the requested resource. The client may repeat the request with a suitable Authorization header field. This might, for instance, be used inside a <deny user=...> tag.
402 Payment Required	The user has to pay you to get the information.
403 Forbidden	The request is forbidden because of some reason that remains unknown to the client.
404 No such file or directory	The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent.
405 Method not allowed	The method specified in the Request-Line is not allowed for the resource identified by the Request-URI.
408 Request timeout	n / a
409 Conflict	n / a
410 This document is no more	The requested resource is no longer available at the server and no forwarding address is known.
500 Internal Server Error	n / a
501 Not implemented	n / a
502 Service Unavailable	n / a
503 Gateway Timeout	n / a

TABLE C.3 *(Cont.)HTTP result codes.*

AVAILABLE MODULES

In the distribution of Roxen we have decided to include even unsupported modules. The ones listed here are documented elsewhere in this manual and most of them are supported unless otherwise indicated. Note that this list will grow as we incorporate more and more modules in our collection.

MODULE	TYPE	PAGE REFERENCE
BOFH	Location	page 112
CGI Executable Support	Location / File extension	page 112
Client Logger	First	page 114
Configuration interface		page 114
CONNECT method	First try / Proxy	page 115
Contenttypes	Content-type	page 115
Deep Thought	Parser	page 116
Directory parser	Directory	page 116
Explicit clock	Location	page 117
FastCGI	Location / File extension	page 117
Fast directory parser	Directory	page 118
Filesystem Module	Location	page 119
FTP Gateway	Location	page 120
Gopher Gateway	Location	page 122
.htaccess support	Location	page 122
HTTP Proxy	Location	page 122
HTTP Relay	Last / First	page 123
Indexfiles	Directory	page 124
Indirect href	Parser	page 124
ISMAP Image Map	File extension	page 125

TABLE C.4 *Available modules in the basic Roxen distribution.*

MODULE	TYPE	PAGE REFERENCE
Language	URL / Directory parser	page 125
Logging disabler		page 127
Lysator specific parsing	Parser	page 127
Main RXML Parser	Location / Parser / File extension	page 128
Pike Script support	File extension	page 129
Redirect module v2.0	First	page 129
Secure Filesystem	Location	page 131
Status Monitor	Location	page 131
Tablify	Parser	page 132
Tablist	Parser	page 132
Timestamp	Extension	page 133
User Database	Auth	page 133
User Filesystem	Location	page 134
User logger	Location	page 135
WAIS Gateway	Location	page 136

TABLE C.4 *(Cont.)Available modules in the basic Roxen distribution.*

INDEX

Detta dokument har framställts med FrameMaker 4.0.4