Programming, using and understanding

Pike
by Fredrik HübINETTE
This book was written with the intention of making anybody with a little programming experience able to use Pike. It should also be possible to gain a deep understanding of how Pike works and to some extent why it works the way it does from this book. It will teach you how to write your own extensions to Pike. I have been trying for years to get someone else to write this book, but since it seems impossible without paying a fortune for it I will have to do it myself. A big thanks goes to Ian Carr-de Avelon* and Henrik Wallin ¡hedda@idonex.se¡ for helping me iron out some of the rough spots. The book assumes that you have programmed some other programming language before and that you have some experience of UNIX.
Contents

0.1 Overview ................................................. 4
0.2 The history of Pike ................................. 4
0.3 A comparison with other languages ............... 5
0.4 What is Pike ................................................. 5
0.5 How to read this manual ............................. 6

1 Getting started ............................................ 7
  1.1 Your first Pike program .......................... 7
  1.2 Improving hello_world.pike ..................... 8
  1.3 Further improvements ......................... 9
  1.4 Control structures ................................. 11
  1.5 Functions ................................................. 11
  1.6 True and false ..................................... 12
  1.7 Data Types .............................................. 12

2 A more elaborate example ............................... 15
  2.1 Taking care of input .............................. 17
    2.1.1 add_record() .................................. 17
    2.1.2 main() ........................................... 18
  2.2 Communicating with files ....................... 19
    2.2.1 save() ......................................... 19
    2.2.2 load() ......................................... 20
    2.2.3 main() revisited .............................. 21
  2.3 Completing the program ......................... 21
    2.3.1 delete() ...................................... 21
    2.3.2 search() ....................................... 22
## 3 Control Structures

3.1 Conditions .................................................. 25
   3.1.1 if ....................................................... 25
   3.1.2 switch .................................................. 26
3.2 Loops .......................................................... 27
   3.2.1 while ..................................................... 27
   3.2.2 for ......................................................... 28
   3.2.3 do-while .................................................. 28
   3.2.4 foreach .................................................... 29
3.3 Breaking out of loops .......................................... 29
   3.3.1 break ..................................................... 29
   3.3.2 continue .................................................. 30
   3.3.3 return .................................................... 30
3.4 Exercises ...................................................... 31

## 4 Data types

4.1 Basic types .................................................. 33
   4.1.1 int ......................................................... 33
   4.1.2 float ....................................................... 34
   4.1.3 string ..................................................... 35
4.2 Pointer types ................................................ 38
   4.2.1 array ...................................................... 38
   4.2.2 mapping ................................................... 40
   4.2.3 multiset .................................................. 42
   4.2.4 program .................................................. 43
   4.2.5 object .................................................... 45
   4.2.6 function .................................................. 48
4.3 Sharing data .................................................. 49
4.4 Writing data types ............................................ 50
## CONTENTS

### 5 Operators

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Arithmetic operators</td>
<td>53</td>
</tr>
<tr>
<td>5.2</td>
<td>Comparison operators</td>
<td>56</td>
</tr>
<tr>
<td>5.3</td>
<td>Logical operators</td>
<td>56</td>
</tr>
<tr>
<td>5.4</td>
<td>Bitwise/set operators</td>
<td>57</td>
</tr>
<tr>
<td>5.5</td>
<td>Indexing</td>
<td>58</td>
</tr>
<tr>
<td>5.6</td>
<td>The assignment operators</td>
<td>59</td>
</tr>
<tr>
<td>5.7</td>
<td>The rest of the operators</td>
<td>61</td>
</tr>
<tr>
<td>5.8</td>
<td>Operator precedence</td>
<td>62</td>
</tr>
<tr>
<td>5.9</td>
<td>Operator functions</td>
<td>63</td>
</tr>
</tbody>
</table>

### 6 Object orientation

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Terminology</td>
<td>65</td>
</tr>
<tr>
<td>6.2</td>
<td>The approach</td>
<td>66</td>
</tr>
<tr>
<td>6.3</td>
<td>How does this help?</td>
<td>66</td>
</tr>
<tr>
<td>6.4</td>
<td>Pike and object orientation</td>
<td>66</td>
</tr>
<tr>
<td>6.5</td>
<td>Inherit</td>
<td>67</td>
</tr>
<tr>
<td>6.6</td>
<td>Multiple inherit</td>
<td>70</td>
</tr>
<tr>
<td>6.7</td>
<td>Pike inherit compared to other languages</td>
<td>71</td>
</tr>
<tr>
<td>6.8</td>
<td>Modifiers</td>
<td>71</td>
</tr>
<tr>
<td>6.9</td>
<td>Operator overloading</td>
<td>72</td>
</tr>
<tr>
<td>6.10</td>
<td>Simple exercises</td>
<td>73</td>
</tr>
</tbody>
</table>

### 7 Miscellaneous functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>scanf</td>
<td>75</td>
</tr>
<tr>
<td>7.2</td>
<td>catch &amp;throw</td>
<td>76</td>
</tr>
<tr>
<td>7.3</td>
<td>gauge</td>
<td>77</td>
</tr>
<tr>
<td>7.4</td>
<td>typeof</td>
<td>77</td>
</tr>
</tbody>
</table>

### 8 Modules

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>How to use modules</td>
<td>81</td>
</tr>
<tr>
<td>8.2</td>
<td>Where do modules come from?</td>
<td>81</td>
</tr>
<tr>
<td>8.3</td>
<td>The . operator</td>
<td>81</td>
</tr>
<tr>
<td>8.4</td>
<td>How to write a module</td>
<td>82</td>
</tr>
</tbody>
</table>
8.5 Simple exercises ........................................... 82

9 File I/O ......................................................... 83
  Stdio ......................................................... 83
  9.1 File management - Stdio.File ............................. 83
    Stdio.File .................................................. 83
  9.2 Buffered file management - Stdio.FILE ................. 93
    Stdio.FILE .................................................. 93
  9.3 Standard streams - Stdio.stdin, stdout and stderr ... 94
    Stdio.Port ................................................... 94
  9.4 Listening to sockets - Stdio.Port ........................ 94
    Stdio.Port ................................................... 94
  9.5 UDP socket and message management - Stdio.UDP ..... 96
    Stdio.UDP ....................................................... 96
  9.6 Terminal management - Stdio.Terminfo ................. 98
    Terminfo ...................................................... 98
    9.6.1 Stdio.Terminfo.Termcap ............................ 99
    9.6.2 Stdio.Terminfo.Terminfo ......................... 100
  9.7 Simple input-by-prompt - Stdio.Readline .............. 100
    Stdio.Readline .............................................. 100
  9.8 Other Stdio functions ................................... 101
  9.9 A simple example ........................................ 105
  9.10 A more complex example - a simple WWW server ....... 105

10 Threads ..................................................... 113
  10.1 Starting a thread ...................................... 113
  10.2 Threads reference section .............................. 114
    Thread.Mutex .................................................. 114
    Thread.Condition ............................................. 116
    Thread.Fifo .................................................... 117
    Thread.Queue .................................................. 118
    Thread.thread_local ........................................... 119
  10.3 Threads example ....................................... 120
11 Modules for specific data types

11.1 String ................................................................. 123
    String ............................................................... 123

11.2 Array ................................................................. 125
    Array ................................................................. 125

12 Image ....................................................................... 135
    Image ................................................................. 135

    12.1 Image.Image ......................................................... 137
        Image.Image ......................................................... 137

    12.2 Image.Colortable .................................................. 171
        Image.Colortable .................................................. 171

    12.3 Image.Layer ......................................................... 180
        Image.Layer ......................................................... 180

    12.4 Image.Font ........................................................... 184
        Image.Font ........................................................... 184

    12.5 Image.colortable .................................................... 186
        Image.colortable .................................................... 186

    12.6 Image.Poly ........................................................... 186
        Image.Poly ........................................................... 186

    12.7 Image.Color ........................................................ 186
        Image.Color ........................................................ 186

    12.7.1 Image.Color.Color ............................................... 188

    12.8 Image.X .............................................................. 192
        Image.X .............................................................. 192

    12.9 Image.ANY .......................................................... 193
        Image.ANY .......................................................... 193

    12.10 Image.AVS .......................................................... 194
        Image.AVS .......................................................... 194

    12.11 Image.BMP .......................................................... 194
        Image.BMP .......................................................... 194

    12.12 Image.GD ............................................................ 196
        Image.GD ............................................................ 196

    12.13 Image.GIF .......................................................... 197
CONTENTS

Image.GIF .................................................. 197
12.14 Image.HRZ ............................................ 204
Image.HRZ ................................................. 204
12.15 Image.ILBM ........................................... 205
Image.ILBM ................................................ 205
12.16 Image.PCX ............................................. 207
Image.PCX ................................................ 207
12.17 Image.PNG ............................................. 208
Image.PNG ............................................... 208
12.18 Image.PNM ............................................. 210
Image.PNM ................................................. 210
12.19 Image.PSD ............................................. 212
Image.PSD ................................................ 212
12.20 Image.TGA ............................................. 212
Image.TGA ................................................. 212
12.21 Image.XBM ............................................. 213
Image.XBM ................................................ 213
12.22 Image.XCF ............................................. 214
Image.XCF ................................................ 214
12.23 Image.XWD ............................................. 215
Image.XWD ................................................ 215
12.24 Image.JPEG ............................................ 216
Image.JPEG ................................................. 216
12.25 Image.TIFF ............................................ 218
Image.TIFF ............................................... 218
12.26 Image.TTF ............................................ 219
Image.TTF ............................................... 219
12.26.1 Image.TTF.Face .................................... 220
12.26.2 Image.TTF.FaceInstance ......................... 221
12.27 Image.XFace ........................................... 221
Image.XFace .............................................. 221
13 Protocols ........................................... 223
   Protocols ........................................... 223
   13.1 Protocols.HTTP ................................. 223
      Protocols.HTTP ................................... 223
      13.1.1 Protocols.HTTP.Query ..................... 225
   Protocols ........................................... 227
   13.2 Protocols.LysKOM ............................... 227
      Protocols.LysKOM .................................. 227
      13.2.1 Protocols.LysKOM.Session .................. 227
      13.2.2 Protocols.LysKOM.Connection .............. 230
      13.2.3 Protocols.LysKOM.Request .................. 230
   Protocols ........................................... 231
   13.3 Protocols.DNS .................................. 232
      Protocols.DNS ..................................... 232
      13.3.1 Protocols.DNS.client ....................... 232

14 Other modules ..................................... 233
   System ............................................. 233
   14.1 System ......................................... 233
   Process ............................................. 239
   14.2 Process ......................................... 239
      Process.create_process ............................ 239
      Process.Spawn ................................... 243
   Regexp .............................................. 244
   14.3 Regexp ......................................... 244
   Gmp .................................................. 246
   14.4 Gmp ............................................. 246
      Gmp.mpz ........................................... 246
   Gdbm ................................................ 248
   14.5 Gdbm ............................................ 248
   Getopt ............................................... 250
   14.6 Getopt .......................................... 250
   Gz .................................................... 254
   14.7 Gz .............................................. 254
14.16.1 Calendar.time_unit ................................. 305
14.16.2 Calendar.Gregorian .............................. 306
14.17 Parser ................................................. 308
14.18 Math .................................................. 308
  Math .................................................. 308
  14.18.1 Math.Matrix ..................................... 308
Calendar .................................................... 310
14.19 Calendar.time_unit .................................. 310
  Calendar.time_unit .................................. 310
14.20 Calendar.Gregorian .................................. 311
  Calendar.Gregorian .................................. 311
  14.20.1 Calendar.Gregorian. ........................... 311
  14.20.2 Calendar.Gregorian.Year ......................... 311
  14.20.3 Calendar.Gregorian.Stardate ................... 312
Crypto ...................................................... 312
14.21 Crypto.randomness .................................. 312
  Crypto.randomness .................................. 312
  14.21.1 Crypto.randomness.pike_random .................. 312
  14.21.2 Crypto.randomness.arcfour_random ............. 313
Geographical ................................................ 313
14.22 Geographical.Position ............................. 313
  Geographical.Position ............................. 313
14.23 Geographical.Countries ............................. 314
  Geographical.Countries ............................. 314
Image ........................................................ 316
Parser ....................................................... 317
14.24 Parser.SGML ......................................... 317
  Parser.SGML ......................................... 317
Protocols .................................................. 318
14.25 Protocols.HTTP ...................................... 318
  Protocols.HTTP ...................................... 318
  14.25.1 Protocols.HTTP.Query ........................... 320
14.26 Protocols.LysKOM ................................... 323
15 The preprocessor 329

16 Builtin functions 333

17 Pike internals - how to extend Pike 379
  17.1 The master object 379
  17.2 Data types from the inside 381
    17.2.1 Basic data types 381
    17.2.2 struct svalue 381
    17.2.3 struct pike_string 384
    17.2.4 struct array 387
    17.2.5 struct mapping 388
    17.2.6 struct object 391
    17.2.7 struct program 391
  17.3 The interpreter 391

A Terms and jargon 393

B Register program 395

C Reserved words 401

D BNF for Pike 403

E How to install Pike 405

F How to convert from old versions of Pike 407

G Image.Layer modes 409
CONTENTS

H Image.Color colors 415
This introduction will give you some background about Pike and this book and also compare Pike with other languages. If you want to start learning Pike immediately you can skip this chapter.

0.1 Overview

This book is designed for people who want to learn Pike fast. Since Pike is a simple language to learn, especially if you have some prior programming experience, this should benefit most people.

Chapter one is devoted to background information about Pike and this book. It is not really necessary to read this chapter to learn how to use and program Pike, but it might help explain why some things work the way they do. It might be more interesting to re-read the chapter after you have learned the basics of Pike programming. Chapter two is where the action starts. It is a crash course in Pike with examples and explanations of some of the basics. It explains the fundamentals of the Pike data types and control structures. The systematic documentation of all Pike capabilities starts in chapter three with a description of all control structures in Pike. It then continues with all the data types in chapter four and operators in chapter five. Chapter six deals with object orientation in Pike, which is slightly different than what you might be used to.

0.2 The history of Pike

In the beginning, there was Zork. Then a bunch of people decided to make multi-player adventure games. One of those people was Lars Pensjö at the Chalmers university in Gothenburg, Sweden. For his game he needed a simple, memory-efficient language, and thus LPC (Lars Pensjö C) was born. About a year later I started playing one of these games and found that the language was the most easy-to-use language I had ever encountered. I liked the language so much that I started improving it and before long I had made my own LPC dialect called LPC4. LPC4 was still geared towards writing adventure games, but was quite useful for writing other things with as well. A major problem with LPC4 was the copyright. Since it was based on Lars Pensjö’s code, it came with a license that did not allow it to be used for commercial gain. So, in 1994 I started writing µLPC, which was a new but similar LPC interpreter. I got financial backing from Signum Support AB for writing µLPC. Signum is a company dedicated to supporting GNU and GPL software and they wanted to create more GPL software.

When µLPC became usable, InformationsVävarna AB started using it for their web-server. Before then, Roxen (then called Spinner) was non-commercial and written in LPC4. Then in 1996 I started working for InformationsVävarna developing µLPC for them. We also changed the name of µLPC to Pike to get a more commercially viable name.
0.3. A COMPARISON WITH OTHER LANGUAGES

0.3 A comparison with other languages

0.4 What is Pike

Pike is:

•

•

•

•

•

•

Pike has:

•

•

•

•

•

•
0.5 How to read this manual

This manual uses a couple of different typefaces to describe different things:

Also, please beware that the word program is also a builtin Pike data type.
Chapter 1

Getting started

First you need to have Pike installed on your computer. See appendix D if this is not already done. It is also vital for the first of the following examples that the Pike binary is in your UNIX search path. If you have problems with this, consult the manual for your shell or go buy a beginners book about UNIX.

1.1 Your first 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, Let’s see what everything means:

int main()

This begins the function main. Before the function name the type of value it returns is declared, in this case int which is the name of the integer number type in Pike. The empty space between the parenthesis 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 a series of statements placed between curly brackets. Every statement has to end in a semicolon. This group of statements will be executed every time the function is called.

```
write("hello world\n");
```

The first statement is a call to the built-in function write. This will execute the code in the function write with the arguments as input data. 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.

### 1.2 Improving hello_world.pike

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()
{

``
1.3. FURTHER IMPROVEMENTS

```
write("hello world
");
return 0;
}
```

And then we tell UNIX that hello_world.pike is executable so we can run hello_world.pike without having to bother with running Pike:

```
$ chmod +x hello_world.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 name after the hash bang must also be the complete file name to the Pike binary, and it may not exceed 30 characters.

1.3 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 type the old hello world. We also 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 style
    }else{
        write("Hello world!\n"); // new style
    }
return 0;
}
```

Let’s run it:

```
$ chmod +x hello_world.pike
$ ./hello_world.pike
Hello world!
```
CHAPTE 1. GETTING STARTED

$ ./hello_world.pike --traditional
hello world
$

What is new in this version, then?

int main(int argc, array(string) argv)

In this version the space between the parenthesis has been filled. What it means
is that main now takes two arguments. One is called argc, and is of the type
int. The other is called argv and is 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 style
} else{
    write("Hello world!\n"); // new style
}

This is an if-else statement, it will execute what’s between the first set of
brackets if the expression between the parenthesis evaluate to something other
than zero. Otherwise what’s between the second set of brackets will be executed.
Let’s look at that expression:

argc > 1 && argv[1] == "--traditional"

Loosely translated, this means: argc is greater than one, and the second element
in the array argv is equal to the string --traditional. Since argc is the number
of words on the command line the first part is true only if there was anything
after the program invocation.

Also note the comments:

write("hello world\n"); // old style

The // begins a comment which continues to the end of the line. Comments
will be ignored by the computer when it reads the code. This allows 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 com-
ments, i.e. /* ... */, which can extend over several lines. The // comment
only extends to the end of the line.
1.4 Control structures

The first thing to understand about Pike is that just like any other programming language it executes one piece of code at a time. Most of the time it simply executes code line by line working its way downwards. Just executing a long list of instructions is not enough to make an interesting program however. Therefore we have control structures to make Pike execute pieces of code in more interesting orders than from top to bottom.

We have already seen an example of the if statement:

```pke
if( expression )
  statement1;
else
  statement2;
```

if simply evaluates the expression and if the result is true it executes statement1, otherwise it executes statement2. If you have no need for statement2 you can leave out the whole else part like this:

```pke
if( expression )
  statement1;
```

In this case statement1 is evaluated if expression is true, otherwise nothing is evaluated.

Note for beginners: go back to our first example and make sure you understand what if does.

Another very simple control structure is the while statement:

```pke
while( expression )
  statement;
```

This statement evaluates expression and if it is found to be true it evaluates statement. After that it starts over and evaluates expression again. This continues until expression is no longer true. This type of control structure is called a loop and is fundamental to all interesting programming.

1.5 Functions

Another control structure we have already seen is the function. A function is simply a block of Pike code that can be executed with different arguments from different places in the program. A function is declared like this:

```pke
modifiers type name(type varname1, type varname2, ...)
{
  statements
}
```
The modifiers are optional. See section 6.7 for more details about modifiers. The type specifies what kind of data the function returns. For example, the word \textbf{int} would signify that the function returns an integer number. The name is used to identify the function when calling it. The names between the parenthesis are the arguments to the function. They will be defined as local variables inside the function. Each variable will be declared to contain values of the preceding type. The three dots signifies that you can have anything from zero to 256 arguments to a function. The statements between the brackets are the function body. Those statements will be executed whenever the function is called.

Example:

\begin{verbatim}
int sqr(int x) { return x*x; }
\end{verbatim}

This line defines a function called \textbf{sqr} to take one argument of the type \textbf{int} and also returns an \textbf{int}. The code itself returns the argument multiplied by itself. To call this function from somewhere in the code you could simply put: \textbf{sqr(17)} and that would return the integer value 289.

As the example above shows, \textbf{return} is used to specify the return value of a function. The value after \textbf{return} must be of the type specified before the function name. If the function is specified to return \textbf{void}, nothing at all should be written after \textbf{return}. Note that when a return statement is executed, the function will finish immediately. Any statements following the return will be ignored.

There are many more control structures, they will all be described in a later chapter devoted only to control structures.

1.6 True and false

Throughout this chapter the words \textbf{true} and \textbf{false} have been used without any explanation to what they mean. Pike has a fairly simple way of looking at this. The number 0 is false and everything else is true. (Except when using operator overloading as I will explain in a later chapter.)

1.7 Data Types

As you saw in our first examples we have to indicate the type of value returned by a function or contained in a variable. We used integers (\textbf{int}), strings (\textbf{string}), and arrays (with the * notation). The others are \textbf{mapping, mixed, void, float, multiset, function, object and program}. Neither \textbf{mixed} nor \textbf{void} are really types, \textbf{void} signifies that no value should be returned and \textbf{mixed} that the return value can be of any type, or that the variable can contain any type of value. \textbf{Function, object} and \textbf{program} are all types related to object orientation. We will not discuss the last three in any great detail here,
1.7. DATA TYPES
Chapter 2

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.

```pike
#!/usr/local/bin/pike

mapping (string:array(string)) records =
{[
    "Star Wars Trilogy" : ({
        "Fox Fanfare",
        "Main Title",
        "Princess Leia's Theme",
        "Here They Come",
        "The Asteroid Field",
        "Yoda's Theme",
        "The Imperial March",
        "Parade of the 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.

```c
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.

```c
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 the called function is done the program just quits.

```c
int main(int argc, array (string) argv)
{
    if(argc <= 1)
    {
        list_records();
    } else {
        show_record((int) argv[1]);
    }
}
2.1 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".

### 2.1.1 add_record()

Using the method Stdio.ReadLine()\rightarrow read() we wait for input which will be put into the variable record\_name. The argument to \rightarrow read() 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) means "loop forever", because 1 is always true. This program does not in fact loop forever, because it uses return to exit the function from within the loop when you type a period. 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}).

```c
void add_record()
{
    string record_name=Stdio.ReadLine()\rightarrow read("Record name: ");
    records[record_name]=({});
    write("Input song names, one per line. End with '.' on its own line.\n");
    while(1)
    {
        string song;
        song=Stdio.ReadLine()\rightarrow read(sprintf("Song %2d: ",
                                                sizeof(records[record_name])+1));
        if(song==".")
            return;
        if (strlen(song))
            records[record_name]+=({song});
    }
}
```
2.1.2 main()

The main function now does not care about any command line arguments. Instead we use `Stdio.ReadLine()->read()` 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` 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 as in the first version of the program. If "list" receives an argument, that record is shown along with all the songs on it. If there is no argument it shows 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 described above. If the command given is "quit" the `exit(0)` statement stops the execution of the program and returns 0 (zero) to the operating system, telling it that everything was ok.

```c
int main(int argc, array(string) argv)
{
    string cmd;
    while(cmd=Stdio.ReadLine()->read(\"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;
        }
    }
}
```
2.2 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 will introduce you to programming with objects. To open a file for reading or writing we will use one of the programs which is builtin in Pike called Stdio.File. To Pike, a program is a data type which contains code, functions and variables. A program can be cloned which means that Pike creates a data area in memory for the program, places a reference to the program in the data area, and initializes it to act on the data in question. The methods (i.e. functions in the object) and variables in the object Stdio.File enable us to perform actions on the associated data file. The methods we need to use are open, read, write and close. See chapter 8.5 for more details.

2.2.1 save()

First we clone a Stdio.File program to the object o. Then we use it to open the file whose name is given in the string file_name for writing. We use the fact that if there is an error during opening, open() 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. If there is 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.

```pike
void save(string file_name)
{
    string name, song;
    Stdio.File o=Stdio.File();

    if(!o->open(file_name,"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();
}
```
2.2.2 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 absence of arguments to the method o->read means that the reading should not end until the end of the file. After having closed the file we initialize 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.

```cpp
void load(string file_name)
{
    string name="ERROR";
    string file_contents,line;

    Stdio.File o=Stdio.File();
    if(!o->open(file_name,"r"))
    {
        write("Failed to open file.
        return;
    }

    file_contents=o->read();
    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;
            }
        }
    }
}
```
2.2.3 main() revisited

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.

```c
    case "save":
        save(args);
        break;

    case "load":
        load(args);
        break;
```

2.3 Completing the program

Now let’s add the last functions we need to make this program useful: the ability to delete entries and search for songs.

2.3.1 delete()

If you sell one of your records it might be nice to be 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`.

```c
    void delete_record(int num)
    {
        array(string) record_names=sort(indices(records));
        string name=record_names[num-1];

        m_delete(records,name);
    }
```
2.3.2 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 initialize variables, that is done automatically when the variable is declared if you do not do it explicitly. To be able to use the built-in 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.

```c
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");
}
```

2.3.3 main() again

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.

```c
   case "delete":
       delete_record((int)args);
       break;

   case "search":
       find_song(args);
       break;
```
2.4 Then what?

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 left for you to do as you continue to read this book. The complete listing of this example can be found in appendix A. Read it, study it and enjoy!

2.5 Simple exercises

Make a program which writes hello world 10 times. Modify hello_world.pike to write the first argument to the program. Make a program that writes a hello_world program to stdout when executed. Modify the register program to store data about programs and diskettes instead of songs and records. Add code to the register program that checks that the user typed an argument when required. The program should notify the user and wait to receive more commands instead of exiting with an error message. Add code to the register program to check that the arguments to show_record and delete_records are numbers. Also make sure that the number isn’t less than one or bigger than the available number of records. Rewrite the register program and put all the code in main().
Chapter 3

Control Structures

In this chapter all the control structures in Pike will be explained. As mentioned earlier, control structures are used to control the flow of the program execution. Note that functions that make the program pause and simple function calls are not qualified as control structures.

3.1 Conditions

Pike only has two major condition control structures. We have already seen examples of both of them in Chapter two. But for completeness they will be described again in this chapter.

3.1.1 if

The simplest one is called the if statement. It can be written anywhere where a statement is expected and it looks like this: if( expression ) statement1; else statement2; Please note that there is no semicolon after the parenthesis or after the else. Step by step, if does the following:

1. 
2. 
3. 
4. 
5. 
6. 

This is actually more or less how the interpreter executes the if statement. In short, statement1 is executed if expression is true otherwise statement2 is executed. If you are interested in having something executed if the expression is false you can drop the whole else part like this:
if( expression )
  statement1;

If on the other hand you are not interested in evaluating something if the expression is false you should use the not operator to negate the true/false value of the expression. See chapter 5 for more information about the not operator. It would look like this:

if( ! expression )
  statement2;

Any of the statements here and in the rest of this chapter can also be a block of statements. A block is a list of statements, separated by semicolons and enclosed by brackets. Note that you should never put a semicolon after a block of statements. The example above would look like this:

if( ! expression )
{
  statement;
  statement1;
  statement2;
}

3.1.2 switch

A more sophisticated condition control structure is the switch statement. A switch lets you select one of many choices depending on the value of an expression and it can look something like this:

switch ( expression )
{
  case constant1:
    statement1;
    break;
  case constant2:
    statement2;
    break;
  case constant3 .. constant4:
    statement3;
    break;
  default:
    statement5;
}
As you can see, a switch statement is a bit more complicated than an if statement. It is still fairly simple however. It starts by evaluating the expression it then searches all the case statements in the following block. If one is found to be equal to the value returned by the expression, Pike will continue executing the code directly following that case statement. When a break is encountered Pike will skip the rest of the code in the switch block and continue executing after the block. Note that it is not strictly necessary to have a break before the next case statement. If there is no break before the next case statement Pike will simply continue executing and execute the code after that case statement as well.

One of the case statements in the above example differs in that it is a range. In this case, any value between constant3 and constant4 will cause Pike to jump to statement3. Note that the ranges are inclusive, so the values constant3 and constant4 are also valid.

### 3.2 Loops

Loops are used to execute a piece of code more than once. Since this can be done in quite a few different ways there are four different loop control structures. They may all seem very similar, but using the right one at the right time makes the code a lot shorter and simpler.

#### 3.2.1 while

While is the simplest of the loop control structures. It looks just like an if statement without the else part:

```c
while ( expression )
    statement;
```

The difference in how it works isn’t that big either, the statement is executed if the expression is true. Then the expression is evaluated again, and if it is true the statement is executed again. Then it evaluates the expression again and so forth... Here is an example of how it could be used:

```c
int e=1;
while(e<5)
{
    show_record(e);
    e=e+1;
}
```

This would call show_record with the values 1, 2, 3 and 4.
3.2.2 for

For is simply an extension of while. It provides an even shorter and more compact way of writing loops. The syntax looks like this:

\[
\text{for ( } \text{initializer} \text{, } \text{expression} \text{, } \text{incrementor} \text{) }
\]
\[
\text{statement ;}
\]

For does the following steps:

1. 
2. 
3. 
4. 
5. 
6. 

This means that the example in the while section can be written like this:

\[
\text{for(int } e=1; e<5; e=e+1) \\
\text{ show_record(e);}
\]

3.2.3 do-while

Sometimes it is unpractical that the expression is always evaluated before the first time the loop is executed. Quite often you want to execute something, and then do it over and over until some condition is satisfied. This is exactly when you should use the do-while statement.

\[
\text{do} \\
\text{ statement ;} \\
\text{while ( } \text{expression} \text{) ;}
\]

As usual, the statement can also be a block of statements, and then you do not need a semicolon after it. To clarify, this statement executes statement first, and then evaluates the expression. If the expression is true it executes the loop again. For instance, if you want to make a program that lets your modem dial your Internet provider, it could look something like this:

\[
\text{do } \\
\text{ modem->write("ATDT441-9109\n"); // Dial 441-9109 } \\
\text{ while(modem->gets()[..6]} \text{ != "CONNECT"};}
\]

This example assumes you have written something that can communicate with the modem by using the functions write and gets.
3.3. BREAKING OUT OF LOOPS

3.2.4 foreach

Foreach is unique in that it does not have an explicit test expression evaluated for each iteration in the loop. Instead, foreach executes the statement once for each element in an array. Foreach looks like this:

```
foreach ( array_expression, variable )
  statement ;
```

We have already seen an example of foreach in the find_song function in chapter 2. What foreach does is:

1.
2.
3.
4.
5.
6.

Foreach is not really necessary, but it is faster and clearer than doing the same thing with a for loop, as shown here:

```
array tmp1= array_expression;
for ( tmp2 = 0; tmp2 < sizeof(tmp1); tmp2++)
{
  variable = tmp1 [ tmp2 ];
  statement;
}
```

3.3 Breaking out of loops

The loop control structures above are enough to solve any problem, but they are not enough to provide an easy solution to all problems. One thing that is still missing is the ability to exit a loop in the middle of it. There are three ways to do this:

3.3.1 break

break exits a loop or switch statement immediately and continues executing after the loop. Break can not be used outside of a loop or switch. It is quite useful in conjunction with while(1) to construct command parsing loops for instance:

```
while(1)
```
{  
    string command=Stdio.ReadLine()->read(">");  
    if(command=="quit") break;  
    do_command(command);  
}

### 3.3.2 continue

continue does almost the same thing as break, except instead of breaking out of the loop it only breaks out of the loop body. It then continues to execute the next iteration in the loop. For a while loop, this means it jumps up to the top again. For a for loop, it jumps to the incrementor expression. For a do-while loop it jumps down to the expression at the end. To continue our example above, continue can be used like this:

```plaintext
while(1)  
{  
    string command=Stdio.ReadLine()->read(">");  
    if(strlen(command) == 0) continue;  
    if(command=="quit") break;  
    do_command(command);  
}
```

This way, do_command will never be called with an empty string as argument.

### 3.3.3 return

Return doesn’t just exit the loop, it exits the whole function. We have seen several examples how to use it chapter 2. None of the functions in chapter two returned anything in particular however. To do that you just put the return value right after return. Of course the type of the return value must match the type in the function declaration. If your function declaration is int main() the value after return must be an int. For instance, if we wanted to make a program that always returns an error code to the system, just like the UNIX command false this is how it would be done:

```plaintext
#!/usr/local/bin/pike

int main()  
{  
    return 1;  
}
```

This would return the error code 1 to the system when the program is run.
3.4 Exercises

End all functions in the examples in chapter two with a return statement. Change all foreach loops to for or while loops. Make the find_song function in chapter 2 return when the first matching song is found. Make the find_song function write the number of the record the song is on. If you failed to get the program to work properly in the last exercise of chapter 2, try it again now. Make a program that writes all the numbers from 1 to 1000. Modify the program in the previous exercise to NOT write numbers divisible by 3, 7 or 17. Make a program that writes all the prime numbers between 1 and 1000.
Chapter 4

Data types

In this chapter we will discuss all the different ways to store data in Pike in detail. We have seen examples of many of these, but we haven’t really gone into how they work. In this chapter we will also see which operators and functions work with the different types. There are two categories of data types in Pike: basic types, and pointer types. The difference is that basic types are copied when assigned to a variable. With pointer types, merely the pointer is copied, that way you get two variables pointing to the same thing.

4.1 Basic types

The basic types are int, float and string. For you who are accustomed to C or C++, it may seem odd that a string is a basic type as opposed to an array of char, but it is surprisingly easy to get used to.

4.1.1 int

Int is short for integer, or integer number. They are normally 32 bit integers, which means that they are in the range -2147483648 to 2147483647. Note that on some machines an int might be larger than 32 bits. Since they are integers, no decimals are allowed. An integer constant can be written in several ways:

78 // decimal number
0116 // octal number
0x4e // hexadecimal number
’N’ // Ascii character

All of the above represent the number 78. Octal notation means that each digit is worth 8 times as much as the one after. Hexadecimal notation means that each digit is worth 16 times as much as the one after. Hexadecimal notation uses the letters a, b, c, d, e and f to represent the numbers 10, 11, 12, 13, 14 and 15. The ASCII notation gives the ASCII value of the character between
the single quotes. In this case the character is \& which just happens to be 78 in ASCII.

Integers are coded in 2-complement and overflows are silently ignored by Pike. This means that if your integers are 32-bit and you add 1 to the number 2147483647 you get the number -2147483648. This works exactly as in C or C++.

All the arithmetic, bitwise and comparison operators can be used on integers. Also note these functions:

4.1.2 float

Although most programs only use integers, they are unpractical when doing trigonometric calculations, transformations or anything else where you need decimals. For this purpose you use float. Floats are normally 32 bit floating point numbers, which means that they can represent very large and very small numbers, but only with 9 accurate digits. To write a floating point constant, you just put in the decimals or write it in the exponential form:

3.14159265358979323846264338327950288419716939937510 // Pi
1.0e9 // A billion
1.0e-9 // A billionth

Of course you do not need this many decimals, but it doesn’t hurt either. Usually digits after the ninth digit are ignored, but on some architectures float might have higher accuracy than that. In the exponential form, e means "times 10 to the power of", so 1.0e9 is equal to "1.0 times 10 to the power of 9".

All the arithmetic and comparison operators can be used on floats. Also, these functions operates on floats:
4.1.3 string

A string can be seen as an array of values from 0 to $2^{32}$-1. Usually a string contains text such as a word, a sentence, a page or even a whole book. But it can also contain parts of a binary file, compressed data or other binary data. Strings in Pike are shared, which means that identical strings share the same memory space. This reduces memory usage very much for most applications and also speeds up string comparisons. We have already seen how to write a constant string:

"hello world" // hello world
"he" "llo" // hello
"\116" // N (116 is the octal ASCII value for N)
"\t" // A tab character
"\n" // A newline character
"\r" // A carriage return character
"\b" // A backspace character
"\0" // A null character
"\"" // A double quote character
"\"" // A single backslash
"\x4e" // N (4e is the hexadecimal ASCII value for N)
"\d78" // N (78 is the decimal ASCII value for N)
"hello world\116\t\n\r\b\0\"" // All of the above
"\xff" // the character 255
"\xffff" // the character 65536
"\x7fffffff" // the character 16777215
"\116""3" // 'N' followed by a '3'
As you can see, any sequence of characters within double quotes is a string. The backslash character is used to escape characters that are not allowed or impossible to type. As you can see, \t is the sequence to produce a tab character, \n is used when you want one backslash and \" is used when you want a double quote (") to be a part of the string instead of ending it. Also, \XXX where XXX is an octal number from 0 to 37777777777 or \xXX where XX is 0 to ffffffff lets you write any character you want in the string, even null characters. From version 0.6.105, you may also use \dXXX where XXX is 0 to 2^32-1. If you write two constant strings after each other, they will be concatenated into one string.

You might be surprised to see that individual characters can have values up to 2^32-1 and wonder how much memory that use. Do not worry, Pike automatically decides the proper amount of memory for a string, so all strings with character values in the range 0-255 will be stored with one byte per character. You should also beware that not all functions can handle strings which are not stored as one byte per character, so there are some limits to when this feature can be used.

Although strings are a form of arrays, they are immutable. This means that there is no way to change an individual character within a string without creating a new string. This may seem strange, but keep in mind that strings are shared, so if you would change a character in the string "foo", you would change *all* "foo" everywhere in the program. However, the Pike compiler will allow you to write code like you could change characters within strings, the following code is valid and works:

```
string s="hello world";
s[6] = 'w';
```

However, you should be aware that this does in fact create a new string and it may need to copy the string s to do so. This means that the above operation can be quite slow for large strings. You have been warned. Most of the time, you can use replace, sscanf, /or some other high-level string operation to avoid having to use the above construction too much.

All the comparison operators plus the operators listed here can be used on strings:
Also, these functions operates on strings:
4.2 Pointer types

The basic types are, as the name implies, very basic. They are the foundation, most of the pointer types are merely interesting ways to store the basic types. The pointer types are array, mapping, multiset, program, object and function. They are all pointers which means that they point to something in memory. This "something" is freed when there are no more pointers to it. Assigning a variable with a value of a pointer type will not copy this "something" instead it will only generate a new reference to it. Special care sometimes has to be taken when giving one of these types as arguments to a function; the function can in fact modify the "something". If this effect is not wanted you have to explicitly copy the value. More about this will be explained later in this chapter.

4.2.1 array

Arrays are the simplest of the pointer types. An array is merely a block of memory with a fixed size containing a number of slots which can hold any type of value. These slots are called elements and are accessible through the index operator. To write a constant array you enclose the values you want in the array with ({} ) like this:

({}) // Empty array
({1}) // Array containing one element of type int
({""}) // Array containing a string
({"", 1, 3.0}) // Array of three elements, each of different type

As you can see, each element in the array can contain any type of value. Indexing and ranges on arrays works just like on strings, except with arrays you can change values inside the array with the index operator. However, there is no way to change the size of the array, so if you want to append values to the end you still have to add it to another array which creates a new array. Figure 4.1 shows how the schematics of an array. As you can see, it is a very simple memory structure.

![fig 4.1](image-url)
4.2. POINTER TYPES

Operators and functions usable with arrays:
4.2.2 mapping

Mappings are really just more generic arrays. However, they are slower and use more memory than arrays, so they cannot replace arrays completely. What makes mappings special is that they can be indexed on other things than integers. We can imagine that a mapping looks like this:

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Value</td>
</tr>
<tr>
<td>Index</td>
<td>Value</td>
</tr>
<tr>
<td>Index</td>
<td>Value</td>
</tr>
<tr>
<td>Index</td>
<td>Value</td>
</tr>
<tr>
<td>Index</td>
<td>Value</td>
</tr>
</tbody>
</table>

Mapping

```
Lookup Function
```
4.2. POINTER TYPES

Each index-value pair is floating around freely inside the mapping. There is exactly one value for each index. We also have a (magical) lookup function. This lookup function can find any index in the mapping very quickly. Now, if the mapping is called \( m \) and we index it like this: \( m[i] \) the lookup function will quickly find the index \( i \) in the mapping and return the corresponding value. If the index is not found, zero is returned instead. If we on the other hand assign an index in the mapping the value will instead be overwritten with the new value. If the index is not found when assigning, a new index-value pair will be added to the mapping. Writing a constant mapping is easy:

\[
(\{\}) \quad // \text{Empty mapping} \\
(\{1:2\}) \quad // \text{Mapping with one index-value pair, the 1 is the index} \\
(\{"one":1, "two":2\}) \quad // \text{Mapping which maps words to numbers} \\
(\{1:{2.0}, "":[\], \}) \quad // \text{Mapping with lots of different types}
\]

As with arrays, mappings can contain any type. The main difference is that the index can be any type too. Also note that the index-value pairs in a mapping are not stored in a specific order. You can not refer to the fourteenth key-index pair, since there is no way of telling which one is the fourteenth. Because of this, you cannot use the range operator on mappings.

The following operators and functions are important:
4.2.3 multiset

A multiset is almost the same thing as a mapping. The difference is that there are no values:

Instead, the index operator will return 1 if the value was found in the multiset and 0 if it was not. When assigning an index to a multiset like this: \texttt{mset[ ind ] = val} the index \texttt{ind} will be added to the multiset \texttt{mset} if \texttt{val} is \texttt{true}. Otherwise \texttt{ind} will be removed from the multiset instead.

Writing a constant multiset is similar to writing an array:

\begin{verbatim}
(< >) // Empty multiset
(< 17 >) // Multiset with one index: 17
(< "", 1, 3.0, 1 >) // Multiset with 3 indices
\end{verbatim}
Note that you can actually have more than one of the same index in a multiset. This is normally not used, but can be practical at times.

4.2.4  program

Normally, when we say program we mean something we can execute from a shell prompt. However, Pike has another meaning for the same word. In Pike a program is the same as a class in C++. A program holds a table of what functions and variables are defined in that program. It also holds the code itself, debug information and references to other programs in the form of inherits. A program does not hold space to store any data however. All the information in a program is gathered when a file or string is run through the Pike compiler. The variable space needed to execute the code in the program is stored in an object which is the next data type we will discuss.

Writing a program is easy, in fact, every example we have tried so far has been a program. To load such a program into memory, we can use compile_file
which takes a file name, compiles the file and returns the compiled program. It could look something like this:

```pike
program p = compile_file("hello_world.pike");
```

You can also use the `cast` operator like this:

```pike
program p = (program) "hello_world";
```

This will also load the program `hello_world.pike`, the only difference is that it will cache the result so that next time you do `(program)"hello_world"` you will receive the _same_ program. If you call `compile_file("hello_world.pike")` repeatedly you will get a new program each time.

There is also a way to write programs inside programs with the help of the `class` keyword:

```pike
class class_name {
    inherits, variables and functions
}
```

The `class` keyword can be written as a separate entity outside of all functions, but it is also an expression which returns the _program_ written between the brackets. The `class_name` is optional. If used you can later refer to that _program_ by the name `class_name`. This is very similar to how classes are written in C++ and can be used in much the same way. It can also be used to create _structs_ (or _records_ if you program Pascal). Let’s look at an example:

```pike
class record {
    string title;
    string artist;
    array(string) songs;
}

array(record) records = ({});

void add_empty_record()
{
    records+=({ record() });
}

void show_record(record rec)
{
    write("Record name: "+rec->title+"\n");
    write("Artist: "+rec->artist+"\n");
    write("Songs: \n");
    foreach(rec->songs, string song)
        write(" "+song+"\n");
}
```
This could be a small part of a better record register program. It is not a complete executable program in itself. In this example we create a program called record which has three identifiers. In add_empty_record a new object is created by calling record. This is called cloning and it allocates space to store the variables defined in the class record. Show_record takes one of the records created in add_empty_record and shows the contents of it. As you can see, the arrow operator is used to access the data allocated in add_empty_record. If you do not understand this section I suggest you go on and read the next section about objects and then come back and read this section again.

The following operators and functions are important:

4.2.5 object

Although programs are absolutely necessary for any application you might want to write, they are not enough. A program doesn’t have anywhere to store data, it just merely outlines how to store data. To actually store the data you need an object. Objects are basically a chunk of memory with a reference to the program from which it was cloned. Many objects can be made from
one program. The program outlines where in the object different variables are stored.

Each object has its own set of variables, and when calling a function in that object, that function will operate on those variables. If we take a look at the short example in the section about programs, we see that it would be better to write it like this:

```java
class record {
    string title;
    string artist;
    array(string) songs;

    void show() {
        write("Record name: "+title+"\n");
        write("Artist: "+artist+"\n");
        write("Songs:\n");
        foreach(songs, string song)
            write("  "+song+"\n");
    }
}
```
array(record) records = ({});

void add_empty_record()
{
    records+=({ record() });
}

void show_record(object rec)
{
    rec->show();
}

Here we can clearly see how the function show prints the contents of the variables in that object. In essence, instead of accessing the data in the object with the -> operator, we call a function in the object and have it write the information itself. This type of programming is very flexible, since we can later change how record stores its data, but we do not have to change anything outside of the record program.

Functions and operators relevant to objects:
4.2.6 function

When indexing an object on a string, and that string is the name of a function in the object a function is returned. Despite its name, a function is really a function pointer.

When the function pointer is called, the interpreter sets this_object() to the object in which the function is located and proceeds to execute the function it points to. Also note that function pointers can be passed around just like any other data type:

```c
int foo() { return 1; }
function bar() { return foo; }
int gazonk() { return foo(); }
int teleledningsanka() { return bar()(); }
```

In this example, the function bar returns a pointer to the function foo. No indexing is necessary since the function foo is located in the same object. The function gazonk simply calls foo. However, note that the word foo in that function is an expression returning a function pointer that is then called. To further illustrate this, foo has been replaced by bar() in the function teleledningsanka.

For convenience, there is also a simple way to write a function inside another function. To do this you use the lambda keyword. The syntax is the same as for a normal function, except you write lambda instead of the function name:

```c
lambda ( types ) { statements }
```
4.3 SHARING DATA

The major difference is that this is an expression that can be used inside an other function. Example:

```pke
function bar() { return lambda() { return 1; }; }
```

This is the same as the first two lines in the previous example, the keyword `lambda` allows you to write the function inside `bar`.

Note that unlike C++ and Java you can not use function overloading in Pike. This means that you cannot have one function called 'foo' which takes an integer argument and another function 'foo' which takes a float argument.

This is what you can do with a function pointer.

4.3 Sharing data

As mentioned in the beginning of this chapter, the assignment operator (=) does not copy anything when you use it on a pointer type. Instead it just creates another reference to the memory object. In most situations this does not present a problem, and it speeds up Pike’s performance. However, you must be aware of this when programming. This can be illustrated with an example:

```pke
int main(int argc, array(string) argv)
{
    array(string) tmp;
    tmp=argv;
    argv[0]="Hello world.\n";
    write(tmp[0]);
}
```

This program will of course write `Hello world`. 

Sometimes you want to create a copy of a mapping, array or object. To do so you simply call `copy_value` with whatever you want to copy as argument. Copy_value is recursive, which means that if you have an array containing arrays, copies will be made of all those arrays.

If you don’t want to copy recursively, or you know you don’t have to copy recursively, you can use the plus operator instead. For instance, to create a copy of an array you simply add an empty array to it, like this: `copy_of_arr = arr + ([]);` If you need to copy a mapping you use an empty mapping, and for a multiset you use an empty multiset.

### 4.4 Writing data types

When declaring a variable, you also have to specify what type of variable it is. For most types, such as `int` and `string` this is very easy. But there are much more interesting ways to declare variables than that, let’s look at a few examples:

```plaintext
int x; // x is an integer
int|string x; // x is a string or an integer
array(string) x; // x is an array of strings
array x; // x is an array of mixed
mixed x; // x can be any type
string *x; // x is an array of strings

// x is a mapping from int to string
mapping(string:int) x;

// x implements Stdio.File
Stdio.File x;

// x implements Stdio.File
object(Stdio.File) x;

// x is a function that takes two integer
// arguments and returns a string
function(int,int:string) x;

// x is a function taking any amount of
// integer arguments and returns nothing.
function(int...:void) x;

// x is ... complicated
mapping(string:function(string|int...:mapping(string|array(string)))) x;
```

As you can see there are some interesting ways to specify types. Here is a list of what is possible:
4.4. WRITING DATA TYPES
Chapter 5

Operators

To make it easier to write Pike, and to make the code somewhat shorter, some functions can be called with just one or two characters in the code. These functions are called operators and we have already seen how they work in plenty of examples. In this chapter I will describe in detail what they do. The operators are divided into categories depending on their function, but beware that some operators have meanings that go way beyond the scope of the category they are in.

5.1 Arithmetic operators

The arithmetic operators are the simplest ones, since they work just like you remember from math in school. The arithmetic operators are:

<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Identifier</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>a + b</td>
<td>'+'</td>
<td>the sum of a and b</td>
</tr>
<tr>
<td>Subtraction</td>
<td>a - b</td>
<td>'-'</td>
<td>b subtracted from a</td>
</tr>
<tr>
<td>Negation</td>
<td>- a</td>
<td>'-'</td>
<td>minus a</td>
</tr>
<tr>
<td>Multiplication</td>
<td>a * b</td>
<td>'*'</td>
<td>a multiplied by b</td>
</tr>
<tr>
<td>Division</td>
<td>a / b</td>
<td>'/'</td>
<td>a divided by b</td>
</tr>
<tr>
<td>Modulo</td>
<td>a % b</td>
<td>'%'</td>
<td>the remainder of a division between a and b</td>
</tr>
</tbody>
</table>

The third column, "Identifier" is the name of the function that actually evaluates the operation. For instance, a + b can also be written as '+'(a, b). I will show you how useful this can be at the end of this chapter.

When applied to integers or floats these operators do exactly what they are supposed to do. The only operator in the list not known from basic math is the modulo operator. The modulo operator returns the remainder from an integer division. It is the same as calculating a - floor(a / b) * b. floor rounds the value down to closest lower integer value. Note that the call to floor isn’t
needed when operating on integers, since dividing two integers will return the result as an integer and it is always rounded down. For instance, 8 / 3 would return 2.

If all arguments to the operator are integers, the result will also be an integer. If one is a float and the other is an integer, the result will be a float. If both arguments are float, the result will of course be a float.

However, there are more types in Pike than integers and floats. Here is the complete list of combinations of types you can use with these operators:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Returned type</th>
<th>Returned value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int + int</td>
<td>int</td>
<td>the sum of the two values</td>
</tr>
<tr>
<td>float + int</td>
<td>float</td>
<td>the sum of the two values</td>
</tr>
<tr>
<td>float + float</td>
<td>float</td>
<td>the sum of the two values</td>
</tr>
<tr>
<td>string + string</td>
<td>string</td>
<td>In this case, any int or float is first converted to a string. Then the two strings are concatenated and the resulting string is returned.</td>
</tr>
<tr>
<td>int + string</td>
<td>string</td>
<td>A copy of the left string with all occurrences of the right string removed.</td>
</tr>
<tr>
<td>float + string</td>
<td>string</td>
<td>A copy of the left string with all elements present in the right array removed. Example: {(2,1,4,5,3,6,7)} - {(3,5,1)} will return {(2,4,6,7)}.</td>
</tr>
<tr>
<td>string + int</td>
<td>string</td>
<td>A new mapping with all index-value pairs from the left mapping, except those indices that are also present in the right mapping.</td>
</tr>
<tr>
<td>int - int</td>
<td>int</td>
<td>The right value subtracted from the left.</td>
</tr>
<tr>
<td>float - int</td>
<td>float</td>
<td>The right value subtracted from the left.</td>
</tr>
<tr>
<td>string - string</td>
<td>string</td>
<td>A copy of the left string with all occurrences of the right string removed.</td>
</tr>
<tr>
<td>array - array</td>
<td>array</td>
<td>A mapping with all the index-value pairs from both mappings is returned. If an index is present in both mappings the index-value pair from the right mapping will be used.</td>
</tr>
<tr>
<td>mapping - mapping</td>
<td>mapping</td>
<td>A new mapping without any index present in the left multiset.</td>
</tr>
<tr>
<td>int - int</td>
<td>int</td>
<td>Same as 0 - int.</td>
</tr>
<tr>
<td>float - float</td>
<td>float</td>
<td>Same as 0 - float.</td>
</tr>
<tr>
<td>int * int</td>
<td>int</td>
<td>the product of the two values</td>
</tr>
<tr>
<td>float * int</td>
<td>float</td>
<td>the product of the two values</td>
</tr>
<tr>
<td>array * array</td>
<td>array</td>
<td>A new array with all elements present in the right array removed. Example: {2,1,4,5,3,6,7} * {3,5,1} will return {2,4,6,7}.</td>
</tr>
<tr>
<td>mapping * mapping</td>
<td>mapping</td>
<td>A new mapping with all index-value pairs from the left mapping, except those indices that are also present in the right mapping.</td>
</tr>
<tr>
<td>int - multiset</td>
<td>multiset</td>
<td>A multiset with all the indices from both multisets is returned.</td>
</tr>
<tr>
<td>float - multiset</td>
<td>multiset</td>
<td>A multiset with all the indices from both multisets is returned.</td>
</tr>
</tbody>
</table>
5.1. ARITHMETIC OPERATORS

<table>
<thead>
<tr>
<th>Operation</th>
<th>Returned type</th>
<th>Returned value</th>
</tr>
</thead>
<tbody>
<tr>
<td>array(string)*string</td>
<td>string</td>
<td>All the strings in the array are concatenated with the string on the right in between each string. Example: <code>{{&quot;foo&quot;,&quot;bar&quot;}}*&quot;-&quot;</code> will return &quot;foo-bar&quot;.</td>
</tr>
<tr>
<td>array(array)*array</td>
<td>array</td>
<td>All the arrays in the left array are concatenated with the array on the right in between each array. Example: <code>{{{&quot;foo&quot;}}},{{&quot;bar&quot;}}}*{{&quot;-&quot;}}</code> will return <code>{{ &quot;foo&quot;,&quot;-&quot;,&quot;bar&quot; }}</code>.</td>
</tr>
<tr>
<td>string*int</td>
<td>string</td>
<td>This operation will concatenate the string N times. Example: &quot;foo&quot;*3 will return &quot;foofoofoo&quot;.</td>
</tr>
<tr>
<td>array*int</td>
<td>string</td>
<td>This operation will concatenate the array N times. Example: <code>{{&quot;foo&quot;}}*3</code> will return <code>{{&quot;foo&quot;,&quot;foo&quot;,&quot;foo&quot;}}</code>.</td>
</tr>
<tr>
<td>int/int</td>
<td>int</td>
<td>The right integer divided by the left integer rounded towards minus infinity.</td>
</tr>
<tr>
<td>float/int</td>
<td>float</td>
<td>The right value divided by the left value.</td>
</tr>
<tr>
<td>int/float</td>
<td>array(string)</td>
<td>In symmetry with the multiplication operator, the division operator can split a string into pieces. The right string will be split at every occurrence of the right string and an array containing the results will be returned. Example: &quot;foo-bar&quot;/&quot;-&quot; will return <code>{{&quot;foo&quot;,&quot;bar&quot;}}</code>.</td>
</tr>
<tr>
<td>string/float</td>
<td>array(string)</td>
<td>This will split the string into pieces. The size of the pieces is given by the integer. Only complete pieces will be included in the result, the 'reminder' is discarded. Example: &quot;foo-bar&quot;/2 will return <code>{{&quot;fo&quot;,&quot;o-&quot;,&quot;ba&quot;}}</code>.</td>
</tr>
<tr>
<td>string/float</td>
<td>array(string)</td>
<td>This is similar to dividing a string with an integer, but it allows fraction-sized segments and the reminder will always be included. Example: &quot;foo-bar&quot;/2.5 will return <code>{{&quot;fo&quot;,&quot;o-b&quot;,&quot;ar&quot;}}</code>.</td>
</tr>
<tr>
<td>array/float</td>
<td>array(string)</td>
<td>This is similar to dividing a string with an integer, but splits an array. Example: <code>(1,2,3,4,5,6,7)/2</code> will return <code>{{(1,2)},{(3,4)},{(5,6)}}</code>.</td>
</tr>
<tr>
<td>array/int</td>
<td>array(array)</td>
<td>You should be able to predict what this does by now. :) Example: <code>(1,2,3,4,5,6,7,8)/2.5</code> will return <code>{{(1,2)},{(3,4,5)},{(6,7)},{(8)}</code>.</td>
</tr>
<tr>
<td>int/int</td>
<td>int</td>
<td>The remainder of a division. If a and b are integers, a%b is the same as a-(a/b)*b.</td>
</tr>
<tr>
<td>float/float</td>
<td>float</td>
<td>The remainder of a division. If a and b are floats, a%b is the same as a-floor(a/b)*b.</td>
</tr>
</tbody>
</table>
### 5.2 Comparison operators

The arithmetic operators would be hard to use for anything interesting without the ability to compare the results to each other. For this purpose there are six comparison operators:

<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Identifier</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same</td>
<td>a == b</td>
<td><code>==</code></td>
<td>1 if a is the same value as b, 0 otherwise</td>
</tr>
<tr>
<td>Not same</td>
<td>a != b</td>
<td><code>!=</code></td>
<td>0 if a is the same value as b, 1 otherwise</td>
</tr>
<tr>
<td>Greater than</td>
<td>a &gt; b</td>
<td><code>&gt;</code></td>
<td>1 if a is greater than b, 0 otherwise</td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td>a &gt;= b</td>
<td><code>&gt;=</code></td>
<td>1 if a is greater to or equal to b, 0 otherwise</td>
</tr>
<tr>
<td>Lesser than</td>
<td>a &lt; b</td>
<td><code>&lt;</code></td>
<td>1 if a is lesser than b, 0 otherwise</td>
</tr>
<tr>
<td>Lesser than or equal to</td>
<td>a &lt;= b</td>
<td><code>&lt;=</code></td>
<td>1 if a is lesser than or equal to b, 0 otherwise</td>
</tr>
</tbody>
</table>

The `==` and `!=` operators can be used on any type. For two values to be same they must be the same type. Therefore 1 and 1.0 are not same. Also, for two values of pointer types to be the same the two values must be pointers to the same object. It is not enough that the two objects are of the same size and contain the same data.

The other operators in the table above can only be used with integers, floats and strings. If you compare an integer with a float, the int will be promoted to a float before the comparison. When comparing strings, lexical order is used and the values of the environment variables `LC_CTYPE` and `LC_LANG` are respected.

### 5.3 Logical operators

Logical operators are operators that operate with truth values. In Pike any value except zero is considered true. Logical operators are a very basic part of Pike. They can also decide which arguments to evaluate and which not to evaluate. Because of this the logical operators do not have any identifiers and can not be called as normal functions. There are four logical operators:


5.4. BITWISE/SET OPERATORS

<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Identifier</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>a &amp;&amp; b</td>
<td>'&amp;&amp;'</td>
<td>If a is false, a is returned and b is not evaluated. Otherwise, b is returned.</td>
</tr>
<tr>
<td>OR</td>
<td>a</td>
<td></td>
<td>b</td>
</tr>
<tr>
<td>NOT</td>
<td>! a</td>
<td>'~'</td>
<td>Returns 0 if a is true, 1 otherwise.</td>
</tr>
<tr>
<td>IF-ELSE</td>
<td>a ? b : c</td>
<td>'?:'</td>
<td>If a is true, b is returned and c is not evaluated. Otherwise c is returned and b is not evaluated.</td>
</tr>
</tbody>
</table>

5.4 Bitwise/set operators

These operators are used to manipulate bits as members in sets. They can also manipulate arrays, multisets and mappings as sets.

<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Identifier</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift left</td>
<td>a &lt;&lt; b</td>
<td>'&lt;&lt;'</td>
<td>Multiplies a by 2 b times.</td>
</tr>
<tr>
<td>Shift right</td>
<td>a &gt;&gt; b</td>
<td>'&gt;&gt;'</td>
<td>Divides a by 2 b times.</td>
</tr>
<tr>
<td>Inverse (not)</td>
<td>~ a</td>
<td>'~'</td>
<td>Returns -1-a.</td>
</tr>
<tr>
<td>Intersection (and)</td>
<td>a &amp; b</td>
<td>'&amp;'</td>
<td>All elements present in both a and b.</td>
</tr>
<tr>
<td>Union (or)</td>
<td>a</td>
<td>b</td>
<td>'</td>
</tr>
<tr>
<td>Symmetric difference (xor)</td>
<td>a ^ b</td>
<td>'^'</td>
<td>All elements present in a or b, but not present in both.</td>
</tr>
</tbody>
</table>

The first three operators can only be used with integers and should be pretty obvious. The other three, intersection, union and symmetric difference, can be used with integers, arrays, multisets and mappings. When used with integers, these operators considers each bit in the integer a separate element. If you do not know about how bits in integers work I suggest you go look it up in some other programming book or just don’t use these operators on integers.

When intersection, union or symmetric difference is used on an array each element in the array is considered by itself. So intersecting two arrays will result in an array with all elements that are present in both arrays. Example: `(7,6,4,3,2,1) & (1, 23, 5, 4, 7)` will return `(7,4,1)`. The order of the elements in the returned array will always be taken from the left array. Elements in multisets are treated the same as elements in arrays. When doing a set operation on a mapping however, only the indices are considered. The values are just copied with the indices. If a particular index is present in both the right and left argument to a set operator, the one from the right side will be used. Example: `(1:2) | (1:3)` will return `(1:3)`.
5.5 Indexing

The index and range operators are used to retrieve information from a complex data type.

<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Identifier</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>a [ b ]</td>
<td>[]</td>
<td>Returns the index b from a.</td>
</tr>
<tr>
<td>Look up</td>
<td>a \rightarrow identifier</td>
<td>\rightarrow</td>
<td>Looks up the identifier. Same as a[&quot;identifier&quot;].</td>
</tr>
<tr>
<td>Assign index</td>
<td>a [ b ] = c</td>
<td>[=;</td>
<td>Sets the index b in a to c.</td>
</tr>
<tr>
<td>Assign index</td>
<td>a \rightarrow identifier = c</td>
<td>\rightarrow=</td>
<td>Sets the index &quot;identifier&quot; in a to c.</td>
</tr>
<tr>
<td>Range</td>
<td>a [ b .. c ]</td>
<td>[..]</td>
<td>Returns a slice of a starting at the index b and ending at c.</td>
</tr>
<tr>
<td>Range</td>
<td>a [ .. c ]</td>
<td>[..]</td>
<td>Returns a slice of a starting at the beginning of a and ending at c.</td>
</tr>
<tr>
<td>Range</td>
<td>a [ b .. ]</td>
<td>[..]</td>
<td>Returns a slice of a from the index b to the end of a.</td>
</tr>
</tbody>
</table>

The index operator can be written in two different ways. It can be written as \texttt{ob [ index ]} or \texttt{ob \rightarrow identifier}. However, the latter syntax is equal to \texttt{ob [ "identifier" ]}. You can only index strings, arrays, mapping, multisets and objects, and some of these can only be indexed on certain things as shown in this list:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{string [int]}</td>
<td>Returns the ascii value of the Nth character in the string.</td>
</tr>
<tr>
<td>\texttt{array [int]}</td>
<td>Return the element in the array corresponding to the integer.</td>
</tr>
<tr>
<td>\texttt{array [int]}=mixed</td>
<td>Sets the element in the array corresponding to the integer.</td>
</tr>
<tr>
<td>\texttt{mapping [mixed]}</td>
<td>Returns the value associated with the index, 0 if it is not found.</td>
</tr>
<tr>
<td>\texttt{mapping \rightarrow identifier}</td>
<td>Associate the second mixed value with the first mixed value.</td>
</tr>
<tr>
<td>\texttt{multiset [mixed]}=mixed</td>
<td>If the mixed value is \texttt{true} the index is added to the multiset. Otherwise the index is removed from the multiset.</td>
</tr>
<tr>
<td>\texttt{object [string]}=mixed</td>
<td>Returns the value of the named identifier in the object.</td>
</tr>
<tr>
<td>\texttt{object \rightarrow identifier}</td>
<td>Set the given identifier in the object to the mixed value. Only works if the identifier references a variable in the object.</td>
</tr>
<tr>
<td>\texttt{program [string]}=mixed</td>
<td>Returns the value of the named constant identifier in the program.</td>
</tr>
</tbody>
</table>
5.6. THE ASSIGNMENT OPERATORS

<table>
<thead>
<tr>
<th>Operation</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>string[int..int]</td>
<td>Returns a piece of the string.</td>
</tr>
<tr>
<td>array[int..int]</td>
<td>Returns a slice of the array.</td>
</tr>
</tbody>
</table>

When indexing an array or string it is sometimes convenient to access index from the end instead of from the beginning. This function can be performed by using a negative index. Thus \texttt{arr[-i]} is the same as \texttt{arr[sizeof(arr)-i]}. Note however that this behavior does not apply to the range operator. Instead the range operator clamps it’s arguments to a suitable range. This means that \texttt{a[b..c]} will be treated as follows:

- 
- 
- 
- 
- 

5.6 The assignment operators

There is really only one assignment operator, but it can be combined with lots of other operators to make the code shorter. An assignment looks like this:

\texttt{variable = expression;}

The \texttt{variable} can be a local variable, a global variable or an index in an array, object, multiset or mapping. This will of course set the value stored in \texttt{variable} to \texttt{expression}. Note that the above is also an expression which returns the value of the \texttt{expression}. This can be used in some interesting ways:

\texttt{variable1 = variable2 = 1; // Assign 1 to both variables}
\texttt{variable1 = (variable2 = 1); // Same as above}

\texttt{// Write the value of the expression, if any}
\texttt{if(variable = expression)}
\texttt{write(variable);}

Using assignments like this can however be confusing to novice users, or users who come from a Pascal or Basic background. Especially the if statement can be mistaken for \texttt{if(variable == expression)} which would mean something completely different. As I mentioned earlier, the assignment operator can be combined with another operator to form operators that modify the contents of a variable instead of just assigning it. Here is a list of all the combinations:
CHAPTER 5. OPERATORS

Syntax | Same as | Function
---|---|---
`variable += expression` | `variable = variable + expression` | Add `expression` to `variable`
`variable -= expression` | `variable = variable - expression` | Subtract `expression` from `variable`
`variable *= expression` | `variable = variable * expression` | Multiply `variable` with `expression`
`variable /= expression` | `variable = variable / expression` | Divide `variable` by `expression`
`variable %= expression` | `variable = variable % expression` | Modulo `variable` by `expression`
`variable <<= expression` | `variable = variable << expression` | Shift bits left
`variable >>= expression` | `variable = variable >> expression` | Shift bits right
`variable |= expression` | `variable = variable | expression` | Or `variable` with `expression`
`variable &= expression` | `variable = variable & expression` | And `variable` with `expression`
`variable ^= expression` | `variable = variable ^ expression` | Xor `variable` with `expression`

In all of the above expressions `variable` can actually be any of type of assignable values. Assignable values are also known as lvalues and here is a list of lvalues:

<table>
<thead>
<tr>
<th>Lvalue type</th>
<th>Syntax</th>
<th>Valid assignment type</th>
</tr>
</thead>
<tbody>
<tr>
<td>a local or global variable</td>
<td><code>identifier</code></td>
<td>same as variable</td>
</tr>
<tr>
<td>an element in an array</td>
<td><code>array [ int ]</code></td>
<td>any type</td>
</tr>
<tr>
<td>elements in elements in an array</td>
<td><code>array [ string ]</code></td>
<td>any type</td>
</tr>
<tr>
<td>an element in an string</td>
<td><code>string [ int ]</code></td>
<td>integer</td>
</tr>
<tr>
<td>an element in a mapping</td>
<td><code>mapping[ mixed ]</code> or <code>mapping[ identifier ]</code></td>
<td>any type</td>
</tr>
<tr>
<td>an element in a multiset</td>
<td><code>multiset[ mixed ]</code> or <code>multiset[ identifier ]</code></td>
<td>true / false</td>
</tr>
<tr>
<td>a variable in an object</td>
<td><code>object[ string ]</code> or <code>object[ identifier ]</code></td>
<td>same type as named variable</td>
</tr>
<tr>
<td>a list of lvalues</td>
<td><code>[ lvalue, lvalue ]</code></td>
<td>an array, first value in the array will be assigned to the first lvalue in the list, second value in the array to the second value in the list etc.</td>
</tr>
</tbody>
</table>
5.7 The rest of the operators

Now there are only a couple of operators left. I have grouped them together in this section, not because they are not important, but because they do not fit in any particular categories.

<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Identifier</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>splice</td>
<td>@ a</td>
<td>none</td>
<td>Sends each element in the array a as an individual argument to a function call.</td>
</tr>
<tr>
<td>Increment</td>
<td>++ a</td>
<td>none</td>
<td>Increments a and returns the new value of a.</td>
</tr>
<tr>
<td>Decrement</td>
<td>-- a</td>
<td>none</td>
<td>Decrements a and returns the new value of a.</td>
</tr>
<tr>
<td>Post increment</td>
<td>a ++</td>
<td>none</td>
<td>Increments a and returns the old value of a.</td>
</tr>
<tr>
<td>Post decrement</td>
<td>a --</td>
<td>none</td>
<td>Decrements a and returns the old value of a.</td>
</tr>
<tr>
<td>casting</td>
<td>(type) a</td>
<td>none</td>
<td>Tries to convert a into a value of the specified type.</td>
</tr>
<tr>
<td>Null</td>
<td>a, b</td>
<td>none</td>
<td>Evaluates a and b, then returns b.</td>
</tr>
</tbody>
</table>

The most important of these operators is the calling operator. It is used to call functions. The operator itself is just a set of parenthesis placed after the expression that returns the function. Any arguments to the function should be placed between the parenthesis, separated by commas. We have already seen many examples of this operator, although you might not have realized it was an operator at the time. The function call operator can do more than just calling functions though; if the ‘function’ is in fact an array, the operator will loop over the array and call each element in the array and returns an array with the results. If on the other hand, the ‘function’ is a program, the operator will clone an object from the program and call create() in the new object with the arguments given. In fact, the function `clone` is implemented like this:

```plaintext
object clone(mixed p, mixed ... args) { ( (program)p )->@args; }
```

On the subject of function calls, the splice operator should also be mentioned. The splice operator is an at sign in front of an expression. The expression should always be an array. The splice operator sends each of the elements in the array as a separate argument to the function call. The splice operator can only be used in an argument list for a function call.

Then there are the increment and decrement operators. The increment and decrement operators are somewhat limited: they can only be used on integers. They provide a short and fast way to add or subtract one to an integer. If the operator is written before the variable (++a) the returned value will be what the variable is after the operator has added/subtracted one to it. If the operator is after the variable (a++) it will instead return the value of the variable before it was incremented/decremented.
Casting is used to convert one type to another, not all casts are possible. Here is a list of all casts that actually do anything:

<table>
<thead>
<tr>
<th>casting from</th>
<th>to</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>string</td>
<td>Convert the int to ASCII representation</td>
</tr>
<tr>
<td>float</td>
<td>string</td>
<td>Convert the float to ASCII representation</td>
</tr>
<tr>
<td>string</td>
<td>int</td>
<td>Convert decimal, octal or hexadecimal number to an int. Note that this will only work with decimal numbers in future versions.</td>
</tr>
<tr>
<td>string</td>
<td>float</td>
<td>Convert ASCII number to a float.</td>
</tr>
<tr>
<td>string</td>
<td>program</td>
<td>String is a filename, compile the file and return the program. Results are cached.</td>
</tr>
<tr>
<td>string</td>
<td>object</td>
<td>This first casts the string to a program, (see above) and then clones the result. Results are cached.</td>
</tr>
<tr>
<td>object</td>
<td>type</td>
<td>This calls the function <code>cast</code> with a string containing the type as an argument.</td>
</tr>
<tr>
<td>string</td>
<td>array</td>
<td>This does the inverse of the operation above. Ie. it constructs a string from an array of integers.</td>
</tr>
<tr>
<td>array(int)</td>
<td>string</td>
<td>This recursively casts all values in the array to type.</td>
</tr>
<tr>
<td>array</td>
<td>array(type)</td>
<td>Same as doing <code>values(string)</code></td>
</tr>
<tr>
<td>mapping</td>
<td>array</td>
<td>Same as doing <code>Array.transpose({{indices(mapping)},values(mapping)})</code>. Example: <code>(array)({1:2,3:4})</code> will return <code>({{{1,2}},{{3,4}}})</code></td>
</tr>
<tr>
<td>multiset</td>
<td>array</td>
<td>Same as doing <code>indices(multiset)</code>.</td>
</tr>
<tr>
<td>int</td>
<td>float</td>
<td>Returns a float with the same value as the integer.</td>
</tr>
<tr>
<td>float</td>
<td>int</td>
<td>Returns the integer closest to the float.</td>
</tr>
<tr>
<td>function</td>
<td>object</td>
<td>Same as <code>function_object(function)</code>.</td>
</tr>
</tbody>
</table>

You can also use the cast operator to tell the compiler things. If `a` is a variable of type `mixed` containing an int, then the expression `(int)a` can be used instead of `a` and that will tell the compiler that the type of that expression is `int`.

Last, and in some respect least, is the comma operator. It doesn’t do much. In fact, it simply evaluates the two arguments and then returns the right hand one. This operator is mostly useful to produce smaller code, or to make defines that can be used in expressions.

### 5.8 Operator precedence

When evaluating an expression, you can always use parenthesis to tell the compiler in which order to evaluate things. Normally, the compiler will evaluate things from left to right, but it will evaluate operators with higher priority before those with lower. The following table shows the relative priority of all the operators in descending order:

```
(a) a() a[b] a->b a[b..c] ({}) ([])(<>)
   !a ~a (type)a ++a --a
   a++ a--
   a*b a/b a%b
```
5.9. OPERATOR FUNCTIONS

The expression is evaluated in this order:

<table>
<thead>
<tr>
<th>The expression</th>
<th>is evaluated in this order:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+2*2</td>
<td>1+(2*2)</td>
</tr>
<tr>
<td>1+2<em>2</em>4</td>
<td>1+((2*2)*4)</td>
</tr>
<tr>
<td>(1+2)<em>2</em>4</td>
<td>((1+2)*2)*4</td>
</tr>
<tr>
<td>1+4,c=2</td>
<td>3+5</td>
</tr>
<tr>
<td>1+5 &amp; 4 == 3</td>
<td>(1+(5 &amp; 4)) == 3</td>
</tr>
<tr>
<td>c=1,99</td>
<td>(c=1),99</td>
</tr>
<tr>
<td>!a++ + --a()</td>
<td>(!a++) + (<del>(</del>(---a())))</td>
</tr>
</tbody>
</table>

Examples:

```
Examples:

<table>
<thead>
<tr>
<th>The expression</th>
<th>is evaluated in this order:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+2*2</td>
<td>1+(2*2)</td>
</tr>
<tr>
<td>1+2<em>2</em>4</td>
<td>1+((2*2)*4)</td>
</tr>
<tr>
<td>(1+2)<em>2</em>4</td>
<td>((1+2)*2)*4</td>
</tr>
<tr>
<td>1+4,c=2</td>
<td>3+5</td>
</tr>
<tr>
<td>1+5 &amp; 4 == 3</td>
<td>(1+(5 &amp; 4)) == 3</td>
</tr>
<tr>
<td>c=1,99</td>
<td>(c=1),99</td>
</tr>
<tr>
<td>!a++ + --a()</td>
<td>(!a++) + (<del>(</del>(---a())))</td>
</tr>
</tbody>
</table>
```

5.9 Operator functions

As mentioned earlier a + b can just as well be written as ‘+(a, b). Together with the function map which calls a function for every index in an array and the splice operator this can be used to create some very very fast and compact code. Let’s look at some examples:
Chapter 6

Object orientation

As mentioned several times, Pike is object oriented. This does not mean that it is identical to C++ in any way. Pike uses a less strict approach to object orientation which creates a more relaxed programming style. If you have never come in contact with object oriented programming before, be warned that the ideas expressed in Pike and in this chapter are my own and do not necessarily reflect what other people think about object oriented programming.

6.1 Terminology

As mentioned before, Pike uses a different terminology than C++ does. This has historic reasons, but might be changed in the future. In the meantime, you might benefit from this mini-dictionary which translates Pike-ish terms to C++-ish terms:
6.2 The approach

Think of the data type program as an executable file. Then we clone this program and create an object. The object is then a running program. The object has its own data and its own functions, however, it can work together with other programs by calling functions in those objects. The functions can be thought of as message carriers, TCP sockets or just a way for programs to communicate. Now we have a running system with many running programs, each performing only the task it was designed for.

This analogy has one major flaw, when running programs in UNIX they actually run simultaneously. UNIX is multitasking, Pike is not. When one object is executing code, all the other objects has to wait until they are called. An exception is if you are using threads as will be discussed in a later chapter.

6.3 How does this help?

Ok, why is it a good idea to use object oriented programming? Well if you believe what you hear, the biggest advantage is that you can re-use your code in several projects. In my experience this is not the case. In my experience, the advantages of object oriented programming are:

Most of these things can be done without object orientation, but it is object orientation that makes them easy.

6.4 Pike and object orientation

In most object oriented languages there is a way to write functions outside of all classes. Some readers might think this is what we have been doing until now. However, in Pike, all functions have to reside within a program. When you write a simple script in Pike, the file is first compiled into a program then cloned and then main() is called in that clone. All this is done by the master object, which is compiled and cloned before before all other objects. What the master object actually does is:

```
program scriptclass=compile_file(argv[0]); // Load script
```
6.5. INHERIT

```c
object script=scriptclass(); // clone script
int ret=script->main(sizeof(argv), argv); // call main()
```

Similarly, if you want to load another file and call functions in it, you can do it with `compile_file()`, or you can use the cast operator and cast the filename to a string. You can also use the module system, which we will discuss further in the next chapter.

If you don’t want to put each program in a separate file, you can use the `class` keyword to write all your classes in one file. We have already seen an example how this in chapter 3.4, but let’s go over it in more detail. The syntax looks like this:

```c
class class_name {
    class_definition
}
```

This construction can be used almost anywhere within a normal program. It can be used outside all functions, but it can also be used as an expression in which case the defined class will be returned. In this case you may also leave out the `class_name` and leave the class unnamed. The `class definition` is simply the functions and programs you want to add to the class.

To make it easier to program, defining a class is also to define a constant with that name. Essentially, these two lines of code do the same thing:

```c
class foo {};  
constant foo = class {}; 
```

Because classes are defined as constants, it is possible to use a class defined inside classes you define later, like this:

```c
class foo 
{
    int test() { return 17; }
};

class bar 
{
    program test2() { return foo; }
};
```

6.5 Inherit

A big part of writing object oriented code is the ability to add functionality to a program without changing (or even understanding) the original code. This is what `inherit` is all about. Let’s say I want to change the `hello_world` program
to write a version number before it writes hello world, using `inherit` I could do this like this:

```c
inherit "hello_world"
int main(int argc, array(string) argv)
{
    write("Hello world version 1.0\n");
    return ::main(argc,argv);
}
```

What `inherit` does is that it copies all the variables and functions from the inherited program into the current one. You can then re-define any function or variable you want, and you can call the original one by using a `::` in front of the function name. The argument to `inherit` can be one of the following:

Let’s look at an example. We’ll split up an earlier example into three parts and let each inherit the previous part. It would look something like this:
Note that the actual code is not copied, only the list of references. Also note that the list of inherits is copied when you inherit a program. This does not mean you can access those copied inherits with the :: operator, it is merely an implementation detail. Although this example does not show an example of a re-defined function, it should be easy to see how that works by just changing
what an identifier is pointing at.

### 6.6 Multiple inherit

You can inherit any number of programs in one program, you can even inherit the same thing more than once. If you do this you will get a separate set of functions and variables for each inherit. To access a specific function you need to name your inheirts. Here is an example of named inheirts:

```pike
inherit Stdio.File;  // This inherit is named File
inherit Stdio.FILE;  // This inherit is named FILE
inherit "hello_world";  // This inherit is named hello_world
inherit Stdio.File : test1;  // This inherit is named test1
inherit "hello_world" : test2;  // This inherit is named test2

void test()
{
  File::read();  // Read data from the first inherit
  FILE::read();  // Read data from the second inherit
  hello_world::main(0,({}));  // Call main in the third inherit
  test1::read();  // Read data from the fourth inherit
  test2::main(0,({}));  // Call main in the fifth inherit
  ::read();  // Read data from all inheirts
}
```

As you can see it would be impossible to separate the different read and main functions without using inherit names. If you tried calling just `read` without any :: or inherit name in front of it Pike will call the last read defined, in this case it will call read in the fourth inherit.

If you leave the inherit name blank and just call ::read Pike will call all inherited read() functions. If there is more than one inherited read function the results will be returned in an array.

Let’s look at another example:

``` Pike
#!/usr/local/bin/pike

inherit Stdio.File : input;
inherit Stdio.File : output;

int main(int argc, array(string) argv)
{
  output::create("stdout");
  for(int e=1;e<sizeof(argv);e++)
  {
    input::open(argv[e],"r");
    while(output::write(input::read(4096)) == 4096);
  }
```
This short piece of code works a lot like the UNIX command `cat`. It reads all the files given on the command line and writes them to stdout. As an example, I have inherited Stdio.File twice to show you that both files are usable from my program.

### 6.7 Pike inherit compared to other languages

Many other languages assign special meaning to inherit. Most common is the notion that if you inherit a class, it means that your class should obey the same rules as the inherited class. In Pike, this is not necessarily so. You may wish to use inherit in Pike like this, but you can just as well choose not to. This may confuse some programmers with previous experience in object oriented programming.

### 6.8 Modifiers

Sometimes, you may wish to hide things from inheriting programs, or prevent functions from being called from other objects. To do so you use **modifiers**. A modifier is simply a word written before a variable definition, function definition, class definition or an inherit that specifies how this identifier should interact with other objects and programs. These modifiers are available:

When modifiers are used in conjunction with inherit, all the variables, functions and classes copied from the inherited class will be modified with the keywords
used. For instance, **private inherit** means that the identifiers from this inherit will not be available to program inheriting this program. **static private inherit** will also hide those identifiers from the index and arrow operators, making the inherit available only to the code in this program.

### 6.9 Operator overloading

Sometimes you want an object to act as if it was a string, an integer or some other data type. It is especially interesting to be able to use the normal operators on objects to allow short and readable syntax. In Pike, special methods are called when an operator is used with an object. To some extent, this is work in progress, but what has been done so far is very useful and will not be subject to change.

The following table assumes that a and b are objects and shows what will be evaluated if you use that particular operation on an object. Note that some of these operators, notably == and ! have default behavior which will be used if the corresponding method is not defined in the object. Other operators will simply fail if called with objects. Refer to chapter 4.4 for information on which operators can operate on objects without operator overloading.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Will call</th>
<th>Or</th>
</tr>
</thead>
<tbody>
<tr>
<td>a+b</td>
<td>a+c(b)</td>
<td>b+c(a)</td>
</tr>
<tr>
<td>a+b+c+d</td>
<td>a+c(b,c,d)</td>
<td>(b+c(a))+c+d</td>
</tr>
<tr>
<td>a-b</td>
<td>a-c(b)</td>
<td>b-c(a)</td>
</tr>
<tr>
<td>a&amp;b</td>
<td>a&amp;b(b)</td>
<td>b&amp;b(a)</td>
</tr>
<tr>
<td>a^b</td>
<td>a^b(b)</td>
<td>b^b(a)</td>
</tr>
<tr>
<td>a_¿b</td>
<td>a_¿b(b)</td>
<td>b_¿b(a)</td>
</tr>
<tr>
<td>a_¿b</td>
<td>a_¿b(b)</td>
<td>b_¿b(a)</td>
</tr>
<tr>
<td>a*b</td>
<td>a*(b)</td>
<td>b*(a)</td>
</tr>
<tr>
<td>a<em>b</em>c*d</td>
<td>a*(b,c,d)</td>
<td>b*(a)<em>c</em>d</td>
</tr>
<tr>
<td>a/b</td>
<td>a/b(b)</td>
<td>b/a(f)</td>
</tr>
<tr>
<td>a%b</td>
<td>a%(b)</td>
<td>b%(a)</td>
</tr>
<tr>
<td>!a</td>
<td>!a()</td>
<td>!a(a)</td>
</tr>
<tr>
<td>a_¿=b</td>
<td>!a_¿=(b)</td>
<td>!(b_¿=(a))</td>
</tr>
<tr>
<td>a_¿=b</td>
<td>!a_¿=(b)</td>
<td>!(b_¿=(a))</td>
</tr>
<tr>
<td>a_¿foo</td>
<td>a_¿foo(b)</td>
<td>a_¿foo(b)</td>
</tr>
<tr>
<td>a_¿foo=b</td>
<td>a_¿foo=b</td>
<td>a_¿foo=b</td>
</tr>
<tr>
<td>sizeof(a)</td>
<td>sizeof()</td>
<td>sizeof(a)</td>
</tr>
</tbody>
</table>
6.10. SIMPLE EXERCISES

Here is a really silly example of a program that will write 10 to stdout when executed.

```pike
#!/usr/local/bin/pike
class three {
    int +(int foo) { return 3+foo; }
};

int main()
{
    write(sprintf("%d\n",three()+7));
}
```

It is important to know that some optimizations are still performed even when operator overloading is in effect. If you define a multiplication operator and multiply your object with one, you should not be surprised if the multiplication operator is never called. This might not always be what you expect, in which case you are better off not using operator overloading.

### 6.10 Simple exercises

Make a program that clones 10 hello world and then runs main() in each one of them. Modify the register program to use an object for each record. Modify the register program to use the following search function:

```pike
void find_song(string title)
{
    string name, song;
    int hits;

    title=lower_case(title);

    foreach(indices(records),name)
    {
        if(string song=records[name][title])
        {
            write(name+"; "+song+"\n");
            hits++;
        }
    }
}
```
if(!hits) write("Not found.\n");
}
Chapter 7

Miscellaneous functions

There are some 'functions' in Pike that are not really functions at all but just as builtin as operators. These special functions can do things that no other functions can do, but they can not be re-defined or overloaded. In this chapter I will describe these functions and why they are implemented as special functions.

7.1 sscanf

Sscanf may look exactly like a normal function, but normal functions can not set the variables you send to it. The purpose of sscanf is to match one string against a format string and place the matching results into a list of variables. The syntax looks like this:

\[
\text{int sscanf(string str, string fmt, lvalue ...)}
\]

The string \textit{str} will be matched against the format string \textit{fmt}. \textit{fmt} can contain strings separated by \%d,\%s,\%c and \%f. Every \% corresponds to one lvalue. An lvalue is the name of a variable, a name of a local variable, an index in an array, mapping or object. It is because of these lvalues that sscanf can not be implemented as a normal function.

Whenever a percent is found in the format string, a match is according to the following table:

- \%b reads a binary integer
- \%d reads a decimal integer
- \%o reads an octal integer
- \%x reads a hexadecimal integer
- \%D reads an integer that is either octal (leading zero), hexadeclmal (leading 0x) or decimal.
- \%f reads a float
- \%c matches one char and returns it as an integer
- \%2c matches two chars and returns them as an integer (short)
CHAPTER 7. MISCELLANEOUS FUNCTIONS

%4F matches four chars and returns them as a float (IEEE single precision)

%8F matches eight chars and returns them as a float (IEEE double precision)

%s reads a string. If followed by %d, %s will read any non-numerical characters. If followed by a %[], %s will read any characters not present in the set. If followed by normal text, %s will match all characters up to but not including the first occurrence of that text.

%5s gives a string of 5 characters (5 can be any number)

%[set] matches a string containing a given set of characters (those given inside the brackets). %[^set] means any character except those inside brackets. Example: %[0-9H] means any number or 'H'.

%–format%˝ Repeatedly matches 'format' as many times as possible and assigns an array of arrays with the results to the lvalue.

If a * is put between the percent and the operator, the operator will only match its argument, not assign any variables.

Sscanf does not use backtracking. Sscanf simply looks at the format string up to the next % and tries to match that with the string. It then proceeds to look at the next part. If a part does not match, sscanf immediately returns how many % were matched. If this happens, the lvalues for % that were not matched will not be changed.

Let's look at a couple of examples:

// a will be assigned "oo" and 1 will be returned
sscanf("foo","f%s",a);

// a will be 4711 and b will be "bar", 2 will be returned
sscanf("4711bar","%d%s",a,b);

// a will become "test"
sscanf(" \t test","%*[ \t]%s",a)

// Remove "the " from the beginning of a string
// If 'str' does not begin with "the " it will not be changed
sscanf(str,"the %s",str);

SEE ALSO: sprintf

7.2 catch & throw

Catch is used to trap errors and other exceptions in Pike. It works by making a block of code into an expression, like this:

catch { statements }
If an error occurs, catch will return a description of the error. The description of the error has the following format:

```pike
({
    "error description",
    backtrace()
})
```

If no error occurs, catch will return zero. You may emulate your own errors using the function throw, described in chapter 15.

Example:

```pke
int x,y;
// This might generate "division by zero"
mixed error=catch { x/=y; };
```

### 7.3 gauge

The syntax for gauge is the same as the syntax for catch:

```pke
gauge { statements }
```

However, gauge simply returns how many seconds the code took to execute. This can be used to find out how fast your code actually is. :) Only CPU time used by the Pike process is measured. This means that if it takes two seconds to execute but only uses 50 % CPU, this function will return 1.0.

### 7.4 typeof

This function returns the type of an expression as a string. It does not evaluate the expression at all, which might be somewhat confusing. Example:

```pke
typeof( exit(1) )
```

This will return the string "void" since exit is a function that returns void. It will not execute the function exit and exit the process as you might expect. If you want to know the type after evaluation, use `sprintf("%t", expr).`
CHAPTER 7. MISCELLANEOUS FUNCTIONS
Chapter 8

Modules

A module is a software package that plugs into the Pike programming environment. They provide you with simple interfaces to system routines and they also constitute a neat way to use your own C/C++ code from within Pike. Pike comes with a number of modules that are ready to use. In this chapter I will explain the basics of modules and how to use them.

Here is a list of the basic Pike modules:
* These modules might not be available depending on how Pike was compiled and whether support for these functions exist on your system.
8.1 How to use modules

A module is a bunch of functions, programs or other modules collected in one symbol. For instance, the module Stdio contains the objects stdin, stdout and stderr. To access these objects you can write Stdio.stdin, Stdio.stdout or Stdio.stderr anywhere in your program where an object of that type is acceptable. If you use Stdio a lot you can put import Stdio; in the beginning of your program. This will import all the identifiers from the module Stdio into your program, making it possible to write just stdin instead of Stdio.stdin. It is also possible to import all modules in a directory with import by putting the directory name in double quotes. So, to import all modules in the current directory, you would use import ".".

8.2 Where do modules come from?

Modules are not loaded until you use them, which saves memory unless you use all the modules. However, if you want to write your own modules it is important to know how modules are located and loaded.

When you use Stdio Pike will look for that module:

1. ...
2. ...
3. ...
4. ...
5. ...

For each of these directories, Pike will do the following:

1. ...
2. ...
3. ...

As you can see, quite a lot of work goes into finding the modules, this makes it possible to choose the most convenient way to build your own Pike modules.

8.3 The . operator

The period operator is not really an operator, as it is always evaluated during the compilation. It works similarly to the index and arrow operators, but can only be used on constant values such as modules. In most cases, modules are simply a clone of a program, in which case the identifiers in the module will be the same as those in the program. But some modules, like those created from directories, overload the index operator so that the identifiers in the module
can be something other than those in the program. For directory modules, the index operator looks in the directory it was cloned for to find the identifiers.

You can also use the . operator without an identifier preceeding it to access modules in the same directory as the program itself. For instance, .my_module.foo would mean 'the identifier foo in the module my_module in this directory.'

8.4 How to write a module

Here is an example of a simple module:

```pike
constant PI = 3.14159265358979323846264338327950;
float cos2(float f) { return pow(cos(f),2.0); }
```

if we save this short file as Trig.pmod we can now use this module like this:

```pike
int main()
{
   write(sprintf("%f\n",.Trig.cos2(.Trig.PI));
}
```

or like this:

```pike
import .Trig;
int main()
{
   write(sprintf("%f\n",cos2(PI));
}
```

8.5 Simple exercises

Save the hello_world.pike program as hello_world.pike.pmod, then make a program that loads this module and calls its main(). Make a directory called Programs.pmod and put all the examples you have written so far in it. Make a program that runs one of those programs. Make sure the program can be modified to run another of your examples by changing what module it loads. Copy the file hello_world.pike.pmod to programs/module.pike.pmod and then write a program that runs hello_world without actually using the identifier hello_world. Try putting Programs.pmod in another directory and then try to run the programs from the last two examples.
Chapter 9

File I/O

Programming without reading and writing data from files, sockets, keyboard etc. would be quite pointless. Luckily enough, Pike provides you with an object oriented interface to files, pipes and TCP sockets. All I/O functions and classes are collected in the module Stdio.

9.1 File management - Stdio.File

Stdio.File
This is the basic I/O object, it provides socket communication as well as file access. It does not buffer reads and writes or provide line-by-line reading, that is done in the FILE object. Stdio.File is completely written in C. What follows is a description of all the functions in Stdio.File.

Stdio.File.create - init file struct
object(Stdio.File) Stdio.File();
object(Stdio.File) Stdio.File(string filename);
object(Stdio.File) Stdio.File(string filename, string mode);
object(Stdio.File) Stdio.File(string filename, string mode, int mask);
object(Stdio.File) Stdio.File(string fd);
object(Stdio.File) Stdio.File(int fd);
object(Stdio.File) Stdio.File(int fd, string mode);

Description
There are four different ways to clone a File. The first is to clone it without any arguments, in which case the you have to call open(), connect() or some other method which connects the File object with a stream.

However, instead of cloning and then calling open(), you can clone the File with a filename and open mode. This is the same thing as cloning and then calling open, except shorter and faster. Default open mode is "r" and default mask is 0666.
Alternatively, you can clone a File with "stdin", "stdout" or "stderr" as argument. This will open the specified standard stream.

For the advanced users, you can use the file descriptors of the systems (note: emulated by pike on some systems - like NT). This is only useful for streaming purposes on unix systems. This is not recommended at all if you don’t know what you’re into. Default mode for this is "rw".

**Note**

Open mode will be filtered through the system UMASK. You might need to use chmod later.

*See Also*
clone and Stdio.File->open

**Stdio.File.open** - open a file

```c
int open(string filename, string how);
int open(string filename, string how, int mode);
```

**Description**

Open a file for read, write or append. The variable `how` should contain one or more of the following letters:

- `'r'` open file for reading
- `'w'` open file for writing
- `'a'` open file for append (use with 'w')
- `'t'` truncate file at open (use with 'w')
- `'c'` create file if it doesn’t exist (use with 'w')
- `'x'` fail if file already exist (use with 'c')

`how` should _always_ contain at least one of `'r` or `'w`.

The third argument is protection bits if the file is created. Default is 0666 (all read+write, in octal notation).

**Returns**

1 on success, 0 otherwise

*See Also*
Stdio.File->close

**Stdio.File.close** - close a file

```c
int close(string how);
int close();
```

**Description**

Close the file. Optionally, specify "r", "w" or "rw" to close just the read, just the write or both read and write part of the file respectively. Note that this function will not call the close callback.

*See Also*
Stdio.File->open
**Stdio.File.read** - read data from a file or stream

```
string read(int nbytes);
string read(int nbytes, int notall);
string read();
```

*Description*
Read tries to read `nbytes` bytes from the file, and return it as a string. If something goes wrong, zero is returned.

If a one is given as second argument to `read()`, read will not try its best to read as many bytes as you asked it to read, it will merely try to read as many bytes as the system read function will return. This mainly useful with stream devices which can return exactly one row or packet at a time.

If no arguments are given, `read` will read to the end of the file/stream.

*See Also*
`Stdio.File->read` and `Stdio.File->write`

**Stdio.File.read_oob** - read out-of-band data from a stream

```
string read_oob(int nbytes);
string read_oob(int nbytes, int notall);
string read_oob();
```

*Description*
Tries to read `nbytes` bytes of out-of-band data from the stream, and returns it as a string. If something goes wrong, zero is returned.

If a one is given as a second argument to `read_oob()`, only as many bytes of out-of-band data as are currently available will be returned.

If no arguments are given, `read_oob` will read to the end of the stream.

*Note*
This function is only available if the option `--with-oob` was specified when Pike was compiled.

It is not guaranteed that all out-of-band data sent from the other end will be received. Most streams only allow for a single byte of out-of-band data at a time.

*See Also*
`Stdio.File->read` and `Stdio.File->write_oob`

**Stdio.File.write** - write data to a file or stream

```
int write(string data);
```

*Description*
Write data to file or stream and return how many bytes that were actually written. 0 is returned in nonblocking mode if it was not possible to write anything without blocking. -1 is returned if something went wrong and no bytes were written.
See Also
Stdio.File->read and Stdio.File->write_oob

**Stdio.File.write_oob** - write out-of-band data to a stream

```c
int write_oob(string data);
```

**Description**
Writes out-of-band data to a stream and returns how many bytes that were actually written. -1 is returned if something went wrong and no bytes were written.

**Note**
This function is only available if the option `--with-oob` was specified when Pike was compiled.

It is not guaranteed that all out-of-band data will be received at the other end. Most streams only allow for a single byte of out-of-band data at a time. Some streams will send the rest of the data as ordinary data.

See Also
Stdio.File->read_oob and Stdio.File->write

**Stdio.File.seek** - seek to a position in a file

```c
int seek(int pos);
```

**Description**
Seek to a position in a file, if pos is less than zero, seek to position pos relative end of file. Returns -1 for failure, or the new position in the file when successful.

See Also
Stdio.File->tell

**Stdio.File.tell** - tell where we are in a file

```c
int tell();
```

**Description**
Returns the current position in the file.

See Also
Stdio.File->seek

**Stdio.File.truncate** - truncate a file

```c
int truncate(int length);
```

**Description**
Truncates a file to that length. Returns 1 if ok, 0 if failed.

See Also
Stdio.File->open
**Stdio.File.stat** - do file_stat on an open file

```php
array(int) stat();
```

**Description**
This function returns the same information as the function `file_stat`, but for the file it is called in. If file is not an open file, zero will be returned. Zero is also returned if file is a pipe or socket.

**See Also**

`file_stat`

**Stdio.File.errno** - what was last error?

```php
int errno();
```

**Description**
Returns the error code for the last command on this file. Error code is normally cleared when a command is successful.

**Stdio.File.set_buffer** - set internal socket buffer

```php
void set_buffer(int bufsize, string mode);
void set_buffer(int bufsize);
```

**Description**
This function sets the internal buffer size of a socket or stream. The second argument allows you to set the read or write buffer by specifying "r" or "w". It is not guaranteed that this function actually does anything, but it certainly helps to increase data transfer speed when it does.

**See Also**

`Stdio.File->open_socket` and `Stdio.Port->accept`

**Stdio.File.set_nonblocking** - make stream nonblocking

```php
void set_nonblocking(function(mixed, string void) read_callback,
```

or

```php
void set_nonblocking(function(mixed, string void) read_callback,
```

or

```php
void set_nonblocking();
```

**Description**
This function sets a stream to nonblocking mode. When data arrives on the
stream, read_callback will be called with some or all of this data. When the stream has buffer space over for writing, write_callback is called so you can write more data to it. If the stream is closed at the other end, close_callback is called.

When out-of-band data arrives on the stream, read_oob_callback will be called with some or all of this data. When the stream allows out-of-band data to be sent, write_oob_callback is called so that you can write out-of-band data to it.

All callbacks will have the id of file as first argument when called.

If no arguments are given, the callbacks are not changed. The stream is just set to nonblocking mode.

Note
Out-of-band data is only supported if Pike was compiled with the option ‘–with-oob’.

See Also
Stdio.File->set_blocking and Stdio.File->set_id

**Stdio.File.set_read_callback** - set the read callback

void set_read_callback(function read_callback)

**Description**
This function sets the read callback for the file. The read callback is called whenever there is data to read from the file. Note that this function does not set the file nonblocking.

See Also
Stdio.File->set_nonblocking

**Stdio.File.set_write_callback** - set the write callback

void set_write_callback(function write_callback)

**Description**
This function sets the write callback for the file. The write callback is called whenever there is buffer space available to write to for the file. Note that this function does not set the file nonblocking.

See Also
Stdio.File->set_nonblocking

**Stdio.File.set_close_callback** - set the close callback

void set_close_callback(function close_callback)

**Description**
This function sets the close callback for the file. The close callback is called when the remote end of a socket or pipe is closed. Note that this function does not set the file nonblocking.
9.1. FILE MANAGEMENT - STDIO.FILE

See Also
Stdio.File->set_nonblocking

Stdio.File.set_blocking - make stream blocking

void set_blocking();

Description
This function sets a stream to blocking mode. i.e. all reads and writes will wait until data has been written before returning.

See Also
Stdio.File->set_nonblocking

Stdio.File.set_id - set id of this file

void set_id(mixed id);

Description
This function sets the id of this file. The id is mainly used as an identifier that is sent as the first arguments to all callbacks. The default id is 0. Another possible use of the id is to hold all data related to this file in a mapping or array.

See Also
Stdio.File->query_id

Stdio.File.query_id - get id of this file

mixed query_id();

Description
This function returns the id of this file.

See Also
Stdio.File->set_id

Stdio.File.query_read_callback - return the read callback function

function query_read_callback();

Description
This function returns the read_callback, which is set with set_nonblocking or set_read_callback.

See Also
Stdio.File->set_nonblocking and Stdio.File->set_read_callback

Stdio.File.query_write_callback - return the write callback function

function query_write_callback();
**Description**
This function returns the write callback, which is set with set_nonblocking or set_write_callback.

**See Also**
Stdio.File->set_nonblocking and Stdio.File->set_write_callback

**Stdio.File.query_close_callback** - return the close callback function

```plaintext
function query_close_callback();
```

**Description**
This function returns the close callback, which is set with set_nonblocking or set_close_callback.

**See Also**
Stdio.File->set_nonblocking and Stdio.File->set_close_callback

**Stdio.File.dup** - duplicate a file

```plaintext
object(Stdio.File) dup();
```

**Description**
This function returns a clone of Stdio.File with all variables copied from this file. Note that all variables, even id, is copied.

**See Also**
Stdio.File->assign

**Stdio.File.dup2** - duplicate a file over another

```plaintext
int dup2(object(Stdio.File) to);
```

**Description**
This function works similarly to Stdio.File->assign, but instead of making the argument a reference to the same file, it creates a new file with the same properties and places it in the argument.

**Example**

```plaintext
/* Redirect stdin to come from the file 'foo' */
object o=Stdio.File();
o->open("foo","r");
o->dup2(Stdio.File("stdin"));
```

**See Also**
Stdio.File->assign and Stdio.File->dup
**Stdio.File.assign** - assign a file

```c
void assign(object f);
```

*Description*

This function takes a clone of Stdio.File and assigns all variables of this file from it. It can be used together with file-\$dup to move files around.

*See Also*

Stdio.File->dup

**Stdio.File.open_socket** - open a socket

```c
int open_socket(int|void port, string|void address);
```

*Description*

This makes this file into a socket ready for connection. The reason for this function is so that you can set the socket to nonblocking or blocking (default is blocking) before you call Stdio.File->connect() This function returns 1 for success, 0 otherwise.

If you give a port number to this function, the socket will be bound to this port locally before connecting anywhere. This is only useful for some silly protocols like FTP. You may also specify an address to bind to if your machine has many IP numbers.

*See Also*

Stdio.File->connect and Stdio.File->set_nonblocking

**Stdio.File.connect** - connect a socket to something

```c
int connect(string IP, int port);
```

*Description*

This function connects a socket previously created with Stdio.File->open_socket to a remote socket. The argument is the IP name or number for the remote machine.

This function returns 1 for success, 0 otherwise. Note that if the socket is in nonblocking mode, you have to wait for a write or close callback before you know if the connection failed or not.

*See Also*

Stdio.File->query_address

**Stdio.File.query_address** - get addresses

```c
string query_address();
string query_address(1);
```

*Description*

This function returns the remote or local address of a socket on the form ”x.x.x.x port”. Without argument, the remote address is returned, with argument
local address is returned. If this file is not a socket, not connected or some other error occurs, zero is returned.

*See Also*

Stdio.File->connect

*Stdio.File.pipe* - create a two-way pipe

```c
object pipe();
```

*Description*

This function creates a pipe between the object it was called in and an object that is returned. The two ends of the pipe are indistinguishable. If the File object this function is called in was open to begin with, it is closed before the pipe is created.

*See Also*

fork

*Stdio.File.set_close_on_exec* - set / clear the close on exec flag

```c
void set_close_on_exec(int onoff);
```

*Description*

This function determines whether this file will be closed when calling exece. Default is that the file WILL be closed on exec except for stdin, stdout and stderr.

*See Also*

exece

Here is an example of how to use the TCP functions in Stdio.File in blocking mode. This short program takes a URL as first argument, connects to the WWW server, sends a HEAD request and writes the reply to stdout. For clarity, all calls to Stdio.File use File:: even if that is not strictly necessary.

```c
import Stdio;
inherit File;

int main(int argc, array(string) argv)
{
    string host;
    string path="";
    int port=80;
    sscanf(argv[1],"http://%s",argv[1]);
    sscanf(argv[1],"%s/%s",host,path);
    sscanf(host,"%s:%d",host,port);

    if(!File::open_socket())
    {
        perror("Open socket failed");
        exit(1);
    }
```
if(!File::connect(host,port))
{
    perror("Failed to connect to remote host");
    exit(1);
}

File::write(sprintf("HEAD /%s HTTP/1.0\n",path));
stdout::write(File::read());

9.2 Buffered file management - Stdio.FILE

**Stdio.FILE**
Stdio.FILE is a buffered version of Stdio.File, it inherits Stdio.File and has most of the functionality of Stdio.File. However, it has an input buffer that allows line-by-line input. Note that the output part of Stdio.FILE is not buffered at this moment. The added functionality of Stdio.FILE is described here:

**Stdio.FILE.gets** - get one line

**string gets();**

*Description*
This function returns one line from the FILE, it returns zero if no more lines are available.

**Stdio.FILE.printf** - formatted print

**string printf(string format, mixed ... data);**

*Description*
This function does approximately the same as: write(sprintf(format,@data))

*See Also*
sprintf

**Stdio.FILE.ungets** - put a string back in the buffer

**string ungets(string s);**

*Description*
This function puts a string back in the input buffer. The string can then be read with read, gets or getchar.
**Stdio.FILE.getchar** - get one character from the input stream

```c
int getchar();
```

**Description**
This function returns one character from the input stream. Note that the return value is the ascii value of the character, not a string containing one character.

### 9.3 Standard streams - Stdio.stdin, stdout and stderr

Any UNIX program has three files open from the beginning. These are called standard input, standard output and standard error stream. These streams are available from Pike as well. They are called Stdio.stdin, Stdio.stdout and Stdio.stderr respectively. Standard input is a clone of Stdio.FILE, which means you can use the line oriented functions. Stdio.stdout and Stdio.stderr are simply clones of Stdio.File.

**Example:**

```c
int main()
{
    int line;
    while(string s=Stdio.stdin.gets())
        write(sprintf("%5d: %s\n", line++, s));
}
```

This example will read lines from standard input for as long as there are more lines to read. Each line will then be written to stdout together with the line number. We could use Stdio.stdout.write instead of just write because they are the same function.

### 9.4 Listening to sockets - Stdio.Port

**Stdio.Port**

Stdio.File can handle connections to any TCP socket, but it cannot listen to a local TCP port. For this purpose there is a special class called Stdio.Port. Stdio.Port cannot read or write any data, but it can accept connections which will be returned as clones of Stdio.File. These are the methods available in Stdio.Port:

**Stdio.Port.bind** - open socket and bind it to a port

```c
int bind(int port);
int bind(int port, function accept_callback);
int bind(int port, function accept_callback, string IP);
```
**Description**

Bind opens a sockets and binds it to port number on the local machine. If the second argument is present, the socket is set to nonblocking and the callback function is called whenever something connects to the socket. The callback will receive the id for this port as argument. Bind returns 1 on success, and zero on failure.

If the optional argument *IP* is given, bind will try to bind to this IP name (or number).

**See Also**

Stdio.Port->accept

**Stdio.Port.listen_fd** - listen to an already open port

```c
int listen_fd(int fd);
int listen_fd(int fd, function accept_callback);
```

**Description**

This function does the same as Stdio.Port->bind, except that instead of creating a new socket and bind it to a port, it expects that the file descriptor *fd* is an already open port.

**Note**

This function is only for the advanced user, and is generally used when sockets are passed to Pike at exec time.

**See Also**

Stdio.Port->bind and Stdio.Port->accept

**Stdio.Port.create** - create and/or setup a port

```c
object(Stdio.Port) Stdio.Port("stdin")
object(Stdio.Port) Stdio.Port("stdin", function accept_callback)
object(Stdio.Port) Stdio.Port("stdin", function accept_callback)
object(Stdio.Port) Stdio.Port(int port)
object(Stdio.Port) Stdio.Port(int port, function accept_callback)
object(Stdio.Port) Stdio.Port(int port, function accept_callback, string ip)
```

**Description**

When create is called with "stdin" as argument, a socket is created out of the file descriptor 0. This is only useful if that actually is a socket to begin with. When create is called with an int as first argument, it does the same as bind() would do with the same arguments. The second and third argument has the same function as in the bind() call.

**See Also**

clone and Stdio.Port->bind

**Stdio.Port.set_id** - set the id of a port

```c
void set_id(mixed id);
```
Description
This function sets the id used for accept_callback by this port. The default id is this_object().

See Also
Stdio.Port->query_id

Stdio.Port.query_id - Return the id for this port.
mixed query_id();

Description
This function returns the id for this port. The id is normally the first argument to accept_callback.

See Also
Stdio.Port->set_id

Stdio.Port.errno - return the last error
int errno();

Description
If the last call done on this port failed, errno will return an integer describing what went wrong. Refer to your Unix manual for further information.

See Also
Stdio.Port->errno

Stdio.Port.accept - accept a connection
object accept();

Description
This function completes a connection made from a remote machine to this port. It returns a two-way stream in the form of a copy of Stdio.File. The new file is by default set to blocking.

See Also
Stdio.File

9.5 UDP socket and message management - Stdio.UDP

Stdio.UDP
Stdio.UDP();

Description
Stdio.UDP is the way of transceiving UDP from Pike.

See Also
Stdio.File
**Stdio.UDP.bind**

bind(int port); bind(int port,string address);

*Description*
binds a port for receiving or transmitting UDP

*Returns*
the called object

**Stdio.UDP.enable broadcast**

enable_broadcast()

*Description*
enable transmission of broadcasts. This is normally only available to root users.

*Returns*
1 upon success, 0 otherwise

**Stdio.UDP.read**

read()
read(int flags)

*Description*
read from the UDP socket. Flags is a bitfield, 1 for out of band data and 2 for peek.

*Returns*
mapping(string:int—string) in the form

```
{
  "data" : string received data
  "ip" : string received from this ip
  "port" : int ...and this port
}
```

*See Also*
Stdio.UDP.set_nonblocking and Stdio.UDP.set_read_callback

**Stdio.UDP.send**

send(string to_addr,int to_port,string data)
send(string to_addr,int to_port,string data,int flags)

*Description*

*Returns*
number of bytes sent
**Stdio.UDP.set_nonblocking**

set_nonblocking();
set_blocking();
set_nonblocking(function, mixed ...);

**Description**
sets this object to be nonblocking or blocking. If set_nonblocking is given with argument, these are passed to set_read_callback().

**Returns**
the called object

**Stdio.UDP.set_read_callback**

set_read_callback(function(mapping(string:int|string), mixed...), mixed ... extra);

**Description**
The called function is receiving mapping similar to the return value of read:

```
{
    "data" : string receieved data
    "ip" : string receieved from this ip
    "port" : int ...and this port
}
```

**Returns**
the called object

**Stdio.UDP.query_address**

query_address()

**Description**

**Returns**
the current address in format "ip:port".

**See Also**
Stdio.File.query_address

---

**9.6 Terminal management - Stdio.Terminfo**

**Stdio.Terminfo.getFallbackTerm** - get fallback terminal

object(Stdio.Terminfo.Terminfo)|object(Stdio.Terminfo.Termcap)
getFallbackTerm(string term);
9.6. TERMINAL MANAGEMENT - STDIO.TERMINFO

Description
Returns an object describing the fallback terminal for the terminal term. This is usually equivalent to Stdio.Terminfo.getTerm("dumb").

See Also
Stdio.Terminfo.getTerm

Stdio.Terminfo.getTerm - get terminal description

object(Stdio.Terminfo.Terminfo)|object(Stdio.Terminfo.Termcap)
getTerm();
or
object(Stdio.Terminfo.Terminfo)|object(Stdio.Terminfo.Termcap)
getTerm(string term);

Description
Returns an object describing the terminal term. If term is not specified, it will default to getenv("TERM") or if that fails to "dumb".

Lookup of terminal information will first be done in the systems terminfo database, and if that fails in the termcap database. If neither database exists, a hardcoded entry for "dumb" will be used.

See Also
Stdio.Terminfo.getTerminfo, Stdio.Terminfo.getTermcap and Stdio.getFallbackTerm

Stdio.Terminfo.getTermcap - get termcap description

object(Stdio.Terminfo.Termcap) getTermcap(string term);

Description
Return the terminal description of term from the systems termcap database. Returns 0 if not found.

See Also
Stdio.Terminfo.getTerm and Stdio.Terminfo.getTerminfo

Stdio.Terminfo.getTerminfo - get terminfo description

object(Stdio.Terminfo.Terminfo) getTerminfo(string term);

Description
Return the terminal description of term from the systems terminfo database. Returns 0 if not found.

See Also
Stdio.Terminfo.getTerm and Stdio.Terminfo.getTermcap

9.6.1 Stdio.Terminfo.Termcap
**Stdio. Terminfo. Termcap**

Termcap terminal description object.

**Stdio. Terminfo. Termcap.create** - initialize termcap object

object(Stdio.Terminfo.Termcap) Termcap(string cap);

or

object(Stdio.Terminfo.Termcap) Termcap(string cap, object tcdb);

or

object(Stdio.Terminfo.Termcap) Termcap(string cap, object tcdb, int maxrecurse);

**Stdio. Terminfo. Termcap.puts** - put termcap string

9.6.2 **Stdio. Terminfo. Terminfo**

**Stdio. Terminfo. Terminfo**

Terminfo terminal description object.

9.7 **Simple input-by-prompt - Stdio.Readline**

**Stdio.Readline**

**Stdio.Readline.create** - init readline

object(Stdio.Readline) Stdio.Readline();

or

object(Stdio.Readline) Stdio.Readline(object infd);

or

object(Stdio.Readline) Stdio.Readline(object infd,

or

object(Stdio.Readline) Stdio.Readline(object infd,

or

object(Stdio.Readline) Stdio.Readline(object infd,

or

Description

Creates a Readline object, that takes input from *infd*, and has output on *outfd*.
9.8 OTHER STDIO FUNCTIONS

`infd` defaults to `Stdio.stdout`.

`interm` defaults to `Stdio.Termino.getTerm()`.

`outfd` defaults to `infd`, unless `infd` is 0, in which case `outfd` defaults to `Stdio.stdout`.

`outterm` defaults to `interm`.

9.8 Other Stdio functions

The Stdio module also contains a collection of high level IO functions to make it easy to write short and readable Pike programs. Most of these functions are implemented using Stdio.File and Stdio.FILE.

`Stdio.append_path` - append paths in a secure way

```pke
string append_path(string absolute, string ... relative);
```

**Description**

Append relative paths to an absolute path and remove any "//", "." or "/." to produce a straightforward absolute path as a result. "./" is ignored in the relative paths if it makes the created path begin with something else than the absolute path (or so far created path).

**Example**

```pke
> Stdio.append_path("/foo/bar/","..");
Result: /foo/bar/
> Stdio.append_path("/foo/bar/","../apa.c");
Result: /foo/bar/apa.c
> Stdio.append_path("/foo/bar/","./sune.c");
Result: /foo/bar/sune.c
> Stdio.append_path("/foo/bar/","bertil/.../sune.c");
Result: /foo/bar/sune.c
> Stdio.append_path("/foo/bar/","klas","bertil/.../sune.c");
Result: /foo/bar/klas/sune.c
```

**See Also**

`combine_path`

`Stdio.filesize` - return the size of a file in bytes

```pke
int filesize(string file);
```

**Description**

Give the size of a file. Size -1 indicates that the file either does not exist, or that it is not readable by you. Size -2 indicates that it is a directory.

**See Also**

`Stdio.write_file` and `Stdio.read_bytes`
**Stdio.exist** - check if a path exists

```
int exist(string path);
```

Description
Returns true if the given path exists (is a directory or file), otherwise false.

See Also
`Stdio.is_dir`, `Stdio.is_file` and `Stdio.is_link`

**Stdio.is_dir** - check if a path is a directory

```
int is_dir(string path);
```

Description
Returns true if the given path is a directory, otherwise false.

See Also
`Stdio.exist`, `Stdio.is_file` and `Stdio.is_link`

**Stdio.is_file** - check if a path is a file

```
int is_file(string path);
```

Description
Returns true if the given path is a file, otherwise false.

See Also
`Stdio.exist`, `Stdio.is_dir` and `Stdio.is_link`

**Stdio.is_link** - check if a path is a symbolic link

```
int is_link(string path);
```

Description
Returns true if the given path is a symbolic link, otherwise false.

See Also
`Stdio.exist`, `Stdio.is_dir` and `Stdio.is_file`

**Stdio.mkdirhier** - make a directory hierarchy

```
int mkdirhier(string pathname, void|int mode);
```

Description
Creates zero or more directories to ensure that the given pathname is a directory. Returns zero if it fails and nonzero if it is successful. If a mode is given, it’s used for the new directories after being &’ed with the current umask (on OS’es that supports this).

See Also
`mkdir`
9.8. OTHER STDIO FUNCTIONS

**Stdio.perror** - print error

```c
void perror(string s);
```

**Description**
This function prints a message to stderr along with a description of what went wrong if available. It uses the system errno to find out what went wrong, so it is only applicable to IO errors.

**See Also**
Stdio.werror

**Stdio.read_bytes** - read a number of bytes into a string from a file

```c
string read_bytes(string file, int start, int len);
string read_bytes(string file, int start);
string read_bytes(string file);
```

**Description**
Read `len` number of bytes from file `file` starting at byte `start` and return it as a string. If `len` is omitted, the rest of the file will be returned. If `start` is also omitted, the entire file will be returned.

**See Also**
Stdio.write_file

**Stdio.read_file** - read a number of lines into a string from file

```c
string read_file(string file, int start, int len);
string read_file(string file);
```

**Description**
Read `len` lines from the file `file` after skipping `start` lines and return those lines as a string. If `start` and `len` are omitted the whole file is read.

**See Also**
Stdio.read_bytes and Stdio.write_file

**Stdio.readline** - read a line from stdin

```c
string readline(string prompt);
```

**Description**
This function is gone. Use Stdio.ReadLine()-read instead.

**See Also**
Stdio.File

**Stdio.recursive_rm** - remove a file or a directory tree recursively

```c
int recursive_rm(string path);
```
Description
Remove a file or directory a directory tree, return 0 if it fails. Nonzero otherwise.

See Also
rm

Stdio.sendfile - send contents from one file to another

object sendfile(array(string) headers, or

object sendfile(array(string) headers,

or

object sendfile(array(string) headers,

Description
Sends headers followed by len bytes starting at offset from the file from followed by trailers to the file to. When completed callback is called with the total number of bytes sent as the first argument, followed by args. Any of headers, from and trailers may be left out by setting them to 0. Setting offset to -1 means send from the current position in from. Setting len to -1 means send until from’s end of file is reached.

Note
The sending is performed asynchronously, and may complete before the function returns.

For callback to be called, the backend must be active (ie main() must have returned -1).

In some cases, the backend must also be active for any sending to be performed at all.

See Also
Stdio.File->set_nonblocking

Stdio.werror - write to stderr

void werror(string s);

Description
Writes a message to stderr. Stderr is normally the console, even if the process output has been redirected to a file or pipe.

Stdio.write_file - append a string to a file

int write_file(string file, string str)
9.9 A SIMPLE EXAMPLE

Description
Append the string \textit{str} onto the file \textit{file}. Returns number of bytes written.

See Also
\texttt{Stdio.read_bytes}

9.9 A simple example

Here is a simple example of how to use the Stdio-functions.

\begin{verbatim}
string grep(string indata, string needle)
{
    object out=Stdio.File(), in=Stdio.File();
    object process =
        Process.create_process(("/bin/grep", needle),
            (["stdin": out->pipe(),
              "stdout": in->pipe()]);
    out->write(indata);
    out->close();
    process->wait();
    return in->read();
}
\end{verbatim}

This function filters the indata through the UNIX-command /bin/grep and return the result.

9.10 A more complex example - a simple WWW server

As most of you know, WWW WWW (World Wide Web), works by using a client program which will fetch files from remote servers when asked. Usually by clicking a picture or text. This example is a program for the server which will send files to any computer that requests them. The protocol used to send the file is called HTTP. (Hyper-Text Transfer Protocol)

Usually WWW involves HTML. HTML (Hyper-Text Markup Language) is a way to write documents with embedded pictures and links to other pages. These links are normally displayed underlined and if you click them your WWW-browser will load whatever document that link leads to.

```pikescript
#!/usr/local/bin/pike
/* A very small httpd capable of fetching files only.
 * Written by Fredrik Hübìnnettë as a demonstration of Pike.
 */
```
We inherit Stdio.Port into this program so we can bind a TCP socket to accept incoming connection. A socket is simply a number to separate communications to and from different programs on the same computer.

Next are some constants that will affect how uHTTPD will operate. This uses the preprocessor directive #define. The preprocessor is the first stage in the compiling process and can make textual processing of the code before it is compiled. As an example, after the first define below, all occurrences of 'BLOCK' will be replaced with 16060.

/* Amount of data moved in one operation */
#define BLOCK 16060

/* Where do we have the html files ? */
#define BASE "/usr/local/html/

/* File to return when we can't find the file requested */
#define NOFILE "/usr/local/html/nofile.html"

/* Port to open */
#define PORT 1905

A port is a destination for a TCP connection. It is simply a number on the local computer. 1905 is not the standard port for HTTP connections though, which means that if you want to access this WWW server from a browser you need to specify the port like this: http://my.host.my.domain:1905/

Next we declare a class called output_class. Later we will clone one instance of this class for each incoming HTTP connection.

class output_class
{
    inherit Stdio.File : socket;
    inherit Stdio.File : file;

    Our new class inherits Stdio.File twice. To be able to separate them they are then named 'socket' and 'file'.

    int offset=0;

    Then there is a global variable called offset which is initialized to zero. (Each instance of this class will have its own instance of this variable, so it is not truly global, but...) Note that the initialization is done when the class is cloned (or instantiated if you prefer C++ terminology).

    Next we define the function write_callback(). Later the program will go into a 'waiting' state, until something is received to process, or until there is buffer
space available to write output to. When that happens a callback will be called
to do this. The write_callback() is called when there is buffer space available. In
the following lines 'void' means that it does not return a value. Write callback
will be used further down as a callback and will be called whenever there is
room in the socket output buffer.

```c
void write_callback()
{
    int written;
    string data;
}
```

The following line means: call seek in the inherited program 'file'.

```c
file::seek(offset);
```

Move the file pointer to the where we want to the position we want to read
from. The file pointer is simply a location in the file, usually it is where the last
read() ended and the next will begin. seek() can move this pointer to where we
want it though.

```c
data=file::read(BLOCK);
```

Read BLOCK (16060) bytes from the file. If there are less that that left to
read only that many bytes will be returned.

```c
if(strlen(data))
{
    written=socket::write(data);
    offset+=written;
    return;
}
```

If we managed to read something...

```c
written=socket::write(data);
```

... we try to write it to the socket.

```c
if(written >= 0)
{
    offset+=written;
    return;
}
```

Update offset if we managed to write to the socket without errors.

```c
werror("Error: "+socket::errno()+"\n");
```
If something went wrong during writing, or there was nothing left to read we destruct this instance of this class.

    destruct(this_object());

That was the end of write_callback()

Next we need a variable to buffer the input received in. We initialize it to an empty string.

    string input="";

And then we define the function that will be called when there is something in the socket input buffer. The first argument 'id' is declared as mixed, which means that it can contain any type of value. The second argument is the contents of the input buffer.

    void read_callback(mixed id,string data)
    {
        string cmd;

        input+=data;

        Append data to the string input. Then we check if we have received a a complete line yet. If so we parse this and start outputting the file.

        if(sscanf(input,"%s %s%[\012\015 \t"]",cmd,input)>2)
        {

            This sscanf is pretty complicated, but in essence it means: put the first word in 'input' in 'cmd' and the second in 'input' and return 2 if successful, 0 otherwise.

            if(cmd!="GET")
            {
                werror("Only method GET is supported.

                destruct(this_object());
                return;
            }

            If the first word isn't GET print an error message and terminate this instance of the program. (and thus the connection)

            sscanf(input,"%s\/[\%\s",input);

            }
9.10. A MORE COMPLEX EXAMPLE - A SIMPLE WWW SERVER

Remove the leading slash.

\[
\text{input} = \text{BASE} + \text{combine_path}(/, \text{input});
\]

Combine the requested file with the base of the HTML tree, this gives us a full filename beginning with a slash. The HTML tree is the directory on the server in which the HTML files are located. Normally all files in this directory can be accessed by anybody by using a WWW browser. So if a user requests 'index.html' then that file name is first added to BASE (/home/hubbe/www/html/ in this case) and if that file exists it will be returned to the browser.

\[
\text{if(!file::open(input, } "r") )
\{
\]

Try opening the file in read-only mode. If this fails, try opening NOFILE instead. Opening the file will enable us to read it later.

\[
\text{if(!file::open(NOFILE, } "r") )
\{
\]

If this fails too. Write an error message and destruct this object.

\[
\text{werror("Couldn't find default file.\n");}
\]

\[
\text{destruct(this_object());}
\]

\[
\text{return;}
\]

\[
\}
\]

Ok, now we set up the socket so we can write the data back.

\[
\text{socket::set_buffer(65536, } "w") ;
\]

Set the buffer size to 64 kilobytes.

\[
\text{socket::set_nonblocking(0, write_callback, 0);} \\
\]

Make it so that write_callback is called when it is time to write more data to the socket.

\[
\text{write_callback();}
\]

Jump-start the writing.
That was the end of read_callback().

This function is called if the connection is closed while we are reading from the socket.

```cpp
void selfdestruct() { destruct(this_object()); }
```

This function is called when the program is instantiated. It is used to set up data the way we want it. Extra arguments to clone() will be sent to this function. In this case it is the object representing the new connection.

```cpp
void create(object f)
{
    socket::assign(f);
}
```

We insert the data from the file f into 'socket'.

```cpp
socket::set_nonblocking(read_callback, 0, selfdestruct);
```

Then we set up the callback functions and sets the file nonblocking. Nonblocking mode means that read() and write() will rather return that wait for I/O to finish. Then we sit back and wait for read_callback to be called.

```cpp
}
```

End of create()

Next we define the function called when someone connects.

```cpp
void accept_callback()
{
    object tmp_output;
}
```

This creates a local variable of type 'object'. An object variable can contain a clone of any program. Pike does not consider clones of different programs different types. This also means that function calls to objects have to be resolved at run time.
The function accept clones a Stdio.File and makes this equal to the newly connected socket.

```c
if(!tmp_output) return;
```

If it failed we just return.

```c
output_class(tmp_output);
```

Otherwise we clone an instance of 'output_class' and let it take care of the connection. Each clone of output_class will have its own set of global variables, which will enable many connections to be active at the same time without data being mixed up. Note that the programs will not actually run simultaneously though.

```c
destruct(tmp_output);
```

Destruct the object returned by accept(), output_class has already copied the contents of this object.

```c
}
```

Then there is main, the function that gets it all started.

```c
int main(int argc, array(string) argv)
{
    werror("Starting minimal httpd\n");
    Write an encouraging message to stderr.
    if(!bind(PORT, accept_callback))
    {
        werror("Failed to open socket (already bound?)\n");
        return 17;
    }
```

Bind PORT and set it up to call accept_callback as soon as someone connects to it. If the bind() fails we write an error message and return the 17 to indicate failure.

```c
    return - 17; /* Keep going */
```
If everything went ok, we return -17, any negative value returned by main() means that the program WON’T exit, it will hang around waiting for events instead. (like someone connecting)

That’s it, this simple program can be used as the basis for a simple WWW-server. Note that today most WWW servers are very complicated programs, and the above program can never replace a modern WWW server. However, it is very fast if you only want a couple of web pages and have a slow machine available for the server.
Chapter 10

Threads

Threads are used to run several Pike functions at the same time without having to start several Pike processes. Using threads often simplifies coding and because the threads are within the same process, data can be shared or sent to other threads very fast. Threads are not supported on all systems, you may test if you have thread support with the preprocessor construction `#if constant(thread_create)`. Pike needs POSIX or UNIX thread support when compiled to support threads.

10.1 Starting a thread

Starting a thread is very easy. You simply call `thread_create` with a function pointer and any arguments it needs and that function will be executed in a separate thread. The function `thread_create` will return immediately and both the calling function and the called function will execute at the same time. Example:

```pke
void foo(int x)
{
    for(int e=0;e<5;e++)
    {
        sleep(1);
        write("Hello from thread "+x+".\n");
    }
}

int main()
{
    thread_create(foo, 2);
    thread_create(foo, 3);
    foo(1);
}
```
CHAPTER 10. THREADS

This may all seem very simple, but there are a few complications to watch out for:

10.2 Threads reference section

This section describes all thread-related functions and classes.

Thread.thread_create - create a thread

object thread_create(function f, mixed ... args);

Description
This function creates a new thread which will run simultaneously to the rest of the program. The new thread will call the function f with the arguments args. When f returns the thread will cease to exist. All Pike functions are 'thread safe' meaning that running a function at the same time from different threads will not corrupt any internal data in the Pike process. The returned value will be the same as the return value of this_thread() for the new thread.

Note
This function is only available on systems with POSIX or UNIX threads support.

See Also
Thread.Mutex, Thread.Condition and Thread.this_thread

Thread.this_thread - return thread id

object this_thread();

Description
This function returns the object that identifies this thread.

See Also
Thread.thread_create

Thread.all_threads - return all thread ids

array(object) all_threads();

Description
This function returns an array with the thread ids of all threads.

See Also
Thread.thread_create
**Thread.Mutex - mutex locks**

**Description**
Thread.Mutex is a pre-compiled Pike program that implements mutual exclusion locks. Mutex locks are used to prevent multiple threads from simultaneously execute sections of code which access or change shared data. The basic operations for a mutex is locking and unlocking, if a thread attempts to lock an already locked mutex the thread will sleep until the mutex is unlocked.

**Note**
Mutex locks are only available on systems with POSIX or UNIX threads support.

In POSIX threads, mutex locks can only be unlocked by the same thread that locked them. In Pike any thread can unlock a locked mutex.

**Example**

```pike
/* This simple program can be used to exchange data between two programs. It is similar to Thread.Fifo, but can only hold one element of data. */

inherit Thread.Mutex : r_mutex;
inherit Thread.Mutex : w_mutex;
object r_lock=r_mutex::lock();
object w_lock;
mixed storage;

void write(mixed data)
{
    w_lock=w_mutex::lock();
    storage=data;
    destruct(r_lock);
}

mixed read()
{
    mixed tmp;
    r_lock=r_mutex::lock();
    tmp=storage;
    storage=0;
    destruct(w_lock);
    return tmp;
}
```
Thread.Mutex.lock - lock the mutex

object lock();

Description
This function attempts to lock the mutex, if the mutex is already locked the current thread will sleep until the lock is unlocked by some other thread. The value returned is the 'key' to the lock. When the key is destructed or has no more references the lock will automatically be unlocked. The key will also be destructed if the lock is destructed.

Thread.Mutex.trylock - try to lock the mutex

object trylock();

Description
This function performs the same operation as lock(), but if the mutex is already locked zero will be returned instead of sleeping until the lock is unlocked.

Thread.Condition - condition variables

Description
Thread.Condition is a pre-compiled Pike program that implements condition variables. Condition variables are used by threaded programs to wait for events happening in other threads.

Note
Condition variables are only available on systems with POSIX or UNIX threads support.

Example

// This program implements a fifo that can be used to send
// data between two threads.
inherit Thread.Condition : r_cond;
inherit Thread.Condition: w_cond;
inherit Thread.Mutex: lock;

array buffer = allocate(128);
int r_ptr, w_ptr;

int query_messages() { return w_ptr - r_ptr; }

// This function reads one mixed value from the fifo.
// If no values are available it blocks until a write has been done.
mixed read()
{
    mixed tmp;
    // We use this mutex lock to make sure no write() is executed
    // between the query_messages and the wait() call. If it did
    // we would wind up in a deadlock.
    object key=lock::lock();
10.2. THREADS REFERENCE SECTION

while(!query_messages()) r_cond::wait(key);
    tmp=buffer[r_ptr++ % sizeof(buffer)];
    w_cond::signal();
    return tmp;
}

// This function pushes one mixed value on the fifo.
// If the fifo is full it blocks until a value has been read.
void write(mixed v)
{
    object key=lock::lock();
    while(query_messages() == sizeof(buffer)) w_cond::wait(key);
    buffer[w_ptr++ % sizeof(buffer)]=v;
    r_cond::signal();
}

See Also
Thread.Mutex

Thread.Condition.wait - wait for condition

void wait();
void wait(object mutex_key);

Description
This function makes the current thread sleep until the condition variable is
signalled. The optional argument should be the 'key' to a mutex lock. If present
the mutex lock will be unlocked before waiting for the condition in one atomic
operation. After waiting for the condition the mutex referenced by mutex_key
will be re-locked.

See Also
Thread.Mutex->lock

Thread.Condition.signal - signal a condition variable

void signal();

Description
Signal wakes up one of the threads currently waiting for the condition.

Bugs
It sometimes wakes up more than one thread.

Thread.Condition.broadcast - signal all waiting threads

void broadcast();

Description
This function wakes up all threads currently waiting for this condition.

Thread.Fifo - first in, first out object
**Description**
Thread.Fifo implements a fixed length fifo. A fifo is a queue of values and is often used as a stream of data between two threads.

**Note**
Fifos are only available on systems with POSIX threads support.

**See Also**
Thread.Queue

<table>
<thead>
<tr>
<th>Thread.Fifo.create - initialize the fifo</th>
</tr>
</thead>
<tbody>
<tr>
<td>void create(int size);</td>
</tr>
<tr>
<td>object(Thread.Fifo) Thread.Fifo();</td>
</tr>
<tr>
<td>object(Thread.Fifo) Thread.Fifo(int size);</td>
</tr>
</tbody>
</table>

**Description**
The function `create()` is called when the fifo is cloned, if the optional size argument is present it sets how many values can be written to the fifo without blocking. The default size is 128.

<table>
<thead>
<tr>
<th>Thread.Fifo.write - queue a value</th>
</tr>
</thead>
<tbody>
<tr>
<td>void write(mixed value);</td>
</tr>
</tbody>
</table>

**Description**
This function puts a value last in the fifo. If there is no more room in the fifo the current thread will sleep until space is available.

<table>
<thead>
<tr>
<th>Thread.Fifo.read - read a value from the fifo</th>
</tr>
</thead>
<tbody>
<tr>
<td>mixed read();</td>
</tr>
</tbody>
</table>

**Description**
This function retrieves a value from the fifo. Values will be returned in the order they were written. If there are no values present in the fifo the current thread will sleep until some other thread writes a value to the fifo.

<table>
<thead>
<tr>
<th>Thread.Fifo.size - return number of values in fifo</th>
</tr>
</thead>
<tbody>
<tr>
<td>int size();</td>
</tr>
</tbody>
</table>

**Description**
This function returns how many values are currently in the fifo.

<table>
<thead>
<tr>
<th>Thread.Queue - a queue of values</th>
</tr>
</thead>
</table>

**Description**
Thread.Queue implements a queue, or a pipeline. The main difference between Thread.Queue and Thread.Fifo is that queues will never block in write(), only allocate more memory.

**Note**
Queues are only available on systems with POSIX or UNIX threads support.
10.2. THREADS REFERENCE SECTION

See Also
Thread.Fifo

Thread.Queue.write - queue a value

```c
void write(mixed value);
```

Description
This function puts a value last in the queue. If the queue is too small to hold
the value the queue will be expanded to make room for it.

Thread.Queue.read - read a value from the queue

```c
mixed read();
```

Description
This function retrieves a value from the queue. Values will be returned in the
order they were written. If there are no values present in the queue the current
thread will sleep until some other thread writes a value to the queue.

Thread.Queue.size - return number of values in queue

```c
int queue->size();
```

Description
This function returns how many values are currently in the queue.

Thread.thread_local - Thread local variable class

Description
This class allows you to have variables which are separate for each thread that
uses it. It has two methods: get and set. A value stored in an instance of
thread_local can only be retrieved by that same thread.

Thread.thread_local.thread_local.set - Set the thread local value

```c
mixed set(mixed value);
```

Description
This sets the value returned by the get method. Note that this value can only
be retrieved by the same thread. Calling this method does not affect the value
returned by get when called by another thread.

Thread.thread_local.thread_local.get - Get the thread_local value

```c
mixed set(mixed value);
```

Description
This returns the value previously stored in the thread_local by the set method
by this thread.
10.3 Threads example

Let’s look at an example of how to work with threads. This program is the same minimal WWW server as in chapter 8.5 but it has been re-written to use threads, as you can see it is a lot smaller this way. This is because we can use blocking I/O operations instead of non-blocking and callbacks. This also makes the program much easier to follow:

#!/usr/local/bin/pike

/*@ A very small threaded httpd capable of fetching files only. *
 * Written by Fredrik H¨ubinette as a demonstration of Pike *
*/

import Thread;
inherit Stdio.Port;

/*@ number of bytes to read for each write */
#define BLOCK 16384

/*@ Where do we have the html files ? */
#define BASE "/home/hubbe/pike/src/"

/*@ File to return when we can’t find the file requested */
#define NOFILE "/home/hubbe/www/html/nofile.html"

/*@ Port to open */
#define PORT 1905

/*@ Number of threads to start */
#define THREADS 5

// There will be one of these for each thread
class worker
{
  inherit Stdio.FILE : socket; // For communication with the browser
  inherit Stdio.File : file; // For reading the file from disc

  void create(function accept)
  {
    string cmd, input, tmp;

    while(1)
    {
      socket::close(); // Close previous connection
      file::close();

      object o=accept(); // Accept a connection
      if(!o) continue;
      socket::assign(o);
    }
  }
}
10.3. THREADS EXAMPLE

destruct(o);

// Read request
sscanf(socket::gets(),"%s %s%[\012\015 \t\n\0]",cmd, input);
if(cmd!="GET")
{
    werror("Only method GET is supported.\n");
    continue;
}

// Open the requested file
sscanf(input,"%s/\s%",input);
input=BASE+combine_path("/",input);

if(!file::open(input,"r"))
{
    if(!file::open(NOFILE,"r"))
    {
        werror("Couldn't find default file.\n");
        continue;
    }
}

// Copy data to socket
while(socket::write(file::read(BLOCK))==BLOCK);
}
}

int main(int argc, array(string) argv)
{
    werror("Starting minimal threaded httpd\n");

    // Bind the port, don't set it nonblocking
    if(!bind(PORT))
    {
        werror("Failed to open socket (already bound?)\n");
        return 17;
    }

    // Start worker threads
    for(int e=1;e<THREADS;e++) thread_create(worker,accept);
    worker(accept);
}

As stated in the beginning of this chapter; Pike threads are only available on some UNIX systems. The above example does not work if your system does not have threads.
Chapter 11

Modules for specific data types

There are a few modules that provide extra functions that operate specifically on one data type. These modules have the same name as the data type, but are capitalized so you can tell the difference. At the time of writing, the only such modules are String and Array, but more are expected to show up in the future.

11.1 String

The module String contains some extra string functionality which is not always used. These functions are mostly implemented in Pike as a complement to those written in C.

**String.fuzzymatch** - make a fuzzy compare of two strings

```pike
int fuzzymatch(string word1, string word2);
```

**Description**
This function compares two strings using a fuzzy matching routine. The higher the resulting value, the better the strings match.

**Example**
```
> fuzzymatch("cat", "hat");
Result: 66
> fuzzymatch("cat", "dog");
Result: 0
> fuzzymatch("United States", "United Nations");
Result: 70
```

**See Also**
Array.diff, Array.diff_compare_table and Array.diff_longest_sequence

**String.implode_nicely** - make an English comma separated list

```pike
string implode_nicely(array(string|float|int) words, string|void separator);
```

133
**Description**
This function implode a list of words to a readable string. If the separator is omitted, the default is "and". If the words are numbers they are convert to strings first.

**Example**
```plaintext
> implode_nicely({'green'});
Result: green
> implode_nicely({'green','blue'});
Result: green and blue
> implode_nicely({'green','blue','white'});
Result: green, blue and white
> implode_nicely({'green','blue','white'},"or");
Result: green, blue or white
> implode_nicely({1,2,3},"or even");
Result: 1, 2 or even 3
```

**See Also**
`
String.capitalize - capitalize a string

string capitalize(string str);

Description
Convert the first character in str to upper case, and return the new string.

See Also
lower_case and upper_case

String.common_prefix - find the longest common beginning

string common_prefix(array(string) strs);

Description
Find the longest common beginning from an array of strings.

Example
```plaintext
> String.common_prefix({'muzzle', 'muzzy'});
Result: "muzz"
> String.common_prefix({'labyrinth', 'diatom'});
Result: ""
> String.common_prefix({});
Result: ""
```

String.sillycaps - Sillycaps A String

string sillycaps(string str);
11.2. ARRAY

Description
Convert the first character in each word (separated by spaces) in str to upper case, and return the new string.

**String.strmult** - multiply strings

```pascal
string strmult(string s, int num);
```

Description
This function multiplies 's' by 'num'. The return value is the same as appending 's' to an empty string 'num' times.

**String.count** - count needles in a haystack string

```pascal
string count(string haystack, string needle);
```

Description
This function counts the number of times the needle can be found in haystack. Intersections between needles are not counted, ie count("...","..") is 2.

**String.width** - return width of string

```pascal
int width(string s);
```

Description
Returns the width in bits (8, 16 or 32) of the widest character in s.

**String.trim_whites** - trim spaces and tabs from a string

```pascal
string trim_whites(string s);
```

Description
Trim leading and trailing spaces and tabs from the string s.

**String.trim_all_whites** - trim all white spaces from a string

```pascal
string trim_all_whites(string s);
```

Description
Trim leading and trailing white space characters (" 	"

11.2 Array

As with String these functions are mostly Pike functions written to supplement those written in C.

**Array.diff** - gives the difference of two arrays

```pascal
array(array(array)) diff(array a, array b);
```
Description
Calculates which parts of the arrays that are common to both, and which parts that are not. Returns an array with two elements, the first is an array of parts in array \texttt{a}, and the second is an array of parts in array \texttt{b}.

Example


gotchas_shell> Array.diff("Hello world!/"","Help!/");
Result: ( { /* 2 elements */
  ( { /* 3 elements */
    ( { /* 3 elements */
        "H",
        "e",
        "l"
      ),
    ( { /* 8 elements */
        "l",
        "o",
        =",
        "w",
        "o",
        "r",
        "l",
        "d"
      ),
    ( { /* 1 elements */
        "!"
      )
    ),
  ( { /* 3 elements */
    ( { /* 3 elements */
        "H",
        "e",
        "l"
      ),
    ( { /* 1 elements */
        "p"
      ),
    ( { /* 1 elements */
        "!"
      )
    )
  }));
}

See Also
Array.diff_compare_table, Array.diff_longest_sequence and String.fuzzymatch
11.2. ARRAY

Array.diff_compare_table - gives the comparison-table used by Array.diff()
array(array(int)) diff_compare_table(array a, array b);

Description
Returns an array which maps from index in a to corresponding indices in b.

Example

> Array.diff_compare_table("Hello world!/\n","Help!/\n");
Result: ({{ /* 12 elements */
  ({{ /* 1 elements */
    0
  }},
  ({{ /* 1 elements */
    1
  }},
  ({{ /* 1 elements */
    2
  }},
  ({{ /* 1 elements */
    2
  }},
  ({{ }},
  ({{ }},
  ({{ }},
  ({{ }},
  ({{ /* 1 elements */
    2
  }},
  ({{ }},
  ({{ /* 1 elements */
    4
  }},
  )
  )
  )
  )
  )
  )
  )
  )})

See Also
Array.diff, Array.diff_longest_sequence and String.fuzzymatch

Array.diff_longest_sequence - gives the longest common sequence of two arrays
array(int) diff_longest_sequence(array a, array b);

Description
Gives the longest sequence of indices in b that have corresponding values in the
same order in `a`.

*Example*

```javascript
> Array.diff_longest_sequence("Hello world!"/"","Help!"/"");
Result: (/* 4 elements */
    0,
    1,
    2,
    4
)
```

*See Also*

`Array.diff`, `Array.diff_compare_table` and `String.fuzzymatch`

**Array.everynth** - return every nth element of an array

```javascript
array(mixed) Array.everynth(array(mixed) arr1, void|int nth, void|int start);
```

*Description*

This function returns an array with every nth element of an other array. If nth is zero, every second element is returned.

*Example*

```javascript
> Array.everynth({"1","2","3","4","5","6"},2,1);
```

Result: (/* 3 elements */
    "2",
    "4",
    "6"
)

*See Also*

`Array.splice` and `'`

**Array.filter** - filter an array or mapping through a function

```javascript
array filter(array arr,function fun,mixed ... args);
array filter(array(object) arr,string fun,mixed ... args);
array filter(array(function) arr,-1,mixed ... args);
```

*Description*

First syntax:

Filter array returns an array holding the items of arr for which fun returns true.
Second syntax:
Filter array calls fun in all the objects in the array arr, and return all objects that returned true.

Third syntax:
Filter array calls all function pointers in the array arr, and return all that returned true.

See Also
Array.sum_arrays and Array.map

Array.longest_ordered_sequence - find the longest ordered sequence of elements
array(int) longest_ordered_sequence(array a);

Description
This function returns an array of the indices in the longest ordered sequence of elements in the array.

Example

```plaintext
> array a = ([1,2,3,4,2,3,5,6]);
Result: ([/* 8 elements */
  1,
  2,
  3,
  4,
  2,
  3,
  5,
  6
]);
> array seq = Array.longest_ordered_sequence(a);
Result: ([/* 6 elements */
  0,
  1,
  4,
  5,
  6,
  7
]);
> rows(a,seq);
Result: ([/* 6 elements */
  1,
  2,
  2,
  3,
  5,
  6
])
```
See Also

Array.diff

Array.map - map an array or mapping over a function

array map(array arr, function fun, mixed ... args);
array map(array(object) arr, string fun, mixed ... args);
array map(array(function) arr, -1, mixed ... arg);

Description

First syntax:
Map array returns an array holding the items of arr mapped through the function fun. i.e. arr[x]=fun(arr[x], @args) for all x.

Second syntax:
Map array calls function fun in all objects in the array arr. i.e. arr[x]=arr[x]->fun(@args);

Third syntax:
Map array calls the functions in the array arr: arr[x]=arr[x]->fun(@ args);

See Also

Array.sum_arrays and Array.filter

Array.permute - Give a specified permutation of an array

array permute(array in, int number);

Description

permute gives you a selected permutation of an array. The numbers of permutations equal sizeof(in)! (the factorial of the size of the given array).

Example

Array.permute( ( { 1,2,3 } ), 0 ) -> ( { 1,2,3 } )
Array.permute( ( { 1,2,3 } ), 3 ) -> ( { 1,3,2 } )

See Also

Array.shuffle

Array.reduce - Iteratively apply a function on an array

mixed reduce(function f, array arr, void|mixed zero);

Description

reduce sends the first two elements in arr to f, then the result and the next element in arr to f and so on. Then it returns the result. The function will return zero if arr is the empty array. If arr has size 1 it will return the element in arr.

Example
11.2. ARRAY

Array.reduce(aggregate, indices(allocate(4))) ->
({ { { 0, 1 }, 2 }, 3 })

Array.reduce('+', {}, "FOO") ->
"FOO"

Array.reduce(lambda(int a, int b) {
    while(b) { int t=b; b=a%b; a=t; }
    return a;
}, ((7191, 21573, 64719, 33694)))

-> 17

See Also
Array.reduce

Array.rreduce - Iteratively apply a function on an array backwards

mixed rreduce(function f, array arr, void|mixed zero);

Description
reduce sends the last two elements in arr to f, then the third last element in
arr and the result to f and so on. Then it returns the result. The function will
return zero if arr is the empty array. If arr has size 1 it will return the element
in arr.

Example

Array.rreduce(aggregate, indices(allocate(4))) ->
({ 0, ({ 1, {{ 2, 3 }} } }) })

See Also
Array.reduce

Array.search_array - search for something in an array

int search_array(array arr, function fun, mixed arg, ...);
int search_array(array(object) arr, string fun, mixed arg, ...);
int search_array(array(function) arr,-1,mixed arg, ...);

Description
search_array works like map_array, only it returns the index of the first call that
returned true instead or returning an array of the returned values. If no call
returns true, -1 is returned.

See Also
Array.sum_arrays and Array.filter
**Array.shuffle** - Random-shuffle an array

```
array shuffle(array in);
```

**Description**

`shuffle` gives back the same elements, but by random order.

**Example**

```
Array.shuffle( ("this","is","an","ordered","array") ) -> ("is","ordered","array","this","an")
```

**See Also**

Array.permute

**Array.sort_array** - sort an array

```
array sort_array(array arr, function fun, mixed ... args);
```

**Description**

This function sorts an array after a compare-function `fun` which takes two arguments and should return 1 if the first argument is larger than the second. The rest of the arguments `args` will be sent as 3rd, 4th etc. argument to `fun`. If `fun` is omitted, `¿` is used instead.

**See Also**

Array.map and sort

**Array.splice** - splice two or more arrays

```
array(mixed) Array.splice(array(mixed) arr1, array(mixed) arr2, array(mixed) ...);
```

**Description**

This function splice two or more arrays together. This means that the the array becomes an array of the first element of the first given array, the first argument in next array and so on for all arrays. Then the second elements are added.

**Example**

```
> Array.splice(((1,2,3)),("1","2","3"),{"a","b","c"});
```

Result: ( /* 9 elements */

```
1,
"1",
"a",
2,
"2",
"b",
3,
"3",
"c"
})
```
> Array.splice({1,2,3}),{"1","2","3"},{"a","b"});
Result: ({{ /* 9 elements */
  1,
  "1",
  "a",
  2,
  "2",
  "b",
}})

See Also
‘/’, ‘*’, ‘+’, ‘-’ and Array.everynth

Array.sum_arrays - map any number of arrays over a function
array sum_arrays(function fun,array arr1,...);

Description
Works like this:

array sum_arrays(function fun,array arr1,...)
{
  int e;
  array res=allocate(sizeof(arr1));
  for(e=0;e<sizeof(arr1);e++)
  {
    res[e]=fun(arr1[e],arr2[e],...);
  }
  return res;
}

Simple eh?

See Also
Array.map, Array.filter and Array.search_array

Array.uniq - remove elements that are duplicates
array uniq(array a);

Description
This function returns an copy of the array a with all duplicate values removed. The order of the values in the result is undefined.
Chapter 12

Image

The Image module is used to manipulate bit-mapped color images. It can read PPM images and do various manipulations, or it can be used to create completely new images. The created images can be saved as PPM or converted to GIF.

All images handled by this module are stored as 24-bit RGB images. This means that a 1024 pixel wide and 1024 pixel high image will use 1024*1024*3 bytes = 3 megabytes. It is quite easy to mess up and use up all the memory by giving the wrong argument to one of the scaling functions.

Most functions in this module work by creating a new Image and then returning that instead of changing the Image you are working with. This makes it possible to share the same image between many variables without having to worry that it will be changed by accident. This can reduce the amount of memory used.

Many functions in this module work with the 'current color', this can be thought of as the background color if you wish. To change the current color you use 'setcolor'.

Let's look at an example of how this can be used:

```pike
#!/usr/local/bin/pike

int main()
{
  write("Content-type: image/gif\n\n");
  object font=Image.font();
  font->load("testfont");
  object image=font->write(ctime(time));
  write(Image.GIF.encode(image));
}
```

This very simple example can be used as a CGI script to produce a gif image which says what time it is in white text on a black background.

Description
This module adds image-drawing and -manipulating capabilities to pike.

- **Image.Image** Basic image manipulation
- **Image.Font** Creating images from text
- **Image.Colortable** Color reduction, quantisation and dither
- **Image.Color** Color names, objects and conversion

### Advanced

- **Image.GIF** GIF encoding/decoding capabilities
- **Image.JPEG** JPEG encoding/decoding capabilities (needs libjpeg)
- **Image.PNG** PNG encoding/decoding capabilities (needs libz)
- **Image.PNM** PNM (PBM/PGM/PPM) encoding/decoding capabilities
- **Image.XFace** XFace encoding/decoding capabilities (needs libgmp)
- **Image.XWD** XWD (X-windows dump) decoding capabilities

---

$Id: module.pmod,v 1.8 2000/03/22 18:12:19 peter Exp $

**Note**

Image module documentation is based on these file versions:

---

**Image.Lay**

Image.Lay lay(array(Image.Layer|mapping))

*Description*

Combine layers.

*Returns*

a new layer object.

*See Also*

Image.Lay

- **Image.load**
- **Image.load_layer**
### 12.1. Image.Image

**Image.load**

object(Image.Image) load()
object(Image.Image) load(object file)
object(Image.Image) load(string filename)
object(Image.Layer) load_layer()
object(Image.Layer) load_layer(object file)
object(Image.Layer) load_layer(string filename)
mapping _load()
mapping _load(object file)
mapping _load(string filename)

**Description**

Helper function to load an image from a file. If no filename is given, Stdio.stdin is used. The result is the same as from the decode functions in Image.ANY.

**Note**

All data is read, i.e. nothing happens until the file is closed. Throws upon error.

### 12.1 Image.Image

**Image.Image**

**Description**

The main object of the Image module, this object is used as drawing area, mask or result of operations.

- **basic:**
  - clear, clone, create, xsize, ysize

- **plain drawing:**
  - box, circle, getpixel, line, setcolor, setpixel, threshold, polyfill

- **operators:**
  - ‘&’, ‘+’, ‘-’, ‘==’, ‘>’, ‘<’

- **pasting images:**
  - paste, paste_alpha, paste_alpha_color, paste_mask

- **getting subimages, scaling, rotating:**
  - autocrop, clone, copy, dct, mirrorx, rotate, rotate_ccw, rotate_cw, rotate_expand, scale, skewx, skewx_expand, skewy, skewy_expand

- **calculation by pixels:**
  - apply_matrix, change_color, color, distancesq, grey, invert, modify_by_intensity, outline select_from, rgb_to_hsv, hsv_to_rgb

- **average, max, min, sum, sumf, find_min, find_max**

- **special pattern drawing:**
  - noise, turbulence, test, tuned_box, gradients, random
See Also
Image, Image\_Font, Image\_Colortable and Image\_X

**Image\_Image\_apply\_matrix**

object apply\_matrix(array(array(int|array(int)))\_matrix)
object apply\_matrix(array(array(int|array(int)))\_matrix, int\_r, int\_g, int\_b)
object apply\_matrix(array(array(int|array(int)))\_matrix, int\_r, int\_g, int\_b, int|float\_div)

**Description**
Applies a pixel-transform matrix, or filter, to the image.

\[
pixel(x,y) = \text{base} + \frac{1}{k} \sum_{k=0}^{2} \sum_{l=0}^{2} \pixel(x+k-1,y+l-1) \cdot \matrix(k,l)
\]

1/k is sum of matrix, or sum of matrix multiplied with div. base is given by r,g,b and is normally black.

blur (ie a 2d gauss function):

\[
\{(1,2,1),
(2,5,2),
(1,2,1))\}
\]

original

This function is not very fast.

**Arguments**

**Returns**
the new image object

**Image\_Image\_apply\_max**

object apply\_max(array(array(int|array(int)))\_matrix)
object apply\_max(array(array(int|array(int)))\_matrix, int\_r, int\_g, int\_b)
object apply\_max(array(array(int|array(int)))\_matrix, int\_r, int\_g, int\_b, int|float\_div)
**Description**
This is the same as apply_matrix, but it uses the maximum instead.

This function is not very fast.

**Arguments**

**Returns**
the new image object

**Note**
experimental status; may not be exact the same output in later versions

**Image.Image.autocrop**

object autocrop()

object autocrop(int border)

object autocrop(int border, Color color)

object autocrop(int border, int left, int right, int top, int bottom)

object autocrop(int border, int left, int right, int top, int bottom, Color color)

array(int) find_autocrop()

array(int) find_autocrop(int border)

array(int) find_autocrop(int border, int left, int right, int top, int bottom)

**Description**
Removes ”unneccesary” borders around the image, adds one of its own if wanted to, in selected directions.

”Unneccesary” is all pixels that are equal – ie if all the same pixels to the left are the same color, that column of pixels are removed.

The find_autocrop() function simply returns x1,y1,x2,y2 for the kept area. (This can be used with copy later.)

**Arguments**

**Returns**
the new image object

**See Also**
copy

**Image.Image.max**

**Image.Image.min**

**Image.Image.sumf**

**Image.Image.sum**
**Image.Image.average**

array(float) average()
array(int) min()
array(int) max()
array(int) sum()
array(float) sumf()

*Description*
Gives back the average, minimum, maximum color value, and the sum of all pixel’s color value.

*Note*
sum() values can wrap! Most systems only have 31 bits available for positive integers. (Meaning, be careful with images that have more than 8425104 pixels.)

average() and sumf() may also wrap, but on a line basis. (Meaning, be careful with images that are wider than 8425104 pixels.) These functions may have a precision problem instead, during to limits in the 'double' C type and/or 'float' Pike type.

**Image.Image.bitscale**

object bitscale(float factor)
object bitscale(float xfactor, float yfactor)

*Description*
scales the image with a factor, without smoothing. This routine is faster than scale, but gives less correct results

*Arguments*

*Returns*
the new image object

**Image.Image.bitscale**

object bitscale(int newxsize, int newysize)
object bitscale(0, int newysize)
object bitscale(int newxsize, 0)

*Description*
scales the image to a specified new size, if one of newxsize or newysize is 0, the image aspect ratio is preserved.

*Arguments*

*Returns*
the new image object

*Note*
resulting image will be 1x1 pixels, at least
12.1. **Image.Image.box**

```c
object box(int x1, int y1, int x2, int y2)
object box(int x1, int y1, int x2, int y2, int r, int g, int b)
object box(int x1, int y1, int x2, int y2, int r, int g, int b, int alpha)
```

*Description*

Draws a filled rectangle on the image.

Arguments

Returns the object called

12.1. **Image.Image.cast**

```c
string cast(string type)
```

*Description*

Cast the image to another datatype. Currently supported are string ("rgbrgrbgrbg...") and array (double array of Image.Color objects).

See Also

Image.Color and Image.X

12.1. **Image.Image.change_color**

```c
object change_color(int tor, int tog, int tob)
object change_color(int fromr, int fromg, int fromb, int tor, int tog, int tob)
```

*Description*

Changes one color (exakt match) to another. If non-exakt-match is preferred, check distancesqand paste_alpha_color.

Arguments

Returns

a new (the destination) image object
**Image.Image.circle**

```plaintext
object circle(int x, int y, int rx, int ry)
object circle(int x, int y, int rx, int ry, int r, int g, int b)
object circle(int x, int y, int rx, int ry, int r, int g, int b, int alpha)
```

**Description**

Draws a circle on the image. The circle is not antialiased.

![Original image](image1.png)  ![Circle image](image2.png)

**Arguments**

**Returns**

the object called

**Image.Image.clear**

```plaintext
void clear()
void clear(int r, int g, int b)
void clear(int r, int g, int b, int alpha)
```

**Description**

gives a new, cleared image with the same size of drawing area

![Original image](image3.png)  ![Clear image](image4.png)

**Arguments**

**See Also**
copy and clone

*Image.Image.clone*

object clone()

object clone(int xsize, int ysize)

object clone(int xsize, int ysize, int r, int g, int b)

object clone(int xsize, int ysize, int r, int g, int b, int alpha)

*Description*

Copies to or initialize a new image object.

*Arguments*

*Returns*

the new object

*See Also*

copy and create

*Image.Image.color*

object color()

object color(int value)

object color(int r, int g, int b)

*Description*

Colorize an image.

The red, green and blue values of the pixels are multiplied with the given value(s). This works best on a grey image...

The result is divided by 255, giving correct pixel values.

If no arguments are given, the current color is used as factors.
CHAPTER 12. IMAGE

Arguments

Returns
the new image object

See Also
grey, ‘* and modify_by_intensity

Image.Image.copy

object copy()
object copy(int x1, int y1, int x2, int y2)
object copy(int x1, int y1, int x2, int y2, int r, int g, int b)
object copy(int x1, int y1, int x2, int y2, int r, int g, int b, int alpha)

Description
Copies this part of the image. The requested area can be smaller, giving a cropped image, or bigger - the new area will be filled with the given or current color.

Arguments

Returns
a new image object
Note
clone(void) and copy(void) does the same operation

See Also
close and autocrop

Image.Image.create

void create()
void create(int xsize, int ysize)
void create(int xsize, int ysize, Color color)
void create(int xsize, int ysize, int r, int g, int b)
void create(int xsize, int ysize, int r, int g, int b, int alpha)
void create(int xsize, int ysize, string method, method...)

Description
Initializes a new image object.

The image can also be calculated from some special methods, for convenience:

channel modes; followed by a number of 1-char-per-pixel strings
or image objects (where red channel will be used),
or an integer value:
"grey" : make a grey image (needs 1 source: grey)
"rgb" : make an rgb image (needs 3 sources: red, green and blue)
"cmyk" : make a rgb image from cmyk (cyan, magenta, yellow, black)

Arguments

Bugs
SIGSEGV can be caused if the size is too big, due to uncheked overflow -
(xsize*ysize)&MAXINT is small enough to allocate.

See Also
copy, clone and Image.Image
**Image.Image.dct**

object dct(int newx, int newy)

*Description*
Scales the image to a new size.

Method for scaling is rather complex; the image is transformed via a cosine transform, and then resampled back.

This gives a quality-conserving upscale, but the algorithm used is \( n^2n+n^2m \), where \( n \) and \( m \) is pixels in the original and new image.

Recommended wrapping algorithm is to scale overlapping parts of the image-to-be-scaled.

This functionality is actually added as an true experiment, but works...

*Arguments*

*Returns*
the new image object

*Note*
Do NOT use this function if you don’t know what you’re dealing with! Read some signal theory first...

It write’s dots on stderr, to indicate some sort of progress. It doesn’t use any fct (compare: fft) algorithms.

**Image.Image.distancesq**

object distancesq()
object distancesq(int r, int g, int b)

*Description*
Makes an grey-scale image, for alpha-channel use.

The given value (or current color) are used for coordinates in the color cube. Each resulting pixel is the distance from this point to the source pixel color, in the color cube, squared, rightshifted 8 steps:

\[
p = \text{pixel color} \\
o = \text{given color} \\
d = \text{destination pixel} \\
d.\text{red}=d.\text{blue}=d.\text{green}= \\
((o.\text{red}-p.\text{red})^2+(o.\text{green}-p.\text{green})^2+(o.\text{blue}-p.\text{blue})^2)\gg8
\]
Arguments

Returns
the new image object

See Also
select_from

Image.Image.find_min,
Image.Image.find_max

array(int) find_min()
array(int) find_max()
array(int) find_min(int r, int g, int b)
array(int) find_max(int r, int g, int b)

Description
Gives back the position of the minimum or maximum pixel value, weighted to grey.

Arguments

Image.Image.gamma

object gamma(float g)
object gamma(float gred, gggreen, gblue)

Description
Calculate pixels in image by gamma curve.

Intensity of new pixels are calculated by:

\[ i' = i^g \]

For example, you are viewing your image on a screen with gamma 2.2. To correct your image to the correct gamma value, do something like:

my_display_image(my_image())->gamma(1/2.2);

Arguments
Returns
a new image object

See Also
grey, ‘*’ and color

Image.Image.getpixel

array(int) getpixel(int x, int y)

Description

Arguments

Returns
color of the requested pixel – (–int red, int green, int blue”)

Image.Image.gradients

int gradients(array(int) point, …)  
int gradients(array(int) point, …, float grad)

Description

Returns
the new image

Image.Image.grey

object grey()
object grey(int r, int g, int b)

Description
Makes a grey-scale image (with weighted values).
12.1. IMAGE

Arguments

Returns
the new image object

See Also
color, ‘* and modify_by_intensity

Image.Image.rgb_to_hsv, Image.Image.hsv_to_rgb

object rgb.to.hsv()
object hsv.to.rgb()

Description
Converts RGB data to HSV data, or the other way around. When converting
to HSV, the resulting data is stored like this: pixel.r = h; pixel.g = s; pixel.b = v;

When converting to RGB, the input data is assumed to be placed in the pixels
as above.
HSV to RGB calculation:

```plaintext
in = input pixel
out = destination pixel
h = -pos*c_angle*3.1415/(float)NUM_SQUARES;
out.r = (in.b + in.g * cos(in.r));
out.g = (in.b + in.g * cos(in.r + pi*2/3));
out.b = (in.b + in.g * cos(in.r + pi*4/3));
```

RGB to HSV calculation: Hmm.

Example: Nice rainbow.

```plaintext
object i = Image.Image(200, 200);
i = i->tuned_box(0, 0, 200, 200,
({ ({ 255, 255, 128 }), ({ 0, 255, 128 }),
({ 255, 255, 255 }), ({ 0, 255, 255 }))),
->hsv_to_rgb();
```

**Returns**
the new image object

**Image.Image.invert**

object invert()

**Description**
Invert an image. Each pixel value gets to be 255-x, where x is the old value.

```
original  ~i.invert();  ~i.rgb_to_hsv(~i.invert()); ~i.hsv_toRgb();
```

**Returns**
the new image object

**Image.Image.line**

object line(int x1, int y1, int x2, int y2)
object line(int x1, int y1, int x2, int y2, int r, int g, int b)
object line(int x1, int y1, int x2, int y2, int r, int g, int b, int alpha)

**Description**
Draws a line on the image. The line is *not* antialiased.

![original image](image1)
- line-(50,10,-10,50,-255,0,0)

**Arguments**

**Returns**
the object called

**Image.Image.make_ascii**

string make_ascii(object orient1, object orient2, object orient3, object orient4, int|void xsize, int|void ysize)

**Description**
This method creates a string that looks like the image. Example:

```c
//Stina is an image with a cat.
array(object) Stina4=Stina->orient4();
Stina4[1]*=215;
Stina4[3]*=215;
string foo=Stina->make_ascii(*Stina4,40,4,8);
```

**Returns**
some nice ascii-art.

**Note**
*experimental status*; may not be exact the same output in later versions
| / − | / − |
| hue= 0 64 128 192 (=red in an hsv image)

See Also
orient and orient4
Image.Image.map_closest,
Image.Image.select_colors,
Image.Image.map_fast,
Image.Image.map_fs

object map_closest(array(array(int))) colors
object map_fast(array(array(int))) colors
object map_fs(array(array(int))) colors
array select_colors(int, num)

Description
Compatibility functions. Do not use!
Replacement examples:
Old code:

```plaintext
img=map_fs(img->select_colors(200));
```

New code:

```plaintext
img=Image.Colortable(img,200)->floyd_steinberg()->map(img);
```

Old code:

```plaintext
img=map_closest(img->select_colors(17)+({({255,255,255}),({0,0,0})}));
```

New code:

```plaintext
img=Image.Colortable(img,19,({({255,255,255}),({0,0,0})}))->map(img);
```

Image.Image.match

object match(int|float scale, _object needle)
object match(int|float scale, _object needle, _object haystack_cert, _object needle_cert)
object match(int|float scale, _object needle, _object haystack_avoid, _int foo)
object match(int|float scale, _object needle, _object haystack_cert, _object needle_cert, _object haystack_avoid, _int foo)

Description
This method creates an image that describes the match in every pixel in the image and the needle-Image.
new pixel value =
sum( my_abs(needle_pixel-haystack_pixel))

The new image only have the red rgb-part set.

**Arguments**

**Returns**
the new image object

**Note**
*experimental status*; may not be exact the same output in later versions

See Also
phasev and phaseh

**Image.Image.mirrorx**

object mirrorx()

**Description**
mirrors an image:

![Image](image_url)

original - mirrorx();

**Returns**
the new image object

**Image.Image.mirrory**

object mirrory()

**Description**
mirrors an image:
Image.Image.modify_by_intensity

object modify_by_intensity(int r, int g, int b, int|array(int) v1, ...
  ..., int|array(int) vn)

Description
Recolor an image from intensity values.

For each color an intensity is calculated, from r, g and b factors (see grey), this gives a value between 0 and max.

The color is then calculated from the values given, v1 representing the intensity value of 0, vn representing max, and colors between representing intensity values between, linear.

Arguments

Returns
the new image object

See Also
grey, ‘*’ and color

Image.Image.noise

void noise(array(float|array(int)) colorrange)
void noise(array(float|array(int)) colorrange, float scale,
  float xdiff, float ydiff, float cscale)
**Description**
Gives a new image with the old image’s size, filled with a ‘noise’ pattern.

The random seed may be different with each instance of pike.

Example:
```c
->noise( (0,({255,0,0}), 0.3,({0,255,0}), 0.6,({0,0,255}),
0.8,({255,255,0})), 0.2,0.0,0.0,1.0 );
```

**Arguments**

**See Also**
turbulence

**Image.orient**, **Image.orient4**

object orient(void|array(object))
array(object) orient4()

**Description**
Draws images describing the orientation of the current image.

**orient** gives an HSV image (run a hsv_to_rgb pass on it to get a viewable image). corresponding to the angle of the orientation:

```
| /   / -   \
hue= 0 64 128 192 (=red in an hsv image)
purple cyan green red
```

Red, green and blue channels are added and not compared separately.

If you first use orient4 you can give its output as input to this function.

The **orient4** function gives back 4 image objects, corresponding to the amount of different directions, see above.

**Returns**
an image or an array of the four new image objects

**Note**
experimental status; may not be exact the same output in later versions
**Image.Image.outline_mask**

object outline()
object outline(int olr, int olg, int olb)
object outline(int olr, int olg, int olb, int bkgr, int bkgg, int bkgb)
object outline(array(array(int)) mask)
object outline(array(array(int)) mask, int olr, int olg, int olb)
object outline(array(array(int)) mask, int olr, int olg, int olb, int bkgr, int bkgg, int bkgb)
object outline()
object outline_mask(int bkgr, int bkgg, int bkgb)
object outline_mask(array(array(int)) mask)
object outline_mask(array(array(int)) mask, int bkgr, int bkgg, int bkgb)

**Description**

Makes an outline of this image, ie paints with the given color around the non-background pixels.

Default is to paint above, below, to the left and the right of these pixels.

You can also run your own outline mask.

The outline_mask function gives the calculated outline as an alpha channel image of white and black instead.

![Image example](image.jpg)

original  masked  ...and  outlined  with red

**Arguments**

**Returns**

the new image object

**Note**

no antialias!

**Image.Image.paste**

object paste(object image)
object paste(object image, int x, int y)
Description
Pastes a given image over the current image.

Arguments

Returns
the object called

See Also
paste_mask, paste_alpha and paste_alpha_color

Image.Image.paste_alpha

object paste_alpha(object image, int alpha)
object paste_alpha(object image, int alpha, int x, int y)

Description
Pastes a given image over the current image, with the specified alpha channel value.
An alpha channel value of 0 leaves nothing of the original image in the paste area, 255 is meaningless and makes the given image invisible.

Arguments

Returns
the object called

See Also
paste_mask, paste and paste_alpha_color

Image.Image.paste_alpha_color

object paste_alpha_color(object mask)
object paste_alpha_color(object mask, int x, int y)
object paste_alpha_color(object mask, int r, int g, int b)
object paste_alpha_color(object mask, int r, int g, int b, int x, int y)
object paste_alpha_color(object mask, Color color)
object paste_alpha_color(object mask, Color color, int x, int y)

Description
Pastes a given color over the current image, using the given mask as opaque channel.
A pixel value of 255 makes the result become the color given, 0 doesn’t change anything.
The masks red, green and blue values are used separately. If no color are given, the current is used.

Arguments
Returns
the object called

See Also
`paste_mask`, `paste_alpha` and `paste_alpha_color`

`Image.Image.paste_mask`

object paste_mask(object image, object mask)
object paste_mask(object image, object mask, int x, int y)

Description
Pastes a given image over the current image, using the given mask as opaque channel.

A pixel value of 255 makes the result become a pixel from the given image, 0 doesn’t change anything.

The masks red, green and blue values are used separately.

Arguments

Returns
the object called

See Also
`paste`, `paste_alpha` and `paste_alpha_color`

`Image.Image.phasev`,
`Image.Image.phaseh`,
`Image.Image.phasehv`,
`Image.Image.phaserv`

object phaseh()
object phasev()
object phasevh()
object phasehv()

Description
Draws images describing the phase of the current image. phaseh gives the horizontal phase and phasev the vertical phase.

phaseh gives an image where

\[
\begin{align*}
\text{max falling} & \quad \text{min rising} \\
\text{value} = & \quad 0 & 64 & 128 & 192
\end{align*}
\]

0 is set if there is no way to determine if it is rising or falling. This is done for the every red, green and blue part of the image.

Phase images can be used to create ugly effects or to find meta-information in the original image.
Returns
the new image object

Note
**experimental status:** may not be exact the same output in later versions

Bugs
0 should not be set as explained above.

**Image.Image.polyfill**

```plaintext
object polyfill(array(int|float) \_\_\_ \_\_ curve)
```

Description
fills an area with the current color

Arguments

Returns
the current object

Note
Lines in the polygon may *not* be crossed without the crossing coordinate specified in both lines.

Bugs
Inverted lines reported on Intel and Alpha processors.

See Also
setcolor

**Image.Image.random**, **Image.Image.randomgrey**

```plaintext
object random()
object random(int \_\_\_ seed)
object randomgrey()
object randomgrey(int \_\_\_ seed)
```

Description
Gives a randomized image;
Use with \(\_\_\_\_\_\_\_\_\_\_\_\) or \(\_\_\_\_\_\_\_\_\_\_\) for one-color-results.

**Returns**
a new image

**See Also**
test and noise

\begin{verbatim}
\end{verbatim}

\begin{verbatim}
object write_lsb_rgb(string what)
object write_lsb_grey(string what)
string read_lsb_rgb()
string read_lsb_grey()
\end{verbatim}

**Description**
These functions read/write in the least significant bit of the image pixel values. The \(\_\_\_\_\_\_\_\_\_\_\) functions read/write on each of the red, green and blue values, and the grey keeps the same lsb on all three.

The string is nullpadded or cut to fit.

**Arguments**

**Returns**
the current object or the read string

\begin{verbatim}
Image.Image.rotate, Image.Image.rotate_expand
\end{verbatim}

\begin{verbatim}
object rotate(int|float angle)
object rotate(int|float angle, int r, int g, int b)
object rotateexpand(int|float angle)
object rotateexpand(int|float angle, int r, int g, int b)
\end{verbatim}

**Description**
Rotates an image a certain amount of degrees (360° is a complete rotation)
counter-clockwise:

The "expand" variant of functions stretches the image border pixels rather than filling with the given or current color.

This rotate uses the skewx() and skewy() functions.

**Arguments**

**Returns**
the new image object

**Image.Image.rotate_ccw**
object rotate_ccw()

**Description**
rotates an image counter-clockwise, 90 degrees.

Returns
the new image object

**Image.Image.rotate_cw**
object rotate_cw()
Description
rotates an image clockwise, 90 degrees.

Returns
the new image object

Image.Image.rotate
object rotate()

Description
rotates an image clockwise, 90 degrees.

Returns
the new image object

Image.Image.scale
object scale(float factor)
object scale(0.5)
object scale(float xfactor, float yfactor)

Description
scales the image with a factor, 0.5 is an optimized case.

Arguments

Returns
the new image object

Image.Image.scale
object scale(int newxsize, int newysize)
object scale(0, int newysize)
object scale(int newxsize, 0)

Description
scales the image to a specified new size, if one of newxsize or newysize is 0, the image aspect ratio is preserved.

Arguments

Returns
the new image object

Note
resulting image will be 1x1 pixels, at least

Image.Image.select_from
object select_from(int x, int y)
object select_from(int x, int y, int edge_value)
12.1.  **IMAGE.IMAGE**

*Description*
Makes a grey-scale image, for alpha-channel use.

This is very close to a floodfill.

The image is scanned from the given pixel, filled with 255 if the color is the same, or 255 minus distance in the colorcube, squared, rightshifted 8 steps (see `distancesq`).

When the edge distance is reached, the scan is stopped. Default edge value is 30. This value is squared and compared with the square of the distance above.

*Arguments*

*Returns*
the new image object

*See Also*
`distancesq`

**Image.Image.setcolor**

object setcolor(int r, int g, int b)
object setcolor(int r, int g, int b, int alpha)

*Description*
set the current color

*Arguments*

*Returns*
the object called

**Image.Image.setpixel**

object setpixel(int x, int y)
object setpixel(int x, int y, int r, int g, int b)
object setpixel(int x, int y, int r, int g, int b, int alpha)

*Description*

```
original -¿setpixel-(10,10,-255,0,0)
```
Arguments

Returns the object called

Image.Image.skewx,
Image.Image.skewx_expand

object skewx(int x)
object skewx(int yfactor)
object skewx(int x, int r, int g, int b)
object skewx(int yfactor, int r, int g, int b)
object skewx_expand(int x)
object skewx_expand(int yfactor)
object skewx_expand(int x, int r, int g, int b)
object skewx_expand(int yfactor, int r, int g, int b)

Description
Skews an image an amount of pixels or a factor; a skew-x is a transformation:

Returns
the new image object

Image.Image.skewy,
Image.Image.skewy_expand

object skewy(int y)
object skewy(int xfactor)
object skewy(int y, int r, int g, int b)
object skewy(int xfactor, int r, int g, int b)
object skewy_expand(int y)
object skewy_expand(int xfactor)
object skewy_expand(int y, int r, int g, int b)
object skewy_expand(int xfactor, int r, int g, int b)

Description
Skews an image an amount of pixels or a factor; a skew-y is a transformation:
The "expand" variant of functions stretches the image border pixels rather than filling with the given or current color.

**Arguments**

Returns
the new image object

**Image.Image.test**

object test()
object test(int seed)

**Description**
Generates a test image, currently random gradients.

Returns
the new image

**Note**
May be subject to change or cease without prior warning.

**See Also**
gradients and tuned_box
Image.Image.threshold

object threshold()
object threshold(int level)
object threshold(int r, int g, int b)
object threshold(Color color)

Description
Makes a black-white image.

If any of red, green, blue parts of a pixel is larger than the given value, the pixel will become white, else black.

This method works fine with the grey method.

If no arguments are given, it will paint all non-black pixels white. (Ie, default is 0,0,0.)

original - threshold(100); 
threshold(0,100,0);

Returns
the new image object

Note
The above statement "any ..." was changed from "all ..." in Pike 0.7 (9906). It also uses 0,0,0 as default input, instead of current color. This is more useful.

See Also
grey

Image.Image.tuned_box

object tuned_box(int x1, int y1, int x2, int y2,
array(array(int)) corner_color)

Description
Draws a filled rectangle with colors (and alpha values) tuned between the corners.

Tuning function is \((1.0-x/xw)*(1.0-y/yw)\) where \(x\) and \(y\) is the distance to the corner and \(xw\) and \(yw\) are the sides of the rectangle.
12.1. IMAGE.IMAGE

original tuned box solid tuning (blue,red,green,yellow) tuning transparency (as left + 255,128,128,0)

**Arguments**

**Returns**
the object called

**Image.Image.turbulence**

```plaintext
void turbulence(array(float|int|array(int)),colorrange)
void turbulence(array(float|int|array(int)),colorrange, int octaves, float scale, float xdiff, float ydiff, float cscale)
```

**Description**
gives a new image with the old image’s size, filled with a 'turbulence' pattern.

The random seed may be different with each instance of pike.

Example:
```plaintext
->turbulence( {0,({229,204,204}), 0.9,({229,20,20}), 0.9,Color.black})
```

**Arguments**

**See Also**
noise and Image.Color

**Image.Image.xsize**

```plaintext
int xsize()
```

**Returns**
the width of the image
**Image.Image.ysize**

int ysize()

*Returns*
the height of the image

**Image.Image.‘/‘, Image.Image.‘%‘**

object ‘/(object_operand)
object ‘/(Color_color)
object ‘/(int_value)
object ‘%(object_operand)
object ‘%(Color_color)
object ‘%(int_value)

*Description*
Divides pixel values and creates a new image from the result or the rest.

*Arguments*

*Returns*
the new image object

*Note*
It isn’t possible to do a modulo 256 either. (why?)

*See Also*

**Image.Image.‘&‘**

object ‘&(object_operand)
object ‘&(array(int)_color)
object ‘&(int_value)

*Description*
makes a new image out of the minimum pixels values

*Arguments*

*Returns*
the new image object

*See Also*
**Image.Image.==**

- `int ==(object, operand)`
- `int ==(array(int), color)`
- `int ==(int, value)`

**Description**
Comparision an image with another image or a color.

Comparison is strict and on pixel-by-pixel basis. (Means if not all pixel r,g,b values are correct compared with the corresponding pixel values, 0 is returned.)

**Arguments**

**Returns**
true (1) or false (0).

**Note**
`
=>` or `=` on empty ("no image") image objects or images with different size will result in an error. `==` is always true on two empty image objects and always false if one and only one of the image objects is empty or the images differs in size.

\(a=b\) and \(a=\) between objects is equal to !(a\(\neq\)b) and !(a\(\neq\)b), which may not be what you want (since both \(=\) and \(\neq\) can return false, comparing the same images).

**See Also**
`
=`, `+=`, `|`, `*` and `&`

**Image.Image.***

- `object *(object, operand)`
- `object *(array(int), color)`
- `object *(int, value)`

**Description**
Multiplies pixel values and creates a new image.

This can be useful to lower the values of an image, making it greyer, for instance:

```
image=image*128+64;
```
Returns
the new image object

See Also
‘+', '|', '& and Image.Layer

Image.Image.'+

object '+(object operand)
object '+(array(int) color)
object '+(int value)

Description
adds two images; values are truncated at 255.

Arguments

Returns
the new image object

See Also
‘-', '|, '& and Image.Layer

Image.Image.'-

object '-(object operand)
object '-(array(int) color)
object '-(int value)

Description
makes a new image out of the difference

Arguments

Returns
the new image object

See Also
‘+', '|, '&, ‘* and Image.Layer, min, max and ‘==

Image.Image.'==

object '|(object operand)
object '|(array(int) color)
object '|(int value)

Description
makes a new image out of the maximum pixels values

Arguments

Returns
the new image object
12.2. **Image.Colortable**

**Description**
This object keeps colortable information, mostly for image re-coloring (quantization).

The object has color reduction, quantisation, mapping and dithering capabilities.

*See Also*
Image, Image.Image, Image.Font and Image.GIF

**Image.Colortable.create**, **Image.Colortable.add**

```plaintext
void create()
void create(array(array(int)), colors)
void create(object(Image.Image), image, int, number)
void create(object(Image.Image), image, int, number, array(array(int)), needed)
void create(int, r, int, g, int, b)
void create(int, r, int, g, int, b, array(int), from1, array(int), to1, int, steps1, ..., array(int), fromn, array(int), ton, int, stepsn)
object add(array(array(int)), colors)
object add(object(Image.Image), image, int, number)
object add(object(Image.Image), image, int, number, array(array(int)), needed)
object add(int, r, int, g, int, b)
object add(int, r, int, g, int, b, array(int), from1, array(int), to1, int, steps1, ..., array(int), fromn, array(int), ton, int, stepsn)
```

**Description**
create initiates a colortable object. Default is that no colors are in the colortable.

add takes the same argument(s) as create, thus adding colors to the colortable.

The colortable is mostly a list of colors, or more advanced, colors and weight.

The colortable could also be a colorcube, with or without additional scales. A colorcube is the by-far fastest way to find colors.

Example:

```plaintext
color=colortable(my_image, 256); // the best 256 colors
color=colortable(my_image, 256, (0, 0, 0)); // black and the best other 255
```
Arguments

Note
max hash size is (probably, set by a #define) 32768 entries, giving maybe half that number of colors as maximum.

**Image.Colortable.cast**

object cast(string\_to)

Description
cast the colortable to an array or mapping, the array consists of Image.Color objects and are not in index order. The mapping consists of index:Image.Color pairs, where index is the index (int) of that color.

example: \((\text{mapping})\text{Image.Colortable(img)}\)

Arguments

**Image.Colortable.corners**

array(object) corners()

Description
Gives the eight corners in rgb colorspace as an array. The "black" and "white" corners are the first two.

**Image.Colortable.cubicles**

object cubicles()
object cubicles(int\_r, int\_g, int\_b)
object cubicles(int\_r, int\_g, int\_b, int\_accuracy)

Description
Set the colortable to use the cubicles algorithm to lookup the closest color. This is a mostly very fast and very accurate way to find the correct color, and the default algorithm.

The colorspace is divided in small cubes, each cube containing the colors in that cube. Each cube then gets a list of the colors in the cube, and the closest from the corners and midpoints between corners.

When a color is needed, the algorithm first finds the correct cube and then compares with all the colors in the list for that cube.

example: \(\text{colors=Image.Colortable(img)->cubicles()}\);

algorithm time: between \(O[m]\) and \(O[m * n]\), where \(n\) is numbers of colors and \(m\) is number of pixels
The arguments can be heavy trimmed for the usage of your colortable; a large number (10×10×10 or bigger) of cubicles is recommended when you use the colortable repeatedly, since the calculation takes much more time than usage. Recommended values:

<table>
<thead>
<tr>
<th>Image size setup</th>
<th>Recommended values:</th>
</tr>
</thead>
<tbody>
<tr>
<td>100×100 cubicles (default)</td>
<td>1000×1000 cubicles (factor 2 faster than default)</td>
</tr>
</tbody>
</table>

In some cases, the full method is faster.

![Images showing original, default cubicles, and accuracy=200 images]

**Arguments**

**Returns**
the called object

**Note**
this method doesn’t figure out the cubicles, this is done on the first use of the colortable.

Not applicable to colorcube types of colortable.

**Image.Colortable.floyd_steinberg**

object floyd_steinberg()
object floyd_steinberg(int bidir, int|float forward, int|float downforward, int|float down, int|float downback, int|float factor)

**Description**
Set dithering method to floyd_steinberg.

The arguments to this method is for fine-tuning of the algorithm (for computer graphics wizards).
CHAPTER 12. IMAGE

original  

floyd_steinberg to a 4×4×4 color-cube

floyd_steinberg to 16 chosen colors

Arguments

Returns
the called object

Image.Colortable.full

object full()

Description
Set the colortable to use full scan to lookup the closest color.

element: colors=Image.Colortable(img)->full();

algorithm time: O[n*m], where n is numbers of colors and m is number of pixels

Returns
the called object

Note
Not applicable to colorbale types of colortable.

See Also
cubicles and map

Image.Colortable.image

object image()

Description
cast the colortable to an image object

each pixel in the image object is an entry in the colortable

Returns
the resulting image object

Image.Colortable.‘*‘
Image.Colortable.”*"
**Image.Colortable.map**

object map(object image)
object *(object image)
object ***(object image)

object map(string data, int xsize, int ysize)
object *(string data, int xsize, int ysize)
object ***(string data, int xsize, int ysize)

**Description**

Map colors in an image object to the colors in the colortable, and creates a new image with the closest colors.
CHAPTER 12. IMAGE

randomcube
dither
original 2 4 8 16 32 colors

Returns
a new image object

Note
Flat (not cube) colortable and not 'full' method: this method does figure out
the data needed for the lookup method, which may take time the first use of
the colortable - the second use is quicker.

See Also
cubicles and full

Image.Colortable.nodither

object nodither()

Description
Set no dithering (default).

Returns
the called object

Image.Colortable.ordered

object ordered()
object ordered(int r, int g, int b)
object ordered(int r, int g, int b, int xsize, int ysize)
object ordered(int r, int g, int b, int xsize, int ysize, int x, int y)
object ordered(int r, int g, int b, int xsize, int ysize, int rx, int ry, int gx, int gy, int bx, int by)

Description
Set ordered dithering, which gives a position-dependent error added to the pixel
values.
12.2. **Image.Colortable**

**Arguments**

**Returns**

the called object

**See Also**

randomcube, nodither, floyd_steinberg and create

_**Image.Colortable.randomcube**_

_**Image.Colortable.randomgrey**_

object randomcube()
object randomcube(int r, int g, int b)
object randomgrey()
object randomgrey(int err)

**Description**

Set random cube dithering. Color chosen is the closest one to color in picture plus (flat) random error; color±random(error).

The randomgrey method uses the same random error on red, green and blue and the randomcube method has three random errors.
CHAPTER 12. IMAGE

original mapped to \texttt{Image.Colortable(4,4,4)}->
\texttt{randomcube()}

randomgrey() randomgrey()

\textbf{Arguments}

\textbf{Returns}
the called object

\textbf{Note}
randomgrey method needs colorcube size to be the same on red, green and blue sides to work properly. It uses the red colorcube value as default.

\textbf{See Also}
ordered, nodither, floyd_steinberg and create

\texttt{Image.Colortable.reduce, Image.Colortable.reduce_fs}

\begin{verbatim}
object reduce(int colors)
object reduce_fs(int colors)
\end{verbatim}

\textbf{Description}
reduces the number of colors

All needed (see create) colors are kept.

reduce_fs creates and keeps the outmost corners of the color space, to improve floyd-steinberg dithering result. (It doesn’t work very well, though.)

\textbf{Arguments}
Returns
the new Colortable object

*Note*
this algorithm assumes all colors are different to begin with (!)

reduce_fs keeps the "corners" as "needed colors".

*See Also*
corners

**Image.Colortable.rigid**

object rigid()
object rigid(int r, int g, int b)

*Description*
Set the colortable to use the "rigid" method of looking up colors.

This is a very simple way of finding the correct color. The algorithm initializes
a cube with $r \times g \times b$ colors, where every color is chosen by closest match to
the corresponding coordinate.

This format is not recommended for exact match, but may be usable when it
comes to quickly view pictures on-screen.

It has a high init-cost and low use-cost. The structure is initiated at first usage.

*Returns*
the called object

*Note*
Not applicable to colorcube types of colortable.

*See Also*
cubicles, map and full

**Image.Colortable.spacefactors**

object spacefactors(int r, int g, int b)

*Description*
Colortable tuning option, this sets the color space distance factors. This is used
when comparing distances in the colorspace and comparing grey levels.

Default factors are 3, 4 and 1; blue is much darker than green. Compare with
Image.Image-¿grey().

*Returns*
the called object

*Note*
This has no sanity check. Some functions may bug if the factors are to high
-color reduction functions sums grey levels in the image, this could exceed maxint
in the case of high factors. Negative values may also cause strange effects. *grin*
**Image.Colortable.**

---

### Image.Colortable.+

`object `+(object, with, ...)

**Description**

Sums colortables

**Arguments**

**Returns**

The resulting new Colortable object

---

### Image.Colortable.-

`object `-(object, with, ...)

**Description**

Subtracts colortables

**Arguments**

**Returns**

The resulting new Colortable object

---

### 12.3 Image.Layer

**Image.Layer**

**Description**

**See Also**

layers

**Image.Layer.alpha**,  
**Image.Layer.image**,  
**Image.Layer.set_image**

`object set_image(object(Image.Image) image)`  
`object set_image(object(Image.Image) image, object(Image.Image), alpha_channel)`  
`object|int(0) image()`  
`object|int(0) alpha()`

**Description**

Set/change/get image and alpha channel for the layer. You could also cancel the channels giving 0 instead of an image object.

**Note**

Image and alpha channel must be of the same size, or canceled.
12.3. IMAGE.LAYER

**Image.Layer.alpha_value**,  
**Image.Layer.set_alpha_value**

object set_alpha_value(float value)  
double alpha_value()

*Description*
Set/get the general alpha value of this layer. This is a float value between 0 and 1, and is multiplied with the alpha channel.

**Image.Layer.autocrop**,  
**Image.Layer.find_autocrop**

object autocrop()  
object autocrop(int left, int right, int top, int bottom)  
array(int) find_autocrop()  
array(int) find_autocrop(int left, int right, int top, int bottom)

*Description*
This crops (or finds) a suitable crop, non-destructive crop. The layer alpha channel is checked, and edges that are transparent are removed.

(What really happens is that the image and alpha channel is checked, and edges equal the fill setup is cropped away.)

find_autocrop() returns an array of xoff, yoff, xsize, ysize, which can be fed to crop().

*Note*
A tiled image will not be cropped at all.

left...bottom arguments can be used to tell what sides cropping are ok on.

*See Also*
crop and Image.Image->autocrop

**Image.Layer.mode**,  
**Image.Layer.set_mode**,  
**Image.Layer.available_modes**

object set_mode(string mode)  
string mode()  
array(string) available_modes()

*Description*
Set/get layer mode. Mode is one of these:  
"normal", "add", "subtract", "multiply", "divide", "modulo", "invs subtract",  
"invdivide", "invm odulo", "difference", "max", "min", "bitwise_and", "bitwise_or", "bitwise_xor",  
"replace", "red", "green", "blue",  
"replace_hsv", "hue", "saturation", "value", "color"
"darken", "lighten",
"dissolve", "behind", "erase",
available_modes() simply gives an array containing the names of these modes.

Note
image and alpha channel must be of the same size, or canceled.

**Image.Layer.clone**

object clone()

*Description*
Creates a copy of the called object.

*Returns*
the copy

**Image.Layer.create**

void create(object image, object alpha, string mode)
void create(mapping info)
void create()
void create(int xsize, int ysize, object color)
void create(object color)

*Description*
The Layer construct either three arguments, the image object, alpha channel and mode, or a mapping with optional elements:

"image":image,
// default: black

The layer can also be created "empty", either giving a size and color - this will give a filled opaque square, or a color, which will set the "fill" values and fill the whole layer with an opaque color.

All values can be modified after object creation.

*Note*
image and alpha channel must be of the same size.

**Image.Layer.crop**

object crop(int xoff, int yoff, int xsize, int ysize)

*Description*
Crops this layer at this offset and size. Offset is not relative the layer offset, so this can be used to crop a number of layers simultaneously.

The fill values are used if the layer is enlarged.
Returns
a new layer object

Note
The new layer object may have the same image object, if there was no cropping
to be done.

Image.Layer.fill,
Image.Layer.fill_alpha,
Image.Layer.set_fill

object set_fill(Color, color)
object set_fill(Color, color, Color, alpha)
object fill()
object fill_alpha()

Description
Set/query fill color and alpha, ie the color used "outside" the image. This is
mostly useful if you want to "frame" a layer.

Image.Layer.set_misc_value,
Image.Layer.get_misc_value

mixed set_misc_value(what, to)
mixed get_misc_value(what)

Description
Set or query misc. attributes for the layer.

As an example, the XCF and PSD image decoders set the 'name' attribute to
the name the layer had in the source file.

Image.Layer.xoffset,
Image.Layer.yoffset,
Image.Layer.set_offset

object set_offset(x, y)
int xoffset()
int yoffset()

Description
Set/query layer offset.

Image.Layer.tiled,
Image.Layer.set_tiled

object set_tiled(yes)
int tiled()

Description
Set/query tiled flag. If set, the image and alpha channel will be tiled rather
then framed by the fill values.
CHAPTER 12. IMAGE

Image.Layer.ysize, Image.Layer.xsize

int xsize()
int ysize()

Description
Query layer offset. This is the same as layer image/alpha image size.

12.4 Image.Font

Image.Font

Description

Note
Short technical documentation on a font file: This object adds the text-drawing and -creation capabilities of the Image module.

For simple usage, see write and load.

other methods: baseline, height, set_xspacing_scale, set_yspacing_scale, text_extents

struct file_head
{
  unsigned INT32 cookie;  - 0x464f4e54
  unsigned INT32 version; - 1
  unsigned INT32 chars;  - number of chars
  unsigned INT32 height; - height of font
  unsigned INT32 baseline; - font baseline
  unsigned INT32 o[1];    - position of char_head's
} *fh;

struct char_head
{
  unsigned INT32 width;   - width of this character
  unsigned INT32 spacing; - spacing to next character
  unsigned char data[1];  - pixmap data (1byte/pixel)
} *ch;

See Also
Image and Image.Image

Image.Font.baseline

int baseline()
Returns
font baseline (pixels from top)

See Also
height and text_extents

Image.Font.create

void create(string filename)

Description
Loads a font file to this font object. Similar to load().

Image.Font.height, Image.Font.text_extents

int height()
array(int) text_extents(string text, ...)

Description
Calculate extents of a text-image, that would be created by calling writewith the same arguments.

Arguments

Returns
an array of width and height

See Also
write, height and baseline

Image.Font.load

object|int load(string filename)

Description
Loads a font file to this font object.

Arguments

Returns
zero upon failure, font object upon success

See Also
write

Image.Font.set_xspacing_scale, Image.Font.set_yspacing_scale

void set_xspacing_scale(float scale)
void set_yspacing_scale(float scale)
**Description**
Set spacing scale to write characters closer or more far away. This does not change scale of character, only the space between them.

**Arguments**

**Image.Font.write**

object write(string text, ...)

**Description**
Writes some text; thus creating an image object that can be used as mask or as a complete picture.

**Arguments**

**Returns**
an Image.Image object

**See Also**
text_extents, load, Image.Image->paste_mask and Image.Image->paste_alpha_color

### 12.5 Image.colortable

**Image.colortable**

### 12.6 Image.Poly

**Image.Poly**

**Description**

### 12.7 Image.Color

**Description**
This module keeps names and easy handling for easy color support. It gives you an easy way to get colors from names.

A color is here an object, containing color information and methods for conversion, see below.

Image.Color can be called to make a color object. Image.Color() takes the following arguments:
Image.Color(string name)  // "red"
Image.Color(string prefix_string)  // "lightblue"
Image.Color(string hex_name)  // "#ff00ff"
Image.Color(string cmyk_string)  // "%17,42,0,19.4"
Image.Color(string hsv_string)  // "%327,90,32"
Image.Color(int red, int green, int blue)

The color names available can be listed by using indices on Image.Color. The
colors are available by name directly as Image.Color.name, too:

...Image.Color.red...
...Image.Color.green...
or, maybe
import Image.Color;
...red...
...green...
...lightgreen...

Giving red, green and blue values is equal to calling Image.Color.rgb().

The prefix_string method is a form for getting modified colors, it understands all modifiers (light, dark, bright, dull and neon). Simply use "method"+"color";
(as in lightgreen, dullmagenta, lightdullorange).

The hex_name form is a simple #rrggbb form, as in HTML or X-program argument. A shorter form (#rgb) is also accepted. This is the inverse to the Image.Color.Color-to-hex() method.

The cmyk_string is a string form of giving cmyk (cyan, magenta, yellow, black) color. These values are floats representing percent.

The hsv_string is another hue, saturation, value representation, but in floats; hue is in degree range (0..360), and saturation and value is given in percent. This is not the same as returned or given to the hsv() methods!

Note
Image.Color["something"] will never(!) generate an error, but a zero_type 0, if the color is unknown. This is enough to give the error "not present in module", if used as Image.Color.something, though.

If you are using colors from for instance a webpage, you might want to create the color from Image.Color.guess(), since that method is more tolerant for mistakes and errors.

Image.Color() is case- and space-sensitive. Use Image.Color.guess() to catch all variants.

and subtract with a space (lower_case(x)-" ") to make sure you get all variants.
See Also

*Image.Color.greylevel*

*Image.Color.hsv*

*Image.Color.rgb*

*Image.Color.html*

*Image.Color.cmyk*

object rgb(int red, int green, int blue)
object hsv(int hue, int saturation, int value)
object cmyk(float c, float m, float y, float k)
object greylevel(int level)
object html(string html_color)

**Description**

Creates a new color object from given red, green and blue, hue, saturation and
value, or greylevel, in color value range. It could also be created from cmyk
values in percent.

The html() method only understands the HTML color names, or the #rrggbb
form. It is case insensitive.

**Returns**

the created object.

*Image.Color.guess*

object guess(string)

**Description**

This is equivalent to `Image.Color(lower_case(str)" ")`, and tries the color
with a prepending ‘#’ if no corresponding color is found.

**Returns**

a color object or zero

*Image.Color._indices*

*Image.Color._values*

array(string) _indices()
array(object) _values()

**Description**

(is as `indices(Image.Color)` or `values(Image.Color)`)) indices gives a list
of all the known color names, values gives there corresponding objects.

See Also
Image.Color

12.7.1  *Image.Color.Color*

*Image.Color.Color*
Description
This is the color object. It has six readable variables, \( r, g, b \), for the red, green and blue values, and \( h, s, v \), for the hue, saturation and value values.


object light()
object dark()
object neon()
object bright()
object dull()

Description
Color modification methods. These return a new color.

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{method} & \text{effect} & h & s & v \\ \hline
\text{light} & \text{raise light level} & \pm 0 & \pm 0 & +50 \\ \hline
\end{array}
\]

light and dark lower/higher saturation when value is min-/maximised respective.

Returns
the new color object

Note
The opposites may not always take each other out. The color is maximised at white and black levels, so, for instance \( \text{Image.Color.white-light()-dark()} \) doesn’t give the white color back, but the equal to \( \text{Image.Color.white-dark()} \), since white can’t get any lighter.

\( \text{Image.Color.Color.cast} \)

array|string cast()

Description
cast the object to an array, representing red, green and blue (equal to \( \text{->rgb()} \)), or to a string, giving the name (equal to \( \text{->name()} \)).

Returns
the name as string or rgb as array

See Also
rgb and name

**Image.Color.Color.rgb**,  
**Image.Color.Color.cmyk**

array(int) rgb()  
array(int) hsv()  
array(int) cmyk()  
int greylevel()  
int greylevel(int r, int g, int b)

**Description**  
This is methods of getting information from an Image.Color.Color object.

They give an array of red, green and blue (rgb) values (color value),  
hue, saturation and value (hsv) values (range as color value),  
cyan, magenta, yellow, black (cmyk) values (in percent)  
or the greylevel value (range as color value).

The greylevel is calculated by weighting red, green and blue. Default weights are 87, 127 and 41, respective, and could be given by argument.

**Returns**  
array(int) respective int

**See Also**  
Image.Color.Color and grey

**Image.Color.Color.create**

void create(int r, int g, int b)

**Description**  
This is the main Image.Color.Color creation method, mostly for internal use.

**Image.Color.Color.grey**

object grey()  
object grey(int red, int green, int blue)

**Description**  
Gives a new color, containing a grey color, which is calculated by the greylevel method.

**Returns**  
a new Image.Color.Color object

**See Also**  
greylevel

**Image.Color.Color.name**,  
**Image.Color.Color.hex**,  
**Image.Color.Color.html**

string hex()  
string hex(int n)  
string name()  
string html()
12.7. IMAGE.COLOR

**Description**

Information methods.

`hex()` simply gives a string on the #rrggbbs format. If `n` is given, the number of significant digits is set to this number. (Ie, `n=3` gives #rrrgggggb.)

`name()` is a simplified method; if the color exists in the database, the name is returned, per default is the `hex()` method use.

`html()` gives the HTML name of the color, or the `hex(2)` if it isn’t one of the 16 HTML colors.

**Returns**
a new Image.Color.Color object

**See Also**

rgb, hsv and Image.Color

---

**Image.Color.Color.**

```plaintext
_sprintf(string s, _mapping flags)
```

**Description**

---

**Image.Color.Color.**

```plaintext
==
```

- `(object, other_color)`
- `(array(int), rgb)`
- `(int, greylevel)`
- `(string, name)`

**Description**

Compares this object to another color, or color name. Example:

```plaintext
object red=Image.Color.red;
object other=Image.Color. ...;
object black=Image.Color.black;
```

**Returns**

1 or 0

**Note**
The other datatype (not color object) must be to the right!

**See Also**

rgb, grey and name
12.8 Image.X

Description
This submodule handles encoding and decoding of the binary formats of X11.

See Also
Image, Image.Image and Image.Colortable

Image.X.decode_pseudocolor

object decode_pseudocolor(string data, int width, int height, int bpp, int alignbits, int swapbytes, object colortable)

Description
lazy support for pseudocolor ZPixmaps

Note
currently, only byte-aligned pixmaps are supported

Image.X.decode_truecolor_masks, Image.X.decode_truecolor

object decode_truecolor_masks(string data, int width, int height, int bpp, int alignbits, int swapbytes, int rmask, int gmask, int bmask)

Description
lazy support for truecolor ZPixmaps

Note
currently, only byte-aligned masks are supported

Image.X.encode_pseudocolor

string encode_pseudocolor(object image, int bpp, int alignbits, int vbpp, object colortable)
string encode_pseudocolor(object image, int bpp, int alignbits, int vbpp, object colortable, string translate)

Description

Arguments

Note
currently, only upto 16 bits pseudocolor are supported.

Image.X.encode_truecolor,
12.9. IMAGE.ANY

**Image.X.encode_truecolor_masks**

string encode_truecolor(object image, int bpp, int alignbits, int swapbytes, int rbits, int rshift, int gbits, int gshift, int bbits, int bshift)
string encode_truecolor_masks(object image, int bpp, int alignbits, int swapbytes, int rmask, int gmask, int bmask)
string encode_truecolor_masks(object image, int bpp, int alignbits, int swapbytes, int rmask, int gmask, int bmask, object ct)

**Description**

Pack an image into a truecolor string. You will get a string of packed red, green and blue bits; ie:

`encode_truecolor(img, 12, 32, 0, 3, 5, 4, 0, 3, 8)` will give (aligned to even 32 bits for each row):

```
0bbbrrr0 gggg0bbb rrr0gggg 0bbb...
```

4,0: 4 bits green shifted 0 bits  
3,5: 3 bits red shifted 5 bits  
3,8: 3 bits blue shifted 8 bits

The above call is equal to

`encode_truecolor_masks(img, 12, 32, 0, 224, 15, 768)`

The latter gives possibility to use dither algorithms, but is slightly slower.

**Arguments**

12.9. Image.ANY

**Description**

This method calls the other decoding methods and has some heuristics for what type of image this is.

Methods: decode, decode_alpha, _decode

**See Also**

Image

**Image.ANY._decode**,  
**Image.ANY.decode**,  
**Image.ANY.decode_alpha**

mapping _decode(string data)  
object decode(string data)  
object decode_alpha(string data)
Description
Tries heuristics to find the correct method of decoding the data, then calls that method.

The result of `decode()` is a mapping that contains

"type": image data type (ie, "image/jpeg" or similar)
"image": the image object,
"alpha": the alpha channel or 0 if N/A

Note
Throws upon failure.

12.10 Image.AVS

Description

Image.AVS.decode, Image.AVS.encode, Image.AVS.decode

object decode(string data)
mapping decode(string data)
string encode(object image)

Description
Handle encoding and decoding of AVS-X images. AVS is rather trivial, and not really useful, but:

An AVS file is a raw (uncompressed) 24 bit image file with alpha. The alpha channel is 8 bit, and there is no separate alpha for r, g and b.

12.11 Image.BMP

Description
This submodule keeps the BMP (Windows Bitmap) encode/decode capabilities of the Image module.

BMP is common in the Windows environment.

Simple encoding:
encode

See Also
Image, Image.Image and Image.Colortable
Image.BMP._decode
Image.BMP.decode
Image.BMP.decode_header

object decode(string_data)
mapping _decode(string_data)
mapping decode_header(string_data)
object decode(string_data, mapping_options)
mapping _decode(string_data, mapping_options)
mapping decode_header(string_data, mapping_options)

Description
Decode a BMP.
decode gives an image object, _decode gives a mapping in the format

"type":"image/bmp",
"image":image object,
"colortable":colortable object (if applicable)

Returns
the encoded image as a string

Bugs
Doesn’t support all BMP modes. At all.

See Also
encode

Image.BMP.encode

string encode(object_image)
string encode(object_image, mapping_options)
string encode(object_image, object_colortable)
string encode(object_image, int_bpp)

Description
Make a BMP. It default to a 24 bpp BMP file, but if a colortable is given, it will be 8bpp with a palette entry.

option is a mapping that may contain:

"colortable": Image.Colortable - palette
"bpp": 1|4|8|24 - force this many bits per pixel
"rle": 0|1 - run-length encode (default is 0)
CHAPTER 12. IMAGE

Arguments

Returns
the encoded image as a string

Bugs
Doesn’t support old BMP mode, only "windows" mode.

See Also
decode

12.12 Image.GD

Description
Handle encoding and decoding of GD images.

GD is the internal format of libgd by Thomas Boutell, http://www.boutell.com/gd/

It is a rather simple, uncompressed, palette format.

Image.GD._decode, Image.GD.decode, Image.GD.decode_alpha, Image.GD.decode_header

object decode(string data)
object decode_alpha(string data)
mapping decode_header(string data)
mapping _decode(string data)

Description
decodes a GD image

The decode_header and _decode has these elements:

"image":object - image object \n"alpha":object - decoded alpha |- not decode_header
"colortable":object - decoded palette /

Image.GD.encode

string encode(object image)
string encode(object image, mapping options)

Description
encode a GD image

options is a mapping with optional values:
"colortable":object - palette to use (max 256 colors)
"alpha":object - alpha channel (truncated to 1 bit)
"alpha_index":int - index to transparency in palette

12.13 Image.GIF

Description
This submodule keeps the GIF encode/decode capabilities of the Image module.

GIF is a common image storage format, usable for a limited color palette - a
GIF image can only contain at most 256 colors - and animations.

Simple encoding: encode, encode_trans

Advanced stuff: render_block, header_block, end_block, netscape_loop_block

Very advanced stuff: _render_block, _gce_block

See Also
Image, Image.Image and Image.Colortable

Image.GIF.decode

object decode(string data)
object decode(array decoded)

Description
Decodes GIF data and creates an image object.

Returns
the decoded image as an image object

Note
This function may throw errors upon illegal GIF data. This function uses

See Also
code

Image.GIF.decode_layers, Image.GIF.decode_layer

object decode_layers(string data)
object decode_layers(array decoded)
object decode_layer(string data)
object decode_layer(array decoded)

Description
Decodes GIF data and creates an array of layers or the resulting layer.
Note
The resulting layer may not have the same size as the gif image, but the resulting bounding box of all render chunks in the gif file. The offset should be correct, though.

See Also
encode and decode_map

Image.GIF.decode_map

mapping decode_map(INT32 args)

Description
Returns a mapping similar to other decoders _decode function.

"image": the image
"alpha": the alpha channel

Note
The wierd name of this function (not _decode as the other decoders) is because gif was the first decoder and was written before the API was finally defined. Sorry about that. /Mirar

Image.GIF.encode,
Image.GIF.encode_trans

string encode(object img);
string encode(object img, int colors);
string encode(object img, object colortable);
string encode_trans(object img, object alpha);
string encode_trans(object img, int tr_r, int tr_g, int tr_b);
string encode_trans(object img, int colors, object alpha);
string encode_trans(object img, int colors, int tr_r, int tr_g, int tr_b);
string encode_trans(object img, int colors, object alpha, int tr_r, int tr_g, int tr_b);
string encode_trans(object img, int colors, object colortable, object alpha);
string encode_trans(object img, int colors, object colortable, int tr_r, int tr_g, int tr_b);
string encode_trans(object img, int colors, object colortable, object alpha, int a_r, int a_g, int a_b);
string encode_trans(object img, int colors, object colortable, int transp_index);

Description
Create a complete GIF file.

The latter (encode_trans) functions add transparency capabilities.
Example:

```cpp
img=Image.Image([...]);  
[...] // make your very-nice image
write(Image.GIF.encode(img)); // write it as GIF on stdout
```

**Arguments**

**Note**
For advanced users:

```cpp
Image.GIF.encode_trans(img,colortable,trans);
```

is equivalent of using

```cpp
Image.GIF.header_block(img->xsize(),img->ysize(),colortable)+
Image.GIF.render_block(img,colortable,0,0,0,0)+
Image.GIF.end_block();
```

and is actually implemented that way.

**Image.GIF.end_block**

```cpp
string end_block();
```

**Description**
This function gives back a GIF end (trailer) block.

**Returns**
the end block as a string.

**Note**
This is in the advanced sector of the GIF support; please read some about how
GIFs are packed.

The result of this function is always ";" or "x3b", but I recommend using this
function anyway for code clarity.

**See Also**
hdr header and end_block

**Image.GIF.header_block**

```cpp
string header_block(int xsize, int ysize, int numcolors);
string header_block(int xsize, int ysize, object colortable);
string header_block(int xsize, int ysize, object colortable, int background_color_index, int gif87a, int aspectx, int aspecty);
string header_block(int xsize, int ysize, object colortable, int background_color_index, int gif87a, int aspectx, int aspecty, int r, int g, int b);
```

**Description**
This function gives back a GIF header block.
Giving a colortable to this function includes a global palette in the header block.

**Arguments**

*Returns*

the created header block as a string

**Note**

This is in the advanced sector of the GIF support; please read some about how GIFs are packed.

This GIF encoder doesn’t support different size of colors in global palette and color resolution.

**See Also**

header_block and end_block

---

**Image.GIF.netscape_loop_block**

string netscape_loop_block();
string netscape_loop_block(int number_of_loops);

**Description**

Creates a application-specific extention block; this block makes netscape and compatible browsers loop the animation a certain amount of times.

**Arguments**

---

**Image.GIF.render_block**

string render_block(object img, object colortable, int x, int y, int localpalette);
string render_block(object img, object colortable, int x, int y, int localpalette, object alpha);
string render_block(object img, object colortable, int x, int y, int localpalette, object_alpha, int r, int g, int b);
string render_block(object img, object colortable, int x, int y, int localpalette, int delay, int transp_index, int interlace, int user_input, int disposal);
string render_block(object img, object colortable, int x, int y, int localpalette, object alpha, int r, int g, int b, int delay, int interlace, int user_input, int disposal);

**Description**

This function gives a image block for placement in a GIF file, with or without transparency. The some options actually gives two blocks, the first with graphic control extensions for such things as delay or transparency.

**Example:**

```plaintext
img1=Image.Image([...]);
img2=Image.Image([...]);
[...] // make your very-nice images
```
nct=Image.Colortable([], []); // make a nice colortable
write(Image.GIF.header_block(xsize, ysize, nct)); // write a GIF header
write(Image.GIF.render_block(img1, nct, 0, 0, 0, 10)); // write a render block
write(Image.GIF.render_block(img2, nct, 0, 0, 0, 10)); // write a render block
[...] write(Image.GIF.end_block()); // write end block
// voila! A GIF animation on stdout.

The above animation is thus created:

object nct=colortable(lena, 32, {({{0,0,0}})}, []);
string s=GIF.header_block(lena->xsize(), lena->ysize(), nct);
foreach ({{lena->xsize(),
   (int)(lena->xsize() * 0.75),
   (int)(lena->xsize() * 0.5),
   (int)(lena->xsize() * 0.25),
   (int)(i),
   (int)(lena->xsize() * 0.25),
   (int)(lena->xsize() * 0.5),
   (int)(lena->xsize() * 0.75)}}, int xsize)
{
    object o=lena->scale(xsize, lena->ysize());
    object p=lena->clear(0,0,0);
    p->paste(o, (lena->xsize()-o->xsize())/2, 0);
    s+=GIF.render_block(p, nct, 0, 0, 0, 25);
}
s+=GIF.netscape_loop_block(200);
s+=GIF.end_block();
write(s);

Arguments

Note
This is in the advanced sector of the GIF support; please read some about how
GIFs are packed.

The user input and disposal method are unsupported in most decoders.

See Also
encode, header_block and end_block

Image.GIF._decode
array _decode(string _gifdata);
array _decode(array _decoded);
Description
Decodes a GIF image structure down to chunks, and also decode the images in
the render chunks.

{int xsize, int ysize, // 0: size of image drawing area
 void*object colortable, // 2: opt. global colortable
 ({ int aspx, int aspy, // 3 0: aspect ratio or 0, 0 if not set
     int background }, // 2: index of background color

followed by any number these blocks in any order (gce chunks are decoded and
incorporated in the render chunks):

{ GIF.RENDER, // 0: block identifier
   int x, int y, // 1: position of render
   object image, // 3: render image
   void*object alpha, // 4: 0 or render alpha channel
   object colortable, // 5: colortable (may be same as global)

and possibly ended with one of these:

{ GIF.ERROR_PREMATURE_EOD }) // premature end-of-data

The decode method uses this data in a way similar to this program:

import Image;

Arguments

Returns
the above array

Note
May throw errors if the GIF header is incomplete or illegal.

This is in the very advanced sector of the GIF support; please read about how
GIF files works.
**Image.GIF._encode**

string _encode(array, data)

*Description*
Encodes GIF data; reverses _decode._

*Arguments*

*Note*
Some given values in the array are ignored. This function does not give the _exact_ data back!

**Image.GIF._gce_block**

string _gce_block(int transparency, int transparency_index, int delay, int user_input, int disposal);

*Description*
This function gives back a Graphic Control Extension block. A GCE block has the scope of the following render block.

*Arguments*

*Note*
This is in the very advanced sector of the GIF support; please read about how GIF files work.

Most decoders just ignore some or all of these parameters.

*See Also*
_render_block and render_block

**Image.GIF._render_block**

string _render_block(int x, int y, int xsize, int ysize, int bpp, string indices, 0|string colortable, int interlace);

*Description*
Advanced (!) method for writing renderblocks for placement in a GIF file. This method only applies LZW encoding on the indices and makes the correct headers.

*Arguments*

*Note*
This is in the very advanced sector of the GIF support; please read about how GIF files work.

*See Also*
encode, _encode, header_block and end_block
**Image.GIF.decode**

array _decode();

*Description*

Decodes a GIF image structure down to chunks and

\[
\text{((int xsize, int ysize, } \quad // \text{ 0: size of image drawing area}
\]

\[
\text{int numcol, } \quad // \text{ 2: suggested number of colors}
\]

\[
\text{void|string colortable, } \quad // \text{ 3: opt. global colortable}
\]

\[
\text{(( int aspx, int aspy, } \quad // \text{ 4,0: aspect ratio or 0, 0 if not set}
\]

\[
\text{int background })), } \quad // \text{ 1: index of background color}
\]

followed by any number these blocks in any order:

\[
\text{(( GIF.EXTENSION, } \quad // \text{ 0: block identifier}
\]

\[
\text{int extension, } \quad // \text{ 1: extension number}
\]

\[
\text{string data }});} \quad // \text{ 2: extension data}
\]

and possibly ended with one of these:

\[
\text{(( GIF.ERROR, PREMATURE_EOD }}));} \quad // \text{ premature end-of-data}
\]

*Returns*

the above array

*Note*

May throw errors if the GIF header is incomplete or illegal.

This is in the very advanced sector of the GIF support; please read about how GIF files works.

### 12.14 Image.HRZ

*Description*

*Image.HRZ.decode*,

*Image.HRZ.encode*,
Image.HRZ.decode

```
object decode(string data)
mapping _decode(string data)
string encode(object image)
```

**Description**
Handle encoding and decoding of HRZ images. HRZ is rather trivial, and not really useful, but:

The HRZ file is always 256x240 with RGB values from 0 to 63. No compression, no header, just the raw RGB data. HRZ is (was?) used for amateur radio slow-scan TV.

12.15 Image.ILBM

**Description**
This submodule keep the ILBM encode/decode capabilities of the Image module.

**See Also**
Image, Image.Image and Image.Colortable

Image.ILBM.decode

```
object decode(string data)
object decode(array decoded)
object decode(array decoded)
```

**Description**
Decodes ILBM data and creates an image object.

**Returns**
the decoded image as an image object

**Note**
This function may throw errors upon illegal ILBM data. This function uses _decode and decode internally.

**See Also**
encode

Image.ILBM.encode

```
string encode(object image)
string encode(object image, mapping options)
```

**Description**
Encodes an ILBM image.

The options argument may be a mapping containing zero or more encoding options:
normal options:
"alpha": image object
Use this image as mask
(Note: ILBM mask is boolean.
The values are calculated by \((r+2g+b)/4\geq 128\).)

Image.ILBM._decode

array _decode(string|array data)

Description
Decode an ILBM image file.
Result is a mapping,

```javascript
{}
"image": object image,
```

image is the stored image.

Image.ILBM._decode

array _decode();

Description
Decodes an ILBM image structure down to chunks and

```javascript
({
int xsize, int ysize, // 0: size of image drawing area
string bitmapheader, // 2: BMHD chunk
void|string colortable, // 3: opt. colortable chunk (CMAP)
void|string colortable, // 4: opt. colormode chunk (CAMG)
string body, // 5: BODY chunk
mapping more_chunks}) // 6: mapping with other chunks
```

Returns
the above array

Note
May throw errors if the ILBM header is incomplete or illegal.
12.16  Image.PCX

Description

Image.PCX.decode

object decode(string data)

Description
Decodes a PCX image.

Note
Throws upon error in data.

Image.PCX.encode,

Image.PCX._encode

string encode(object image)
string encode(object image, _mapping _options)
string _encode(object image)
string _encode(object image, _mapping _options)

Description
Encodes a PCX image. The _encode and the encode functions are identical

The options argument may be a mapping containing zero or more encoding options:

normal options:
"raw":1
Do not RLE encode the image
"dpy":int
"xdpy":int
"ydpy":int
Image resolution (in pixels/inch, integer numbers)
"xoffset":int
"yoffset":int
Image offset (not used by most programs, but gimp uses it)

Image.PCX._decode

mapping _decode(string data)

Description
Decodes a PCX image to a mapping.

Note
Throws upon error in data.
12.17 Image.PNG

Description

Note
This module uses zlib.

**Image.PNG.decode**

<table>
<thead>
<tr>
<th>object decode(string data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>object decode(string data, mapping options)</td>
</tr>
</tbody>
</table>

Description
Decodes a PNG image.
The **options** argument may be a mapping containing zero or more encoding options:

Note
Throws upon error in data.

**Image.PNG.encode**

<table>
<thead>
<tr>
<th>string encode(object image)</th>
</tr>
</thead>
<tbody>
<tr>
<td>string encode(object image, mapping options)</td>
</tr>
</tbody>
</table>

Description
Encodes a PNG image.
The **options** argument may be a mapping containing zero or more encoding options:

normal options:
"alpha": image object
Use this image as alpha channel
(Note: PNG alpha channel is grey. The values are calculated by \((r+2g+b)/4.)

Note
Please read some about PNG files.

**Image.PNG.chunk**

| string .chunk(string type, string data) |
Description
Encodes a PNG chunk.

Note
Please read about the PNG file format.

Image.PNG._decode

array \_decode(string|array\_data)
array \_decode(string|array\_data, mapping\_options)

Description
Decode a PNG image file.

Result is a mapping,

```
{  
  "image": object image,
```

image is the stored image.

Valid entries in options is a superset of the one given to encode:

basic options:

This method can also take options, as a mapping:

advanced options:
"palette": colortable object
- replace the decoded palette with this when unpacking the image data, if applicable

Note
Please read about the PNG file format. This function ignores any checksum errors in the file. A PNG of higher color resolution than the Image module supports (8 bit) will lose that information in the conversion. It throws an error if the image data is erroneous.
Image.PNG._decode

array _decode(string data)
array _decode(string data, int dontcheckcrc)

Description
Splits a PNG file into chunks.

Result is an array of arrays, ({{ string chunk_type, string data, int crc ok }}, {{ string chunk_type, string data, int crc_ok }}, ... )

chunk_type is the type of the chunk, like "IHDR" or "IDAT".

data is the actual chunk data.

crcok is set to 1 if the checksum is ok and dontcheckcrc parameter isn’t set.

Returns 0 if it isn’t a PNG file.

Note
Please read about the PNG file format.

12.18 Image.PNM

Description
This submodule keeps the PNM encode/decode capabilities of the Image module.

PNM is a common image storage format on unix systems, and is a very simple format.

This format doesn’t use any color palette.

The format is divided into seven subformats:

P1(PBM) - ascii bitmap (only two colors)
P2(PGM) - ascii greymap (only grey levels)
P3(PPM) - ascii truecolor
P4(PBM) - binary bitmap
P5(PGM) - binary greymap
P6(PPM) - binary truecolor

Simple encoding:
encode,
encode_binary,
encode_ascii

Simple decoding:
decode
Advanced encoding:
encode_P1,
encode_P2,
encode_P3,
encode_P4,
encode_P5,
encode_P6

See Also
Image, Image.Image and Image.GIF

**Image.PNM.decode**

object decode(string data)

**Description**
Decodes PNM (PBM/PGM/PPM) data and creates an image object.

**Returns**
the decoded image as an image object

**Note**
This function may throw errors upon illegal PNM data.

See Also
encode

**Image.PNM.encode**,  
**Image.PNM.encode_P1**,  
**Image.PNM.encode_P5**,  
**Image.PNM.encode_ascii**,  
**Image.PNM.encode_P6**,  
**Image.PNM.encode_P2**,  
**Image.PNM.encode_binary**,  
**Image.PNM.encode_P3**,  
**Image.PNM.encode_P4**

string encode(object image)
string encode_binary(object image)
string encode_ascii(object image)
string encode_P1(object image)
string encode_P2(object image)
string encode_P3(object image)
string encode_P4(object image)
string encode_P5(object image)
string encode_P6(object image)

**Description**
Make a complete PNM file from an image.

encode_binary() and encode_ascii() uses the most optimized encoding for this image (bitmap, grey or truecolor) - P4, P5 or P6 respective P1, P2 or P3.
CHAPTER 12. IMAGE

encode

P1/encode

P4

assumes the image is black and white. Use Image.Image-

\texttt{threshold()}

or something like \texttt{Image\_Colortable( \{(0,0,0), (255,255,255)\})}\n
\rightarrow \texttt{floyd\_steinberg()} \rightarrow \texttt{map(my\_image)} to get a black and white image.

encode

P2/encode

P5

assumes the image is greyscale. Use Image.Image-\texttt{grey()} to get a greyscale image.

\textit{Returns}
the encoded image as a string

\textit{Note}
encode() is equal to encode\_binary(), but may change in a future release.

\textit{See Also}
decode

\textbf{12.19 Image.PSD}

\textbf{Description}

\textbf{12.20 Image.TGA}

\textbf{Description}

\textbf{Image.TGA.decode}

\texttt{object decode(string data)}

\textbf{Description}
Decodes a Targa image.

\textit{Note}
Throws upon error in data.

\textbf{Image.TGA.encode}

\texttt{string encode(object image)}
\texttt{string encode(object image, mapping options)}

\textbf{Description}
Encodes a Targa image.

The \texttt{options} argument may be a mapping containing zero or more encoding options:

\texttt{normal options:}
"alpha": image object
12.21. IMAGE.XBM

Use this image as alpha channel
(Note: Targa alpha channel is grey.
The values are calculated by \((r+2g+b)/4\).)

Image.TGA._decode

object _decode(string data)

Description
Decodes a Targa image to a mapping. The mapping follows this format:
(["image":img_object,"alpha":alpha_channel])

Note
Throws upon error in data.

12.21 Image.XBM

Description

Image.XBM.decode

object decode(string data)

Description
Decodes a XBM image.

Note
Throws upon error in data.

Image.XBM.encode

string encode(object image)
string encode(object image, mapping_options)

Description
Encodes a XBM image.

The options argument may be a mapping containing zero or more encoding options.

normal options:
"name":"xbm_image_name"
The name of the XBM. Defaults to ‘image’
CHAPTER 12. IMAGE

**Image.XBM.decode**

object \_decode(string \_data)

object \_decode(string \_data, \_mapping\_options)

*Description*

Decodes a XBM image to a mapping.

Supported options:

```javascript
{
"fg":{fgcolor}, // Foreground color. Default black
"bg":{bgcolor}, // Background color. Default white
"invert":1, // Invert the mask
}
```

*Note*

Throws upon error in data.

### 12.22 Image.XCF

*Description*

**Image.XCF.decode**

object decode(string \_data)

*Description*

Decodes a XCF image to a single image object.

*Note*

Throws upon error in data, you will lose quite a lot of information by doing this. See Image.XCF.decode and Image.XCF.decode

**Image.XCF.decode_layers**

array(object) decode_layers(\_string\_data)

*Description*

Decodes a XCF image to an array of Image.Layer objects

The layer object have the following extra variables (to be queried using get_m misc_value):

image.xres, image.yres, image.colormap, image.guides, image.parasites, name, parasites, visible, active
**Image.XCF._decode**

`mapping _decode(string|object data, mapping|void options)`

**Description**
Decodes a XCF image to a mapping, with at least an 'image' and possibly an 'alpha' object. Data is either a XCF image, or a XCF.GimpImage object structure (as received from _decode)

**Supported options**

```
{
  "background":(r,g,b)||Image.Color object
  "draw_all_layers":1,
  Draw invisible layers as well
```

**Note**
Throws upon error in data. For more information, see Image.XCF._decode

**Image.XCF._decode**

`object _decode(string|mapping data, _mapping|void options)`

**Description**
Decodes a XCF image to a Image.XCF.GimpImage object.

**Returned structure reference**

**Image.XCF._decode**

`object _decode(string|mapping data)`

**Description**
Decodes a XCF image to a mapping.

**Structure reference**
Description
This submodule keeps the XWD (X Windows Dump) decode capabilities of the Image module.

XWD is the output format for the xwd program.

Simple decoding:
decode

Advanced decoding:
_decode

See Also

Image.XWD.decode

object decode(string data)

Description
Simple decodes a XWD image file.

Image.XWD._decode,
Image.XWD.decode_header

mapping _decode(string data)

mapping decode_header(string data)

Description
Decodes XWD data and returns the result.

Supported XWD visual classes and encoding formats are TrueColor / ZPixmap
DirectColor / ZPixmap PseudoColor / ZPixmap

If someone sends me files of other formats, these formats may be implemented.
:)/mirar@idonex.se

Returns
the decoded image as an image object

Note
This function may throw errors upon illegal or unknown XWD data.

See Also
decode

12.24 Image.JPEG

Description

Note
This module uses libjpeg, a software from Independent JPEG Group.
Image.JPEG.decode,

Description
Decodes a JPEG image. The simple decode function simply gives the image object, the other functions gives a mapping of information (see below)

The options argument may be a mapping containing zero or more encoding options:

advanced options:
"block_smoothing":0|1
Do interblock smoothing. Default is on (1).
"fancy_upsampling":0|1
Do fancy upsampling of chroma components. Default is on (1).
"method":JPEG.IFAST|JPEG.ISLOW|JPEG.FLOAT|JPEG.DEFAULT|JPEG.FASTEST
DCT method to use.
DEFAULT and FASTEST is from the jpeg library, probably ISLOW and IFAST respective.

_decode and decode_header gives a mapping as result, with this content:

"xsize":int
"ysize":int
size of image
"xdpi":float
"ydpi":float
image dpi, if known
"type":"image/jpeg"
file type information as MIME type

Note
Please read some about JPEG files.
Image.JPEG.encode

string encode(object image)
string encode(object image, mapping options)

Description
Encodes a JPEG image.

The options argument may be a mapping containing zero or more encoding options:

- "quality": 0..100
  Set quality of result. Default is 75.
- "optimize": 0|1
  Optimize Huffman table. Default is on (1) for images smaller than 50k pixels.
- "progressive": 0|1
  Make a progressive JPEG. Default is off.

Note
Please read some about JPEG files. A quality setting of 100 does not mean the result is lossless.

12.25 Image.TIFF

Description

Image.TIFF.decode

object decode(string data)

Description
Decodes a TIFF image.

Note
Throws upon error in data.

Image.TIFF.encode

string encode(object image)
string encode(object image, mapping options)
string encode(object image)
string encode(object image, mapping options)
**Description**

encode and _encode are identical.

The options argument may be a mapping containing zero or more encoding options:

normal options:
"compression":Image.TIFF.COMPRESSION_*,
"name":"an image name",
"comment":"an image comment",
"alpha":An alpha channel,
"dpy":Dots per inch (as a float),
"xdpy":Horizontal dots per inch (as a float),
"ydpy":Vertical dots per inch (as a float),

**Image.TIFF._decode**

mapping _decode(string data)

**Description**

Decodes a TIFF image to a mapping with at least the members image and alpha.

**Note**

Throws upon error in data.

**12.26 Image.TTF**

**Description**

This module adds TTF (TrueType font) capability to the Image module.

**Note**

This module needs the libttf "Freetype" library

**Image.TTF.**

object ‘() (string filename)
object ‘() (string filename, mapping options)

**Description**

Makes a new TTF Face object.

**Arguments**

**Returns**

0 if failed.
12.26.1 Image.TTF.Face

**Image.TTF.Face**

*Description*

This represents instances of TTF Faces.

**Image.TTF.Face.flush**

object flush()

*Description*

This flushes all cached information. Might be used to save memory - the face information is read back from disk upon need.

*Returns*

the called object

**Image.TTF.Face.names**, **Image.TTF.Face._names**

mapping names()

array(array) _names()

*Description*

Gives back the names or the complete name-list of this face.

The result from names() is a mapping, which has any or all of these indices:

```
"copyright":  ("Copyright the Foo Corporation...bla bla")
"family":     ("My Little Font")
"style":      ("Bold")
"full":       ("Foo: My Little Font: 1998")
"expose":     ("My Little Font Bold")
"version":    ("June 1, 1998; 1.00, ...")
"postscript": ("MyLittleFont-Bold")
"trademark":  ("MyLittleFont is a registered...bla bla")
```

This is extracted from the information from _names(), and fit into pike-strings using unicode or iso-8859-1, if possible.

The result from _names() is a matrix, on this form:

```
{ ( { int platform, encoding, language, id, string name } ),
  { int platform, encoding, language, id, string name } ),
  ...
}
```
Returns
the name as a mapping to string or the names as a matrix

Note
To use the values from _names(), check the TrueType standard documentation.

**Image.TTF.Face.properties**

mapping properties()

Description
This gives back a structure of the face’s properties. Most of this stuff is information you can skip.

The most interesting item to look at may be ->numFaces, which describes the number of faces in a .TTC font collection.

Returns
a mapping of a lot of properties

**Image.TTF.Face.‘**

object ‘() ()

Description
This instantiates the face for normal usage - to convert font data to images.

Returns
a Image.TTF.FaceInstance object.

12.26.2 **Image.TTF.FaceInstance**

**Image.TTF.FaceInstance**

Description
This is the instance of a face, with geometrics, encodings and stuff.

**Image.TTF.FaceInstance.create**

void create(object face)

Description
creates a new Instance from a face.

12.27 **Image.XFace**

Description

Note
This module uses libgmp.
**Image.XFace.decode**

object decode(string data)
object decode(string data, mapping options)

*Description*
Decodes an X-Face image.

The `options` argument may be a mapping containing zero options.

**Image.XFace.decode_header**

object decode_header(string data)
object decode_header(string data, mapping options)

*Description*
Decodes an X-Face image header.

"xsize": int
"ysize": int
size of image
"type": "image/x-xface"

file type information

The `options` argument may be a mapping containing zero options.

*Note*
There ain’t no such thing as a X-Face image header. This stuff tells the characteristics of an X-Face image.

**Image.XFace.encode**

string encode(object img)
string encode(object img, mapping options)

*Description*
Encodes an X-Face image.

The `img` argument must be an image of the dimensions 48 by 48 pixels. All non-black pixels will be considered white.

The `options` argument may be a mapping containing zero options.
Chapter 13

Protocols

The Protocol modules is some helper modules that makes it easier for the pike programmer to use some of the protocols used on the internet.

13.1 Protocols.HTTP

Protocols.HTTP.delete_url

object(Protocols.HTTP.Query) delete_url(string url)
object(Protocols.HTTP.Query) delete_url(string url, mapping query variables)
object(Protocols.HTTP.Query) delete_url(string url, mapping query variables, mapping request headers)

Description
Sends a HTTP DELETE request to the server in the URL and returns the created and initialized Query object. 0 is returned upon failure.

Protocols.HTTP.get_url

object(Protocols.HTTP.Query) get_url(string url)
object(Protocols.HTTP.Query) get_url(string url, mapping query variables)
object(Protocols.HTTP.Query) get_url(string url, mapping query variables, mapping request headers)

Description
Sends a HTTP GET request to the server in the URL and returns the created and initialized Query object. 0 is returned upon failure.

Protocols.HTTP.get_url_nice
**Protocols.HTTP.get_url_data**

```plaintext
array(string) get_url.nice(string_url)
array(string) get_url.nice(string_url, mapping_query_variables)
array(string) get_url.nice(string_url, mapping_query_variables, _mapping_request_headers)
string get_url.data(string_url)
string get_url.data(string_url, mapping_query_variables)
string get_url.data(string_url, mapping_query_variables, _mapping_request_headers)
```

**Description**

Returns an array of (“content_type,data”) and just the data string respective, after calling the requested server for the information. 0 is returned upon failure.

**Protocols.HTTP.post_url.nice, Protocols.HTTP.post_url_data, Protocols.HTTP.post_url**

```plaintext
array(string) post_url.nice(string_url, mapping_query_variables)
array(string) post_url.nice(string_url, mapping_query_variables, _mapping_request_headers)
string post_url.data(string_url, mapping_query_variables)
string post_url.data(string_url, mapping_query_variables, _mapping_request_headers)
oobject(Protocols.HTTP.Query) post_url(string_url, mapping_query_variables)
oobject(Protocols.HTTP.Query) post_url(string_url, mapping_query_variables, _mapping_request_headers)
```

**Description**

Similar to the get_url class of functions, except that the query variables is sent as a post request instead of a get.

**Protocols.HTTP.put_url**

```plaintext
object(Protocols.HTTP.Query) put_url(string_url)
object(Protocols.HTTP.Query) put_url(string_url, string_file)
object(Protocols.HTTP.Query) put_url(string_url, string_file, mapping_query_variables)
object(Protocols.HTTP.Query) put_url(string_url, string_file, mapping_query_variables, _mapping_request_headers)
```

**Description**

Sends a HTTP PUT request to the server in the URL and returns the created and initialized Query object. 0 is returned upon failure.

**Protocols.HTTP.unentity**

```plaintext
string unentity(string_s)
```

**Description**

Helper function for replacing HTML entities with the corresponding iso-8859-1
13.1. PROTOCOLS.HTTP

Note
All characters isn’t replaced, only those with corresponding iso-8859-1 characters.

13.1.1 Protocols.HTTP.Query

Protocols.HTTP.Query

Description
Open and execute a HTTP query.

Protocols.HTTP.Query.set_callbacks
Protocols.HTTP.Query.async_request

object set_callbacks(function request_ok, function request_fail, mixed,...extra)
object async_request(string server, int port, string query);
object async_request(string server, int port, string query,
mapping headers, void|string data);

Description
Setup and run an asynchronous request, otherwise similar to thread_request.

request_ok(object httpquery,...extra args) will be called when connection is complete, and headers are parsed.

request_fail(object httpquery,...extra args) is called if the connection fails.

variable int ok Tells if the connection is successfull. variable int errno Errno copied from the connection.

variable mapping headers Headers as a mapping. All header names are in lower case, for convinience.

variable string protocol Protocol string, ie "HTTP/1.0".

variable int status variable string status_desc Status number and description (ie, 200 and "ok").

variable mapping hostname_cache Set this to a global mapping if you want to use a cache, prior of calling *request().

variable mapping async_dns Set this to an array of Protocols.DNS.async_clients, if you wish to limit the number of outstanding DNS requests. Example: async_dns=allocate(20,Protocols.DNS.async_client);

Returns
the called object

Protocols.HTTP.Query.cast

array cast("array")
Description
Gives back (mapping headers,string data, string protocol,int status,string status_desc);

Protocols.HTTP.Query.cast
mapping cast("mapping")

Description
Gives back headers — 
("protocol":protocol,"status":status number,"status_desc":status description,"data":data);

Protocols.HTTP.Query.cast
string cast("string")

Description
Gives back the answer as a string.

Protocols.HTTP.Query.data
string data()

Description
Gives back the data as a string.

Protocols.HTTP.Query.downloaded_bytes
int downloaded_bytes()

Description
Gives back the number of downloaded bytes.

Protocols.HTTP.Query.thread_request
object thread_request(string server, int port, string query);
object thread_request(string server, int port, string query, 
mapping headers, void|string data);

Description
Create a new query object and begin the query.

The query is executed in a background thread; call () in this object to wait for the request to complete.

‘query’ is the first line sent to the HTTP server; for instance "GET /index.html HTTP/1.1".

headers will be encoded and sent after the first line, and data will be sent after the headers.

Returns
the called object
Protocols.HTTP.Query.total_bytes

```c
int total_bytes()
```

**Description**
Gives back the size of a file if a content-length header is present and parsed at the time of evaluation. Otherwise returns -1.

```c
object(pseudofile) file() object(pseudofile) file(mapping newheaders, void—mapping removeheaders)
object(pseudofile) datafile();
```

Gives back a pseudo-file object, with the method `read()` and `close()`. This could be used to copy the file to disc at a proper tempo.

datafile() doesn’t give the complete request, just the data.

```c
newheaders, removeheaders is applied as: (oldheaders|newheaders)-removeheaders
```

Make sure all new and remove-header indices are lower case.

```c
void async_fetch(function done, callback);
```

Fetch all data in background.

Protocols.HTTP.Query.

```c
int '()'()
```

**Description**
Wait for connection to complete.

**Returns**
1 on successful connection, 0 if failed

13.2 Protocols.LysKOM

13.2.1 Protocols.LysKOM.Session

Protocols.LysKOM.Session

**Description**
variable user This variable contains the person that are logged in.

Protocols.LysKOM.Session.create

```c
void create(string server)
void create(string server, mapping options)
```

**Description**
Initializes the session object, and opens a connection to that server.

options is a mapping of options,

```c
{
    "login" : int|string login as this person number
    (get number from name)
    "create" : string
```
create a new person and login with it

"password" : string  send this login password
"invisible" : int(0..1) if set, login invisible
advanced
"port" : int(0..65535)  server port (default is 4894)
"whoami" : string  present as this user
    (default is from uid/getpwent and hostname)
}

See Also

Connection

Protocols.LysKOM.Session.create_person

object create_person(string name, string password)

Description

Create a person, which will be logged in.

Returns

the new person object

Protocols.LysKOM.Session.create_text

object create_text(string subject, string body, mapping options)
object create_text(string subject, string body, mapping options,
  function callback, mixed ...extra)

Description

Creates a new text.

if "callback" are given, this function will be called when the text is created, with
the text as first argument. Otherwise, the new text is returned.

options is a mapping that may contain:

{
  "recpt" : Conference|array(Conference)
    recipient conferences
  "cc" : Conference|array(Conference)
    cc-recipient conferences
  "bcc" : Conference|array(Conference)
    bcc-recipient conferences *

  "comm_to" : Text|array(Text)
    what text(s) is commented
  "foot_to" : Text|array(Text)
    what text(s) is footnoted

  "anonymous" : int(0..1)
    send text anonymously
}
Note
The above marked with a '*' is only available on a protocol 10 server. A LysKOM error will be thrown if the call fails.

See Also
Conference.create.text, Text.comment and Text.footnote

Protocols.LysKOM.Session.login

object login(int user_no, string password)
object login(int user_no, string password, int invisible)

Description
Performs a login. Returns 1 on success or throws a lyskom error.

Returns
the called object

Protocols.LysKOM.Session.logout

object logout()

Description
Logouts from the server.

Returns
the called object

Protocols.LysKOM.Session.send_message

object send_message(string message, mapping options)

Description
Sends a message.

options is a mapping that may contain:

{(  "recpt" : Conference recipient conference  )}

Protocols.LysKOM.Session.try_complete_person

array(object) try_complete_person(string orig)

Description
Runs a LysKOM completion on the given string, returning an array of confzinfos of the match.
13.2.2 Protocols.LysKOM.Connection

Description
This class contains nice abstraction for calls into the server. They are named "call", "async_call" or "async_cb_call", depending on how you want the call to be done.


mixed /call/(mixed...args)
object async_call/(mixed...args)
object async_cb_call/(function, callback, mixed...args)

Description
Do a call to the server. This really clones a request object, and initialises it. /call/ is to be read as one of the calls in the lyskom protocol. ("-" is replaced with ".") (ie, logout, async_login or async_cb_get_conf_stat.)

The first method is a synchronous call. This will send the command, wait for the server to execute it, and then return the result.

The last two is asynchronous calls, returning the initialised request object.

variable int protocol_level variable string session variable string software_version Description of the connected server.

Protocols.LysKOM.Connection.create

void create(string, server)
void create(string, server, mapping, options)

Description


def

{
    "login" : int|string login as this person number
    (get number from name)
    "password" : string send this login password
    "invisible" : int(0..1) if set, login invisible advanced
    "port" : int(0..65535) server port (default is 4894)
    "whoami" : string present as this user
    (default is from uid/getpwent and hostname)
}

13.2.3 Protocols.LysKOM.Request
Description
This class contains nice abstraction for calls into the server. They are named "call", "async_call" or "async_cb_call", depending on how you want the call to be done.

Protocols.LysKOM.Request._Request

Protocols.LysKOM.Request._Request

Description
This is the main request class. All lyskom request classes inherits this class.

Protocols.LysKOM.Request._Request.async , Protocols.LysKOM.Request._Request.sync

void async(mixed...args)
mixed sync(mixed...args)

Description
initialise an asynchronous or a synchronous call, the latter is also evaluating the result. This calls 'indata' in itself, to get the correct arguments to the lyskom protocol call.

Protocols.LysKOM.Request._Request._reply , Protocols.LysKOM.Request._Request.reply

mixed _reply(object|array what)
mixed reply(object|array what)

Description
_reply is called as callback to evaluate the result, and calls reply in itself to do the real work.

Protocols.LysKOM.Request._Request.'
mixed '()'()

Description
wait for the call to finish.

variable int ok tells if the call is executed ok variable object error how the call failed The call is completed if (ok——error).

Protocols.LysKOM.Request._Request._async , Protocols.LysKOM.Request._Request._sync

void _async(int call, _mixed data)
mixed _sync(int call, _mixed data)

Description
initialise an asynchronous or a synchronous call, the latter is also evaluating the result. These are called by async and sync respectively.
13.3 Protocols.DNS

Description

13.3.1 Protocols.DNS.client

**Protocols.DNS.client**

Description
Synchronous DNS client.

**Protocols.DNS.client.create**

void create()

void create(void|string|array server, void|int|array domain)

Description

**Protocols.DNS.client.gethostbyname, Protocols.DNS.client.gethostbyaddr**

array gethostbyname(string hostname)
array gethostbyaddr(string hostip)

Description
Querys the host name or ip from the default or given DNS server. The result is a mapping with three elements,

```
{
    string hostname [0] hostname
    array(string) ip [1] ip number(s)
    array(string) ip [2] dns name(s)
}
```

**Protocols.DNS.client.get_primary_mx**

string get_primary_mx(string hostname)

Description
Querys the primary mx for the host.

Returns
the hostname of the primary mail exchanger
Chapter 14

Other modules

Pike also include a number of smaller modules. These modules implement support for various algorithms, data structures and system routines.

14.1 System

The system module contains some system-specific functions that may or may not be available on your system. Most of these functions do exactly the same thing as their UNIX counterpart. See the UNIX man pages for detailed information about what these functions do on your system.

Please note that these functions are available globally, you do not need to import System to use these functions.

chroot - change the root directory

int chroot(string newroot);
int chroot(object(File) obj);

Description
Changes the root directory for this process to the indicated directory.

Note
Since this function modifies the directory structure as seen from Pike, you have to modify the environment variables PIKE_MODULE_PATH and PIKE_INCLUDE_PATH to compensate for the new root-directory.

This function only exists on systems that have the chroot(2) system call. The second variant only works on systems that also have the fchroot(2) system call.

getegid - get the effective group ID

int getegid();
CHAPTER 14. OTHER MODULES

Description
Get the effective group ID.

See Also
setuid, getuid, setgid, getgid, seteuid, geteuid and setegid

groupeuid - get the effective user ID

int geteuid();

Description
Get the effective user ID.

See Also
setuid, getuid, setgid, getgid, seteuid, geteuid and setegid

groupegid - get the group ID

int getgid();

Description
Get the real group ID.

See Also
setuid, getuid, setgid, getgid, seteuid, geteuid and setegid

groupegroups - get the supplemental group access list

array(int) getgroups();

Description
Get the current supplemental group access list for this process.

See Also
initgroups, setgroups, getgid, setgid, getegid and setegid

guposethostbyaddr - gets information about a host given its address

array gethostbyaddr(string addr);

Description
Returns an array with information about the specified IP address.

The returned array contains the same information as that returned by gethostbyname().

Note
This function only exists on systems that have the gethostbyaddr(2) or similar system call.

See Also
gethostbyname
**gethostbyname** - gets information about a host given its name

```c
array gethostbyname(string hostname);
```

**Description**
Returns an array with information about the specified host.

The array contains three elements:

The first element is the hostname.

The second element is an array(string) of IP numbers for the host.

The third element is an array(string) of aliases for the host.

**Note**
This function only exists on systems that have the gethostbyname(2) or similar system call.

**See Also**
gethostbyaddr

**gethostname** - get the name of this host

```c
string gethostname();
```

**Description**
Returns a string with the name of the host.

**Note**
This function only exists on systems that have the gethostname(2) or uname(2) system calls.

**getpgid** - get the process group ID

```c
int getpgid();
int getpgid(int pid);
```

**Description**
With no arguments or with `pid` equal to zero, returns the process group ID of this process.

If `pid` is specified, returns the process group ID of that process.

**See Also**
getpid and getppid

**getpid** - get the process ID

```c
int getpid();
```

**Description**
Returns the process ID of this process.
See Also
getppid and getpgrp

getppid - get the parent process ID
int getppid();

Description
Returns the process ID of the parent process.

See Also
getpid and getpgid

getpid - get the user ID
int getpid();

Description
Get the real user ID.

See Also
setuid, setgid, getgid, seteuid, geteuid, setegid and getegid

hardlink - create a hardlink
void hardlink(string from, string to);

Description
Creates a hardlink named to from the file from.

See Also
symlink, mv and rm

initgroups - initialize the group access list
void initgroups(string username, int base_gid);

Description
Initializes the group access list according to the system group database. base_gid is also added to the group access list.

See Also
setuid, getuid, setgid, getgid, seteuid, geteuid, setegid, getegid, getgroups and setgroups

openlog - initializes the connection to syslogd
void openlog(string ident, int options, facility);

Description
Initializes the connection to syslogd.

The ident argument specifies an identifier to tag all log entries with.
options is a bit field specifying the behavior of the message logging. Valid options are:

facility specifies what subsystem you want to log as. Valid facilities are:

Note
Only available on systems with syslog(3).

Bugs
LOG_NOWAIT should probably always be specified.

See Also
syslog, closelog and setlogmask

readlink - read a symbolic link

string readlink(string linkname);

Description
Returns what the symbolic link linkname points to.

See Also
symlink

setegid - set the effective group ID

void setegid(int uid);

Description
Sets the effective group ID to gid.

See Also
setuid, getuid, setgid, getgid, seteuid, geteuid and getegid

seteuid - set the effective user ID

void seteuid(int uid);

Description
Sets the effective user ID to uid.

See Also
setuid, getuid, setgid, getgid, geteuid, setegid and getegid
CHAPTER 14. OTHER MODULES

**setgid** - set the group ID

```c
void setgid(int gid);
```

**Description**

Sets the real group ID, effective group ID and saved group ID to `gid`.

**See Also**

* `setuid`, `getuid`, `getgid`, `seteuid`, `geteuid`, `setegid` and `getegid`

---

**setgroups** - set the supplemental group access list

```c
void getgroups(array(int) gids);
```

**Description**

Set the supplemental group access list for this process.

**See Also**

* `initgroups`, `getgroups`, `getgid`, `setgid`, `getegid` and `setegid`

---

**setuid** - set the user ID

```c
void setuid(int uid);
```

**Description**

Sets the real user ID, effective user ID and saved user ID to `uid`.

**See Also**

* `getuid`, `setgid`, `getgid`, `seteuid`, `geteuid`, `setegid` and `getegid`

---

**symlink** - create a symbolic link

```c
void symlink(string from, string to);
```

**Description**

Creates a symbolic link for an original file `from` with the new name `to`.

**See Also**

* `hardlink`, `readlink`, `mv` and `rm`

---

**uname** - get operating system information

```c
mapping(string:string) uname();
```

**Description**

Returns a mapping describing the operating system.

The mapping contains the following fields:

- "sysname": Operating system name
- "nodename": Host name
- "release": Release of this OS
- "version": Version number of this OS
- "machine": Machine architecture
Note
This function only exists on systems that have the `uname(2)` system call.

## 14.2 Process

The Process module contains functions to start and control other programs from Pike.

**Process.create_process** - Create a new process

```pascal
Process.create_process(array(string) command, void | mapping(string:mixed) modifiers);
```

**Description**

This is the recommended and most portable way to start processes in Pike. The command name and arguments are sent as an array of strings so that you do not have to worry about quoting them. The optional mapping `modifiers` can contain zero or more of the following parameters:

The following options are only available on some systems.
These parameter limits the new process in certain ways, they are not available on all systems. Each of these can either be given as a mapping \(["soft" : soft\_limit,"hard":hard\_limit]\) as an integer. If an integer is given both the hard limit and the soft limit will be changed. The new process may change the soft limit, but may not change it any higher than the hard limit.

See Also

Process.popen and Process.system
Process.create_process.set_priority - Set the priority of this process

int set_priority(string pri);

Description
This function sets the priority of this process, the string pri should be one of:

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;realtime&quot; or &quot;highest&quot;</td>
<td>This gives the process realtime priority, this basically gives the process access to the cpu whenever it wants.</td>
</tr>
<tr>
<td>&quot;high&quot;</td>
<td>This gives your process a little bit higher priority than what is 'normal'.</td>
</tr>
<tr>
<td>&quot;normal&quot;</td>
<td>This sets the process' priority to whatever is 'normal' for new processes on your system.</td>
</tr>
<tr>
<td>&quot;low&quot;</td>
<td>This will give the process a lower priority than what is normal on your system.</td>
</tr>
<tr>
<td>&quot;lowest&quot;</td>
<td>This will give the process a lower priority than most processes on your system.</td>
</tr>
</tbody>
</table>

Process.create_process.wait - Wait for this process to finish

int wait();

Description
This function makes the calling thread wait for the process to finish. It returns the exit code of that process.

Process.create_process.status - Check if process is still running

int status();

Description
This function returns zero if the process is still running. If the process has exited, it will return one.

Process.create_process.pid - Get the process id of this process

int pid();

Description
This function returns the process identifier for the process.

Process.create_process.kill - Kill this process

int kill(int signal);

Description
This function sends the given signal to the process, it returns one if the operation succeeded, zero otherwise.

See Also
signum
**Process.popen** - pipe open

```pascal
string popen(string cmd);
```

**Description**
This function runs the command `cmd` as in a shell and returns the output. See your Unix/C manual for details on `popen`.

**Process.system** - run an external program

```pascal
void system(string cmd);
```

**Description**
This function runs the external program `cmd` and waits until it is finished. Standard `/bin/sh` completions/redirections/etc. can be used.

**See Also**

**Process.spawn** - spawn a process

```pascal
int spawn(string cmd);
int spawn(string cmd, object stdin);
int spawn(string cmd, object stdin, object stdout);
int spawn(string cmd, object stdin, object stdout, object stderr);
```

**Description**
This function spawns a process but does not wait for it to finish. Optionally, clones of Stdio.File can be sent to it to specify where stdin, stdout and stderr of the spawned processes should go.

**See Also**

**Process.exece** - execute a program

```pascal
int exece(string file, array(string) args);
int exece(string file, array(string) args, mapping(string:string) env);
```

**Description**
This function transforms the Pike process into a process running the program specified in the argument `file` with the argument `args`. If the mapping `env` is present, it will completely replace all environment variables before the new program is executed. This function only returns if something went wrong during `exece()`, and in that case it returns zero.

**Note**
The Pike driver `_dies_` when this function is called. You must use `fork()` if you
wish to execute a program and still run the Pike driver.

**Example**

execute("/bin/ls", {"-l"});
execec("/bin/sh", {"-c", "echo $HOME"}, {"HOME":"/not/home"});

**See Also**

fork and Stdio.File->pipe

**Process.exe** - simple way to use exece()

```c
int exec(string file, string ... args);
```

**Description**

This function destroys the Pike parser and runs the program `file` instead with the arguments. If no there are no `/` in the filename, the variable PATH will be consulted when looking for the program. This function does not return except when the exec fails for some reason.

**Example**

exec("/bin/echo","hello","world");

**Process.Spawn** - spawn off another program with control structure

```c
object Process.Spawn(string program, void|array(string) args, void|mapping(string:string) env, void|string cwd);
```

or

```c
object Process.Spawn(string program, void|array(string) args, void|mapping(string:string) env, void|string cwd, array(void|object(Stdio.file)) fds, void|array(void|object(Stdio.file)) fds_to_close);
```

**Description**

Spawn off another program (`program`) and creates the control structure. This does not spawn off a shell to find the program, therefore must `program` be the full path the the program.

`args` is per default no argument(s),

`env` is reset to the given mapping, or kept if the argument is 0. `fds` is your stdin, stdout and stderr. Default is local pipes (see stdin et al). `fds_to_close` is file descriptors that are closed in the forked branch when the program is executed (ie, the other end of the pipes).


```c
object(Stdio.File) stdin;
object(Stdio.File) stdout;
object(Stdio.File) stderr;
array(void|object(Stdio.File)) fd;
```
CHAPTER 14. OTHER MODULES

Description
Pipes to the spawned program (if set). fd[0]==stdin, etc.

Process.Spawn.pid - spawned program pid
int pid;

Description
pid of the spawned program.

Process.Spawn.wait - wait for the spawned program to finish
void wait();

Description
Returns when the program has exited.

Process.Spawn.kill - send a signal to the program
int kill(int signal);

Description
Sends a signal to the program.

14.3 Regexp

Regexp is short for Regular Expression. A regular expression is a standardized way to make pattern that match certain strings. In Pike you can often use the sscanf, range and index operators to match strings, but sometimes a regexp is both faster and easier.

A regular expression is actually a string, then compiled into an object. The string contains characters that make up a pattern for other strings to match. Normal characters, such as A through Z only match themselves, but some characters have special meaning.

<table>
<thead>
<tr>
<th>pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>any one character</td>
</tr>
<tr>
<td>[abc]</td>
<td>a, b or c</td>
</tr>
<tr>
<td>[a-z]</td>
<td>any character a to z inclusive</td>
</tr>
<tr>
<td>[^ac]</td>
<td>any character except a and c</td>
</tr>
<tr>
<td>(x)</td>
<td>x (x might be any regexp) If used with split, this also puts the string matching x into the result array.</td>
</tr>
<tr>
<td>x*</td>
<td>zero or more occurrences of 'x' (x may be any regexp)</td>
</tr>
<tr>
<td>x+</td>
<td>one or more occurrences of 'x' (x may be any regexp)</td>
</tr>
<tr>
<td>x—y</td>
<td>x or y. (x or y may be any regexp)</td>
</tr>
<tr>
<td>xy</td>
<td>xy (x and y may be any regexp)</td>
</tr>
<tr>
<td>$</td>
<td>beginning of string (but no characters)</td>
</tr>
<tr>
<td>$</td>
<td>end of string (but no characters)</td>
</tr>
</tbody>
</table>
14.3. REEXP

<table>
<thead>
<tr>
<th>pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>the beginning of a word (but no characters)</td>
</tr>
<tr>
<td>\b</td>
<td>the end of a word (but no characters)</td>
</tr>
</tbody>
</table>

Let’s look at a few examples:

<table>
<thead>
<tr>
<th>Regexp</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0-9]+</td>
<td>one or more digits</td>
</tr>
<tr>
<td><code>t</code>n</td>
<td>exactly one non-whitespace character</td>
</tr>
<tr>
<td>(foo)(bar)</td>
<td>either 'foo' or 'bar'</td>
</tr>
<tr>
<td>.html$</td>
<td>any string ending in '.html'</td>
</tr>
<tr>
<td>.</td>
<td>any string starting with a period</td>
</tr>
</tbody>
</table>

Note that `" can be used to quote these characters in which case they match themselves, nothing else. Also note that when quoting these something in Pike you need two `" because Pike also uses this character for quoting.

To make make regexps fast, they are compiled in a similar way that Pike is, they can then be used over and over again without needing to be recompiled. To give the user full control over the compilations and use of regexp an object oriented interface is provided.

You might wonder what regexps are good for, hopefully it should be more clear when you read about the following functions:

**Regexp.create** - compile regexp

```pke
void create();
void create(string regexp);
object(Regexp) Regexp();
object(Regexp) Regexp(string regexp);
```

**Description**

When create is called, the current regexp bound to this object is cleared. If a string is sent to create(), this string will be compiled to an internal representation of the regexp and bound to this object for later calls to match or split. Calling create() without an argument can be used to free up a little memory after the regexp has been used.

**See Also**

clone and Regexp->match

**Regexp.match** - match a regexp

```pke
int match(string s)
```

**Description**

Return 1 if $s$ matches the regexp bound to the object regexp, zero otherwise.

**See Also**

Regexp->create and Regexp->split
**Regexp.split** - split a string according to a pattern

`array(string) split(string s)`

*Description*
Works as `regexp`-match, but returns an array of the strings that matched the sub-regexps. Sub-regexps are those contained in `( )` in the regexp. Sub-regexps that were not matched will contain zero. If the total regexp didn’t match, zero is returned.

*Bugs*
You can only have 40 sub-regexps.

*See Also*
`Regexp->create` and `Regexp->match`

### 14.4 Gmp

Gmp is short for GNU Multi-Precision library. It is a set of routines that can manipulate very large numbers. Although much slower than regular integers they are very useful when you need to handle extremely large numbers. Billions and billions as Mr Attenborough would have said.

The Gmp library can handle large integers, floats and rational numbers, but currently Pike only has support for large integers. The others will be added later or when demand arises. Large integers are implemented as objects cloned from Gmp.Mpz.

**Gmp.mpz** - bignum program

*Description*
Gmp.mpz is a builtin program written in C. It implements large, very large integers. In fact, the only limitation on these integers is the available memory.

The mpz object implements all the normal integer operations. (except xor) There are also some extra operators:

*Note*
This module is only available if libgmp.a was available and found when Pike was compiled.

**Gmp.mpz.create** - initialize a bignum

```
object Mpz();
object Mpz(int|object|float i);
object Mpz(string digits, int base);
```

*Description*
When cloning an mpz it is by default initialized to zero. However, you can give a second argument to clone to initialize the new object to that value. The argument can be an int, float another mpz object, or a string containing an
ascii number. You can also give the number in the string in another base by specifying the base as a second argument. Valid bases are 2-36 and 256.

See Also
clone

Gmp.mpz.powm - raise and modulo

object powm(int|string|float|object a, int|string|float|object b);

Description
This function returns (mpz ** a) % b. For example, Mpz(2)->powm(10,42); would return 16 since 2 to the power of 10 is 1024 and 1024 modulo 42 is 16.

Gmp.mpz.sqrt - square root

object sqrt();

Description
This function returns the truncated integer part of the square root of the value of mpz.

Gmp.mpz.probably_prime_p - is this number a prime?

int probably_prime_p();

Description
This function returns 1 if mpz is a prime, and 0 most of the time if it is not.

Gmp.mpz.gcd - greatest common divisor

object gcd(object|int|float|string arg)

Description
This function returns the greatest common divisor for arg and mpz.

Gmp.mpz.cast - cast to other type

object cast( "string" | "int" | "float" );
(string) mpz
(int) mpz
(float) mpz

Description
This function converts an mpz to a string, int or float. This is necessary when you want to view, store or use the result of an mpz calculation.

See Also
cast
**Gmp.mpz.digits** - convert mpz to a string

```plaintext
string digits();
string digits(int|void base);
```

**Description**
This function converts an mpz to a string. If a base is given the number will be represented in that base. Valid bases are 2-36 and 256. The default base is 10.

**See Also**
Gmp.mpz->cast

**Gmp.mpz.size** - how long is a number

```plaintext
string size();
string size(int|void base);
```

**Description**
This function returns how long the mpz would be represented in the specified base. The default base is 2.

**See Also**
Gmp.mpz->digits

### 14.5 Gdbm

Gdbm is short for GNU Data Base Manager. It provides a simple data base similar to a file system. The functionality is similar to a mapping, but the data is located on disk, not in memory. Each gdbm database is one file which contains a key-pair values, both keys and values have to be strings. All keys are always unique, just as with a mapping.

This is the an interface to the gdbm library. This module might or might not be available in your Pike depending on whether the gdbm library was available on your system when Pike was compiled.

**Gdbm.gdbm.create** - open database

```plaintext
int create();
int create(string file);
int create(string file, string mode);
object(Gdbm) Gdbm(); object(Gdbm) Gdbm(string file); object(Gdbm) Gdbm(string file, string mode);
```

**Description**
Without arguments, this function does nothing. With one argument it opens the given file as a gdbm database, if this fails for some reason, an error will be generated. If a second argument is present, it specifies how to open the database using one or more of the follow flags in a string:
14.5. **GDBM**

<table>
<thead>
<tr>
<th>Open Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>r</code></td>
<td>open database for reading</td>
</tr>
<tr>
<td><code>w</code></td>
<td>open database for writing</td>
</tr>
<tr>
<td><code>c</code></td>
<td>create database if it does not exist</td>
</tr>
<tr>
<td><code>t</code></td>
<td>overwrite existing database</td>
</tr>
<tr>
<td><code>f</code></td>
<td>fast mode</td>
</tr>
</tbody>
</table>

The fast mode prevents the database from synchronizing each change in the database immediately. This is dangerous because the database can be left in an unusable state if Pike is terminated abnormally.

The default mode is "rwc".

**Gdbm.gdbm.close** - close database

```cpp
void close();
```

**Description**

This closes the database.

**Gdbm.gdbm.store** - store a value in the database

```cpp
int store(string key, string data);
```

**Description**

Associate the contents of `data` with the key `key`. If the key `key` already exists in the database the data for that key will be replaced. If it does not exist it will be added. An error will be generated if the database was not open for writing.

**Gdbm.gdbm.fetch** - fetch a value from the database

```cpp
string fetch(string key);
```

**Description**

Returns the data associated with the key `key` in the database. If there was no such key in the database, zero is returned.

**Gdbm.gdbm.delete** - delete a value from the database

```cpp
int delete(string key);
```

**Description**

Remove a key from the database. Note that no error will be generated if the key does not exist.

**Gdbm.gdbm.firstkey** - get first key in database

```cpp
string firstkey();
```

**Description**

Returns the first key in the database, this can be any key in the database.
**Gdbm.gdbm.nextkey** - get next key in database

```c
string nextkey(string key);
```

*Description*
This returns the key in database that follows the key `key`. This is of course used to iterate over all keys in the database.

*Example*
```c
/* Write the contents of the database */
for(key=gdbm->firstkey(); k; k=gdbm->nextkey(k))
```

**Gdbm.gdbm.reorganize** - reorganize database

```c
int reorganize();
```

*Description*
Deletions and insertions into the database can cause fragmentation which will make the database bigger. This routine reorganizes the contents to get rid of fragmentation. Note however that this function can take a LOT of time to run.

**Gdbm.gdbm.sync** - synchronize database

```c
void sync();
```

*Description*
When opening the database with the 'f' flag writings to the database can be cached in memory for a long time. Calling sync will write all such caches to disk and not return until everything is stored on the disk.

### 14.6 Getopt

Getopt is a group of function which can be used to find command line options. Command line options come in two flavors: long and short. The short ones consists of a dash followed by a character (\(-t\)), the long ones consist of two dashes followed by a string of text (\(--test\)). The short options can also be combined, which means that you can write `\(-tda\)` instead of `\(-t -d -a\)`. Options can also require arguments, in which case they cannot be combined. To write an option with an argument you write `\(-t argument\)` or `\(-t argument\)` or `\(--test=argument\)`.
14.6. GETOPT

Getopt.find_option - find command line options

mixed find_option(array(string) argv,

Description
This is a generic function to parse command line options of the type '-f', '–foo' or '-foo=bar'. The first argument should be the array of strings that is sent as second argument to your main() function, the second is a string with the short form of your option. The short form must be only one character long. The 'longform' is an alternative and maybe more readable way to give the same option. If you give "foo" as longform your program will accept '–foo' as argument. The envvar argument specifies what environment variable can be used to specify the same option. The envvar option exists to make it easier to customize program usage. The 'def' has two functions: It specifies that the option takes an argument and it tells find_option what to return if the option is not present. If 'def' is given and the option does not have an argument find_option will print an error message and exit the program.

Also, as an extra bonus: shortform, longform and envvar can all be arrays, in which case either of the options in the array will be accepted.

Note
find_option modifies argv.
This function reads options even if they are written after the first non-option on the line.

Example

/* This program tests two different options. One is called -f or
 * --foo and the other is called -b or --bar and can also be given
 * by using the BAR_OPTION environment variable.
 */

int main(int argc, array(string) argv)
{
    if(find_option(argv,"f","foo"))
        werror("The FOO option was given.\n");

    werror("The BAR option got the "
    find_option(argv,"b","bar","BAR_OPTION","default")+
* argument.\n");
}

See Also
Getopt.get_args
Getopt.find_all_options - find command line options

```php
array find_all_options(array(string) argv, array option, int|void posix_me_harder, int|void throw_errors);
```

**Description**
This function does the job of several calls to `find_option`. The main advantage of this is that it allows it to handle the POSIX_ME_HARDER environment variable better. When the either the argument `posix_me_harder` or the environment variable POSIX_ME_HARDER is true, no arguments will be parsed after the first non-option on the command line.

Each element in the array `options` should be an array on the following form:

Only the first three elements has to be included.

The good news is that the output from this function is a lot simpler. `Find_all_options` returns an array where each element is an array on this form:
The name is the identifier from the input and value is the value given to it from the argument, environment variable or default. If no default is given, value will be 1.

Note
find_all_options modifies argv.

Example
First let's take a look at input and output:

```c
> Getopt.find_all_options({"test","-dd"}),
>> ({ ({ "debug", Getopt.NO_ARG, ({"-d","--debug"}), "DEBUG", 1}) });
Result: ({
   ({ "debug", 1 )},
   ({ "debug", 1 })
})
```

This is what it would look like in real code:

```c
import Getopt;

int debug=0;

int main(int argc, array(string) argv
{
   foreach(find_all_options(argv, ({{"debug", MAY_HAVE_ARG, ({"-d","--debug"}), "DEBUG", 1})),
   ({ "version", NO_ARG, ({"-v","--version" }) }) ))),
   mixed option)
{
   switch(option[0])
   {
   case "debug": debug+=option[1]; break;
   case "version":
      write("Test program version 1.0\n");
      exit(1);
   }
}
argv=Getopt.get_args(argv);
```

See Also
Getopt.get_args

Getopt.get_args - get the non-option arguments

array(string) get_args(array(string) argv, void|int throw_errors);

Description
This function returns the remaining command line arguments after you have run find_options to find all the options in the argument list. If there are any options left not handled by find_options an error message will be written and the program will exit. Otherwise a new 'argv' array without the parsed options is returned.

Example

int main(int argc, array(string) argv)
{
    if(find_option(argv,"f","foo"))
        werror("The FOO option was given.\n");

    argv=get_args(argv);
    werror("The arguments are: "+(argv* " ")+".\n");
}

See Also
Getopt.find_option

14.7 Gz

The Gz module contains functions to compress and uncompress strings using the same algorithm as the program gzip. Packing can be done in streaming mode or all at once. Note that this module is only available if the gzip library was available when Pike was compiled. The Gz module consists of two classes; Gz.deflate and Gz.inflate. Gz.deflate is used to pack data and Gz.inflate is used to unpack data. (Think "inflatable boat") Note that these functions use the same algorithm as gzip, they do not use the exact same format however, so you cannot directly unzip gzipped files with these routines. Support for this will be added in the future.

Gz.deflate - string packer

Description
Gz.inflate is a builtin program written in C. It interfaces the packing routines in the libz library.

Note
This program is only available if libz was available and found when Pike was compiled.
See Also
Gz.inflate

**Gz.deflate.create** - initialize packer

```c
void create(int X)
object(Gz.deflate) Gz.deflate(int X)
```

*Description*
This function is called when a new Gz.deflate is created. If given, X should be a number from 0 to 9 indicating the packing / CPU ratio. Zero means no packing, 2-3 is considered 'fast', 6 is default and higher is considered 'slow' but gives better packing.

This function can also be used to re-initialize a Gz.deflate object so it can be re-used.

**Gz.deflate.deflate** - pack data

```c
string deflate(string data, int flush);
```

*Description*
This function performs gzip style compression on a string and returns the packed data. Streaming can be done by calling this function several times and concatenating the returned data. The optional 'flush' argument should be one of the following:

- **Gz.NO_FLUSH**
  Only data that doesn’t fit in the internal buffers is returned.
- **Gz.PARTIAL_FLUSH**
  All input is packed and returned.
- **Gz.SYNC_FLUSH**
  All input is packed and an 'end of data' marker is appended.
- **Gz.FINISH**
  All input is packed and an 'end of data' marker is appended.

Using flushing will degrade packing. Normally NO_FLUSH should be used until the end of the data when FINISH should be used. For interactive data PARTIAL_FLUSH should be used.

See Also
Gz.inflate->inflate

**Gz.inflate** - string unpacker

*Description*
Gz.inflate is a builtin program written in C. It interfaces the packing routines in the libz library.

*Note*
This program is only available if libz was available and found when Pike was compiled.

See Also
Gz.deflate
**Gz.inflate.create** - initialize unpacker

```c
void create()
object(Gz.inflate) Gz.inflate()
```

**Description**

This function is called when a new Gz.inflate is created. It can also be called after the object has been used to re-initialize it.

**Gz.inflate.inflate** - unpack data

```c
string inflate(string data);
```

**Description**

This function performs gzip style decompression. It can inflate a whole file at once or in blocks.

**Example**

```c
import Stdio;
// whole file
write(Gz.inflate())->inflate(stdin->read());

// streaming (blocks)
function inflate=Gz.inflate()->inflate;
while(string s=stdin->read(8192))
```

**See Also**

Gz.deflate->deflate

**Gz.crc32** - calculate checksum

```c
string crc32(string data, void|int start_value);
```

**Description**

This method is usable for calculating checksums, and presents the standard ISO3309 Cyclic Redundancy Check.

**See Also**

Gz

### 14.8 Yp

This module is an interface to the Yellow Pages functions. Yp is also known as NIS (Network Information System) and is most commonly used to distribute
passwords and similar information within a network.

\[ \text{YP.default\_yp\_domain} \] - get the default Yp domain

\text{string default\_yp\_domain();}

\text{Description}
Returns the default yp-domain.

\[ \text{YP.YpDomain} \] - class representing an Yp domain

\text{object(YP.YpDomain) Yp.YpDomain(string|void domain);}  

\text{Description}
This creates a new YpDomain object.

If there is no YP server available for the domain, this function call will block until there is one. If no server appears in about ten minutes or so, an error will be returned. The timeout is not configurable from the C interface to Yp either.

If no domain is given, the default domain will be used. (As returned by Yp.default\_yp\_domain)

\[ \text{YP.YpDomain.bind} \] - bind this object to another domain  

\text{void bind(string|void domain);}  

\text{Description}
Re-bind the object to another (or the same) domain. If no domain is given, the default domain will be used.

\[ \text{YP.YpDomain.match} \] - match a key in a map  

\text{string match(string map, string key);}  

\text{Description}
If 'map' does not exist, an error will be generated. Otherwise the string matching the key will be returned. If there is no such key in the map, 0 will be returned.

\text{arguments} is the map Yp-map to search in. This must be a full map name, for example, you should use passwd\_byname instead of just passwd. \text{key} is the key to search for. The key must match exactly, no pattern matching of any kind is done.

\text{Example}

\text{object dom = Yp.YpDomain();}

\text{write(dom->match("passwd\_byname", "root"));}
**Yp.YpDomain.all** - return the whole map

```cpp
mapping(string:string) all(string map);
```

**Description**
Returns the whole map as a mapping. *map* is the YP-map to search in. This must be the full map name, you have to use `passwd.byname` instead of just `passwd`.

**Yp.YpDomain.map** - call a function for each entry in an Yp map

```cpp
void map(string map, function(string,string,void) over);
```

**Description**
For each entry in 'map', call the function(s) specified by 'over'. Over will get two arguments, the first being the key, and the second the value. *map* is the YP-map to search in. This must be the full map name, as an example, `passwd.byname` instead of just `passwd`.

**Yp.YpDomain.server** - find an Yp server

```cpp
string server(string map);
```

**Description**
Returns the hostname of the server serving the map *map*. *map* is the YP-map to search in. This must be the full map name, as an example, `passwd.byname` instead of just `passwd`.

**Yp.YpDomain.order** - get the 'order' for specified map

```cpp
int order(string map);
```

**Description**
Returns the 'order' number for the map *map*. This is usually a time_t (see the global function `time()`). When the map is changed, this number will change as well. *map* is the YP-map to search in. This must be the full map name, as an example, `passwd.byname` instead of just `passwd`.

**Yp.YpMap** - class representing one Yp map

```cpp
object(Yp.YpMap) YpYpmap(string map, string|void domain);
```

**Description**
This creates a new YpMap object.

If there is no YP server available for the domain, this function call will block until there is one. If no server appears in about ten minutes or so, an error will be returned. The timeout is not configurable from the C-yp interface either. *map* is the YP-map to bind to. This must be the full map name, as an example, `passwd.byname` instead of just `passwd`. If no domain is specified, the default domain will be used. This is usually best.
YP

YP.Map.match - find key in map

string match(string key)
    ; string Yp.YpMap [string key];

Description
Search for the key key. If there is no key in the map, 0 will be returned, otherwise
the string matching the key will be returned. key must match exactly, no pattern
matching of any kind is done.

YP.Map.all - return the whole map as a mapping

mapping(string:string) all();
    (mapping) Yp.YpMap;

Description
Returns the whole map as a mapping.

YP.Map.map - call a function for each entry in the map

void map(function(string,string:void) over);

Description
For each entry in the map, call the function(s) specified by 'over'. The function
will be called like 'void over(string key, string value)'.

YP.Map.server - find what server servers this map

string server();

Description
Returns the hostname of the server serving this map.

YP.Map.order - find the 'order' of this map

int order();

Description
Returns the 'order' number for this map. This is usually a time_t (see the global
function time())

YP.Map.sizeof - return the number of entries in the map

int sizeof(Yp.YpMap);

Description
Returns the number of entries in the map. This is equivalent to sizeof((mapping)map);

YP.Map.indices - return the indices from the map

array(string) indices(Yp.YpMap)
CHAPTER 14. OTHER MODULES

Description
Returns the indices of the map. If indices is called first, values must be called immediately after. If values is called first, it is the other way around.

See Also
Yp.YpMap->values

Yp.YpMap->values - return the values from the map

array(string) values(Yp.Ypmap)

Description
Returns the values of the map. If values is called first, indices must be called immediately after. If indices is called first, it is the other way around.

See Also
Yp.YpMap->indices

Here is an example program using the Yp module, it lists users and their GECOS field from the Yp map "passwd.byname" if your system uses Yp.

import Yp;

void print_entry(string key, string val)
{
    val = (val/".")[4];
    if(strlen(val))
    {
        string q = ".......... ";
        werror(key+q[strlen(key)..]+val+"\n");
    }
}

void main(int argc, array(string) argv)
{
    object (YpMap) o = YpMap("passwd.byname");

    werror("server.... "+ o->server() + "\n"
    "age........ "+ (-o->order()+time()) + "\n"
    "per........ "+ o["per"] + "\n"
    "size...... "+ sizeof(o) + "\n");

    o->map(print_entry); // Print username/GECOS pairs
}

14.9 ADT.Table

ADT.Table is a generic module for manipulating tables. Each table contains one or several columns. Each column is associated with a name, the column
name. Optionally, one can provide a column type. The Table module can do a number of operations on a given table, like computing the sum of a column, grouping, sorting etc.

All column references are case insensitive. A column can be referred to by its position (starting from zero). All operations are non-destructive. That means that a new table object will be returned after, for example, a sort.

An example is available at the end of this section.

**ADT.Table.table.create** - create a table object

```java
void create(array(array) table, array(string) column_names, array(mapping)|void column_types);
```

**Description**
The table class takes two or three arguments:

1.
2.
3.

**Example**

```java
object t = ADT.Table.table( ({ ({ "Blixt", "Gordon" }),
   ({ "Buck", "Rogers" }) )},
   ({ "First name", "Last name" }) );
```

**See Also**

ADT.Table.ASCII.encode

**ADT.Table.table._indices** - gives the column names

```java
array(string) _indices();
```

**Description**
The method returns the column names for the table. The case used when the table was created will be returned.

**ADT.Table.table._values** - gives the table contents

```java
array(array) _values();
```

**Description**
The method returns the contents of a table as a two dimensional array. The format is an array of rows. Each row is an array of columns.
ADT.Table.table.sizeof - gives the number of table rows
int sizeof();

Description
This method returns the number of rows in the table.

ADT.Table.table.reverse - reverses the table rows
object reverse();

Description
This method reverses the rows of the table and returns a new table object.

ADT.Table.table.rename - rename a column
object rename(string|int from, string to);

Description
This method renames the column named from to to and returns a new table object. Note that from can be the column position.

ADT.Table.table.type - fetch or set the type for a column
mapping type(string|int column, mapping|void type);

Description
This method gives the type for the given column. If a second argument is given, the old type will be replaced with type. The column type is only used when the table is displayed. The format is as specified in create.

ADT.Table.table.limit - truncate the table
object limit(int n);

Description
This method truncates the table to the first n rows and returns a new object.

ADT.Table.table.sort - sort the table on one or several columns
object sort(string|int column1, string|int column2, ...);

Description
This method sorts the table in ascendent order on one or several columns and returns a new table object. The left most column is sorted last. Note that the sort is stable.

ADT.Table.table.rsort - sort the table in reversed order on one or several columns
object rsort(string|int column1, string|int column2, ...);
### ADT.Table

#### table.distinct
- Description: Keep unique rows only.
- Usage: 
  ```
  object distinct(string|int column1, string|int column2, ...);
  ```

#### table.sum
- Description: Computes the sum of equal rows.
- Usage: 
  ```
  object sum(string|int column1, string|int column2, ...);
  ```

#### table.group
- Description: Groups the table using functions.
- Usage: 
  ```
  object group(mapping(string|int:funcion) fus, mixed ... arg);
  object group(funcion f, array(string|int)|string|int columns, mixed ... arg);
  ```

#### table.map
- Description: Maps columns over a function.
- Usage: 
  ```
  object map(funcion fu, string|int|array(int|string) columns, mixed ... arg);
  ```

#### table.where
- Description: Filters the table through a function.
- Usage: 
  ```
  object where(array(string|int)|string|int column1, funcion fu, mixed ... arg);
  ```

*Description*

- **Like sort, but the order is descendent.**
- **ADT.Table.table.distinct** - keep unique rows only
- **ADT.Table.Table.table.sum** - computes the sum of equal rows
- **ADT.Table.table.group** - group the table using functions
- **ADT.Table.table.map** - map columns over a function
- **ADT.Table.table.where** - filter the table through a function
**ADT.Table.table.select** - keep only the given columns

element select(string|int column1, string|int column2, ...);

*Description*
This method returns a new table object with the selected columns only.

**ADT.Table.table.remove** - remove columns

element remove(string|int column1, string|int column2, ...);

*Description*
Like select, but the given columns will not be in the resulting table.

**ADT.Table.table.encode** - represent the table as a binary string

string encode();

*Description*
This method returns a binary string representation of the table. It is useful when one wants to store a table, for example in a file.

**ADT.Table.table.decode** - decode an encoded table

string decode(string table_string);

*Description*
This method returns a table object from a binary string representation of a table.

**ADT.Table.table.col** - fetch a column

array col(string|int column);

*Description*
This method returns the contents of a given column as an array.

**ADT.Table.table.[[]]** - fetch a column

array [][](string|int column);

*Description*
Same as col.

**ADT.Table.table.row** - fetch a row

array row(int row_number);

*Description*
This method returns the contents of a given row as an array.
14.9. **ADT.TABLE**

**ADT.Table.table.==='** - compare two tables

```c
int '==='(object table);
```

**Description**

This method compares two tables. They are equal if the contents of the tables and the column names are equal. The column name comparison is case insensitive.

**ADT.Table.table.append_bottom** - concatenate two tables

```c
object append_bottom(object table);
```

**Description**

This method appends two tables. The table given as an argument will be added at the bottom of the current table. Note, the column names must be equal. The column name comparison is case insensitive.

**ADT.Table.table.append_right** - concatenate two tables

```c
object append_right(object table);
```

**Description**

This method appends two tables. The table given as an argument will be added on the right side of the current table. Note that the number of rows in both tables must be equal.

**ADT.Table.ASCII.encode** - produces an ASCII formatted table

```c
string encode(object table, mapping|void options);
```

**Description**

This method returns a table represented in ASCII suitable for human eyes. `options` is an optional mapping. If the keyword "indent" is used with a number, the table will be indented with that number of space characters.

**Example**

```c
array(string) cols = ({ "Fruit", "Provider", "Quantity" });
array(mapping) types = ({ 0, 0, ([ "type":"num" ] ) });
array(array) table = ({ ({ "Avocado", "Frukt AB", 314 }),
                      ({ "Banana", "Banankompaniet", 4276 }),
                      ({ "Orange", "Frukt AB", 81 }),
                      ({ "Banana", "Frukt AB", 1174 }),
                      ({ "Orange", "General Food", 523 }) });

object t = ADT.Table.table(table, cols, types);
write("FRUITS

");
write(ADT.Table.ASCII.encode(t, ([ "indent":4 ])));
Yabu transaction database

Yabu is a all purpose database, used to store data records associated with a unique key. Yabu is very similar to mappings, however, records are stored in files and not in memory. Also, Yabu features tables, which is a way to handle several mapping-like structures in the same database. A characteristic feature of Yabu is that it allows for transactions. A transaction is a sequence of database commands that will be accepted in whole, or not at all.

Some effort has been made to make sure that Yabu is crash safe. This means that the database should survive process kills, core dumps and such – although this is not something that can be absolutely guaranteed. Also, all non-commited and pending transactions will be cancelled in case of a crash.

Yabu uses three types of objects, listed below:

- 
- 
- 

A simple example is illustrated below.

Example

```javascript
// Create a database called "my.database" in write/create mode.
object db = Yabu.db("my.database", "wc");

// Create a table called "fruit".
object table = db["fruit"];```
14.10. YABU TRANSACTION DATABASE

// Store a record called "orange" with the value "yummy".
table["orange"] = "yummy";

// Store a record called "apple" with the value 42.
table["apple"] = 42;

Transactions are slightly more complex, but not much so. See example below.

Example

// Create a database called "my.database"
// in write/create/transaction mode.
object db = Yabu.db("my.database", "wct");

// Create a table called "fruit".
object table = db["fruit"];  

// Create a transaction object for table "fruit".
object transaction = table->transaction();

// Store a record called "orange" with
// the value "yummy". Note that this record
// is only visible through the transaction object.
transaction["orange"] = "yummy";

// Store a record called "apple" with the value 42.
// As with "orange", this record is invisible
// for all objects except this transaction object.
transaction["apple"] = 42;

// Commit the records "orange" and "apple".
// These records are now a part of the database.
transaction->commit();

14.10.1 The database

The db object is the main Yabu database object. It is used to open the database and it can create table objects.

A Yabu database can operate in two basic modes:

•

•
Compressed databases opened without compress mode enabled will be handled correctly in both modes, provided that the Gz module is available. However, new records will no longer be compressed in write mode.

**Yabu.db.create** - Open a Yabu database

```cpp
void create(string directory, string mode);
```

**Description**

Create takes two arguments:

1. 
2. 

**Note**

It is very important not to open the same a database more than once at a time. Otherwise there will be conflicts and most likely strange failures and unpredictable behaviours. Yabu does not check weather a database is already open or not.

**Example**

```cpp
// Open a database in create/write/transaction mode.
object db = Yabu.db("my.database", "cwt");

// Open a database in read mode.
object db = Yabu.db("my.database", "r");

// Open a database in create/write/compress mode.
object db = Yabu.db("my.database", "cwC");
```

**See Also**

Yabu.db->table, Yabu.db->list_tables, Yabu.db->sync and Yabu.db->purge

**Yabu.db.table** - Open a table

```cpp
object(table) table(string table_name);
object(table) '[]'(string table_name);
```

**Description**

This method opens a table with `table_name`. If the table does not exist, it will be created. A table object will be returned.

**See Also**

Yabu.db->list_tables, Yabu.db->sync and Yabu.db->purge
14.10. YABU TRANSACTION DATABASE

**Yabu.db.list_tables** - List all tables

```php
array(string) list_tables();
array(string) _indices();
```

**Description**
This method lists all available tables.

**See Also**
Yabu.db->table, Yabu.db->sync, Yabu.db->purge and Yabu.db->_values

**Yabu.db._values** - Get all tables

```php
array(Yabu.table) _values();
```

**Description**
This method returns all available tables.

**See Also**
Yabu.db->table, Yabu.db->sync, Yabu.db->purge and Yabu.db->_indices

**Yabu.db.sync** - Synchronize database

```php
void sync();
```

**Description**
This method synchronizes the database on disk. Yabu stores some information about the database in memory for performance reasons. Syncing is recommended when one wants the information on disk to be updated with the current information in memory.

**See Also**
Yabu.db->table, Yabu.db->list_tables and Yabu.db->purge

**Yabu.db.purge** - Delete database

```php
void purge();
```

**Description**
This method deletes the whole database and all database files stored on disk.

**See Also**
Yabu.db->table, Yabu.db->list_tables and Yabu.db->sync

14.10.2 Tables

The table object is used to store and retrieve information from a table. Table objects are created by the db class. A table object can create a transaction object.
**Yabu.table.set** - Store a record in a table

```markdown
mixed set(string key, mixed data);
mixed '[]=(string key, mixed data);
```

**Description**

This method stores the contents of `data` as a record with the name `key`. If a record with that name already exists, it will be replaced. Records can only be added to the database in write mode.

**See Also**

Yabu.table->get, Yabu.table->delete, Yabu.table->list_keys and Yabu.table->purge

**Yabu.table.get** - Fetch a record from a table

```markdown
mixed get(string key);
mixed '[](string key);
```

**Description**

This method fetches the data associated with the record name `key`. If a record does not exist, zero is returned.

**See Also**

Yabu.table->set, Yabu.table->delete, Yabu.table->list_keys and Yabu.table->purge

**Yabu.table.list_keys** - List the records in a table

```markdown
array(string) list_keys();
array(string) _indices();
```

**Description**

This method lists all record names in the table.

**See Also**

Yabu.table->set, Yabu.table->get, Yabu.table->delete, Yabu.table->purge and Yabu.table->values

**Yabu.table.values** - Get all the records in a table

```markdown
array(mixed) _values();
```

**Description**

This method returns all record names in the table.

**See Also**

Yabu.table->set, Yabu.table->get, Yabu.table->delete, Yabu.table->purge and Yabu.table->indices

**Yabu.table.delete** - Delete a record in a table

```markdown
void delete(string key);
```
**14.10. YABU TRANSACTION DATABASE**

**Description**
This method deletes the record with the name `key`.

**See Also**
Yabu.table->set, Yabu.table->get, Yabu.table->list_keys and Yabu.table->purge

**Yabu.table.purge** - Delete a table
void purge();

**Description**
This method deletes the whole table and the table files on disk.

**See Also**
Yabu.table->set, Yabu.table->get, Yabu.table->list_keys and Yabu.table->delete

**Yabu.table.transaction** - Begin a transaction
object(transaction) transaction();

**Description**
A transaction is a sequence of table commands that will be accepted in whole, or not at all. If the program for some reason crashes or makes an abrupt exit during a transaction, the transaction is cancelled.

This method returns a transaction object.

**See Also**
Yabu.transaction->commit and Yabu.transaction->rollback

**14.10.3 Transactions**

Transactions make it possible to alter, add or delete several database records and guarantee that all changes, or no changes, will be accepted by the database.

A transaction object is basically a table object with a few restrictions and additions, listed below:

- Rollbacks always succeeds. However, with commit that is not always the case. A commit will fail if:
  -
CHAPTER 14. OTHER MODULES

Description
This method commits the changes made in a transaction. If a record affected by the transaction is altered during the transaction, a conflict will arise and an error is thrown.

See Also
Yabu.table->transaction and Yabu.transaction->rollback

Yabu.transaction.rollback - Rollback a transaction
void rollback();

Description
This method cancels a transaction. All changes made in the transaction are lost.

See Also
Yabu.table->transaction and Yabu.transaction->commit

14.11 MIME

RFC1521*, the Multipurpose Internet Mail Extensions memo, defines a structure which is the base for all messages read and written by modern mail and news programs. It is also partly the base for the HTTP protocol. Just like RFC822*, MIME declares that a message should consist of two entities, the headers and the body. In addition, the following properties are given to these two entities:

The MIME module can extract and analyze these two entities from a stream of bytes. It can also recreate such a stream from these entities. To encapsulate the headers and body entities, the class MIME.Message is used. An object of this class holds all the headers as a mapping from string to string, and it is possible to obtain the body data in either raw or encoded form as a string. Common attributes such as message type and text char set are also extracted into separate variables for easy access.

The Message class does not make any interpretation of the body data, unless the content type is multipart. A multipart message contains several individual messages separated by boundary strings. The create method of the Message class will divide a multipart body on these boundaries, and then create individual Message objects for each part. These objects will be collected in the array body_parts within the original Message object. If any of the new Message objects have a body of type multipart, the process is
of course repeated recursively. The following figure illustrates a multipart message containing three parts, one of which contains plain text, one containing a graphical image, and the third containing raw uninterpreted data:

14.11.1 Global functions

MIME.decode - Remove transfer encoding

string decode(string data, string encoding);

Description
Extract raw data from an encoded string suitable for transport between systems. The encoding can be any of

•
•
•
•
•
•
The encoding string is not case sensitive.

See Also
MIME.encode

MIME.decode_base64 - Decode base64 transfer encoding

string decode_base64(string encoded_data);

Description
This function decodes data encoded using the base64 transfer encoding.

See Also
MIME.encode_base64

MIME.decode_qp - Decode quoted-printable transfer encoding

string decode_qp(string encoded_data);

Description
This function decodes data encoded using the quoted-printable (a.k.a. quoted-unreadable) transfer encoding.

See Also
MIME.encode_qp

MIME.decode_uue - Decode x-uue transfer encoding

string decode_uue(string encoded_data);

Description
This function decodes data encoded using the x-uue transfer encoding. It can also be used to decode generic UUEncoded files.

See Also
MIME.encode_uue

MIME.decode_word - De-scramble RFC1522 encoding

array(string) decode_word(string word);

Description
Extracts the textual content and character set from an encoded word as specified by RFC1522. The result is an array where the first element is the raw text, and the second element the name of the character set. If the input string is not an encoded word, the result is still an array, but the char set element will be set to 0. Note that this function can only be applied to individual encoded words.

Example

> Array.map("=?iso-8859-1?b?S2lscm95?= was =?us-ascii?q?h=65re?="/", MIME.decode_word);
14.11. MIME

```plaintext
Result: ({ /* 3 elements */
    ({ /* 2 elements */
        "Kilroy",
        "iso-8859-1"
    },
    ({ /* 2 elements */
        "was",
        0
    },
    ({ /* 2 elements */
        "here",
        "us-ascii"
    })
})
}
```

See Also
MIME.encode_word

**MIME.encode** - Apply transfer encoding

```plaintext
string encode(string data, string encoding,
```

**Description**
Encode raw data into something suitable for transport to other systems. The encoding can be any of

- 
- 
- 
- 
- 
- 
- 
- 
- 
- 

The encoding string is not case sensitive. For the x-uue encoding, an optional filename string may be supplied. If a nonzero value is passed as no_linebreaks, the result string will not contain any linebreaks (base64 and quoted-printable only).

See Also
MIME.decode
MIME.encode_base64 - Encode string using base64 transfer encoding

string encode_base64(string data, void|int no_linebreaks);

Description
This function encodes data using the base64 transfer encoding. If a nonzero value is passed as no_linebreaks, the result string will not contain any linebreaks.

See Also
MIME.decode_base64

MIME.encode_qp - Encode string using quoted-printable transfer encoding

string encode_qp(string data, void|int no_linebreaks);

Description
This function encodes data using the quoted-printable (a.k.a. quoted-unreadable) transfer encoding. If a nonzero value is passed as no_linebreaks, the result string will not contain any linebreaks.

Note
Please do not use this function. QP is evil, and there’s no excuse for using it.

See Also
MIME.decode_qp

MIME.encode_uue - Encode string using x-uue transfer encoding

string encode_uue(string encoded_data, void|string filename);

Description
This function encodes data using the x-uue transfer encoding. The optional argument filename specifies an advisory filename to include in the encoded data, for extraction purposes. This function can also be used to produce generic UUEncoded files.

See Also
MIME.decode_uue

MIME.encode_word - Encode word according to RFC1522

string encode_word(array(string) word, string encoding);

Description
Create an encoded word as specified in RFC1522 from an array containing a raw text string and a char set name. The text will be transfer encoded according to the encoding argument, which can be either "base64" or "quoted-printable" (or either "b" or "q" for short). If either the second element of the array (the char set name), or the encoding argument is 0, the raw text is returned as is.

Example
> MIME.encode_word( ( { "Quetzalcoatl", "iso-8859-1" } ), "base64" );
Result: =?iso-8859-17b?UXVldHphbGhWvYXRs?=
> MIME.encode_word( ( { "Foo", 0 } ), "base64" );
Result: Foo

See Also
MIME.decode_word

MIME.generate_boundary - Create a suitable boundary string for multipart

string generate_boundary();

Description
This function will create a string that can be used as a separator string for multipart messages. The generated string is guaranteed not to appear in base64, quoted-printable, or x-ue encoded data. It is also unlikely to appear in normal text. This function is used by the cast method of the Message class if no boundary string is specified.

MIME.guess_subtype - Provide a reasonable default for the subtype field

string guess_subtype(string type);

Description
Some pre-RFC1521 mailers provide only a type and no subtype in the Content-Type header field. This function can be used to obtain a reasonable default subtype given the type of a message. (This is done automatically by the MIME.Message class.) Currently, the function uses the following guesses:

<table>
<thead>
<tr>
<th>type</th>
<th>subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>plain</td>
</tr>
<tr>
<td>message</td>
<td>rfc822</td>
</tr>
<tr>
<td>multipart</td>
<td>mixed</td>
</tr>
</tbody>
</table>

MIME.parse_headers - Separate a bytestream into headers and body

array(mapping(string:string)|string) parse_headers(string message);

Description
This is a low level function that will separate the headers from the body of an encoded message. It will also translate the headers into a mapping. It will however not try to analyze the meaning of any particular header. This also means that the body is returned as is, with any transfer-encoding intact. It is possible to call this function with just the header part of a message, in which case an empty body will be returned.
The result is returned in the form of an array containing two elements. The first element is a mapping containing the headers found. The second element is a string containing the body.

**MIME.quote** - Create an RFC822 header field from lexical elements

```php
string quote(array(string|int) lexical_elements);
```

**Description**
This function is the inverse of the tokenize function. A header field value is constructed from a sequence of lexical elements. Characters (ints) are taken to be special-characters, whereas strings are encoded as atoms or quoted-strings, depending on whether they contain any special characters.

**Example**

```bash
> MIME.quote( ({ "attachment", ';', "filename", '=', "/usr/dict/words" }) );
Result: attachment;filename="/usr/dict/words"
```

**Note**
There is no way to construct a domain-literal using this function. Neither can it be used to produce comments.

**See Also**
MIME.tokenize

**MIME.reconstruct_partial** - Join a fragmented message to its original form

```php
int|object reconstruct_partial(array(object) collection);
```

**Description**
This function will attempt to reassemble a fragmented message from its parts. The array collection should contain MIME.Message objects forming a complete set of parts for a single fragmented message. The order of the messages in this array is not important, but every part must exist at least once.

Should the function succeed in reconstructing the original message, a new MIME.Message object is returned. Note that this message may in turn be a part of another, larger, fragmented message. If the function fails to reconstruct an original message, it returns an integer indicating the reason for its failure:

- 
- 
- 

**See Also**
MIME.Message->is_partial
**MIME.tokenize** - Separate an RFC822 header field into lexical elements

```plaintext
array(string|int) tokenize(string header);
```

**Description**
A structured header field, as specified by RFC822, is constructed from a sequence of lexical elements. These are:

- ...
- ...
- ...
- ...
- ...
- ...

This function will analyze a string containing the header value, and produce an array containing the lexical elements. Individual special characters will be returned as characters (i.e. `ints`). Quoted-strings, domain-literals and atoms will be decoded and returned as strings. Comments are not returned in the array at all.

**Example**

```bash
> MIME.tokenize("multipart/mixed; boundary="foo/bar" (Kilroy was here))
```

Result: `({ /* 7 elements */
  "multipart",
  47,
  "mixed",
  59,
  "boundary",
  61,
  "foo/bar"
})`

**Note**
As domain-literals are returned as strings, there is no way to tell the domain-literal `[127.0.0.1]` from the quoted-string `"[127.0.0.1]"`. Hopefully this won’t cause any problems. Domain-literals are used seldom, if at all, anyway...

The set of special-characters is the one specified in RFC1521 (i.e. "<", ">", ":", ",", ":", ":", "\", "/", ":", "="), and not the one specified in RFC822.

**See Also**
MIME.quote

**14.11.2 The MIME.Message class**
**MIME.Message** - The MIME.Message class

This class is used to hold a decoded MIME message.

**Public fields**

**MIME.Message.body_parts** - Multipart sub-messages

array(object) msg->body_parts;

*Description*
If the message is of type `multipart`, this is an array containing one Message object for each part of the message. If the message is not a multipart, this field is 0.

*See Also*
MIME.Message->type and MIME.Message->boundary

**MIME.Message.boundary** - Boundary string for multipart messages

string msg->boundary;

*Description*
For multipart messages, this **Content-Type** parameter gives a delimiter string for separating the individual messages. As multiparts are handled internally by the module, you should not need to access this field.

*See Also*
MIME.Message->setboundary

**MIME.Message.charset** - Character encoding for text bodies

string msg->charset;

*Description*
One of the possible parameters of the **Content-Type** header is the charset attribute. It determines the character encoding used in bodies of type text. If there is no **Content-Type** header, the value of this field is "us-ascii".

*See Also*
MIME.Message->type

**MIME.Message.disposition** - Multipart subpart disposition

string msg->disposition;

*Description*
The first part of the **Content-Disposition** header, hinting on how this part of a multipart message should be presented in an interactive application. If there is no **Content-Disposition** header, this field is 0.
**MIME.Message.disp_params** - Content-Disposition parameters

```
mapping(string:string) msg->disp_params;
```

*Description*

A mapping containing all the additional parameters to the `Content-Disposition` header.

*See Also*

`MIME.Message->setdisp_param` and `MIME.Message->get_filename`

**MIME.Message.headers** - All header fields of the message

```
mapping(string:string) msg->headers;
```

*Description*

This mapping contains all the headers of the message. The key is the header name (in lower case) and the value is the header value. Although the mapping contains all headers, some particular headers get special treatment by the module and should **not** be accessed through this mapping. These fields are currently:

- 
- 
- 
- 

The contents of these fields can be accessed and/or modified through a set of variables and methods available for this purpose.

*See Also*


**MIME.Message.params** - Content-Type parameters

```
mapping(string:string) msg->params;
```

*Description*

A mapping containing all the additional parameters to the `Content-Type` header. Some of these parameters have fields of their own, which should be accessed instead of this mapping wherever applicable.

*See Also*

`MIME.Message->charset`, `MIME.Message->boundary` and `MIME.Message->setparam`
MIME.Message.subtype - The subtype attribute of the Content-Type header

```cpp
string msg->subtype;
```

Description
The Content-Type header contains a type, a subtype, and optionally some parameters. This field contains the subtype attribute extracted from the header. If there is no Content-Type header, the value of this field is "plain".

See Also
MIME.Message->type and MIME.Message->params

MIME.Message.transfer_encoding - Body encoding method

```cpp
string msg->transfer_encoding;
```

Description
The contents of the Content-Transfer-Encoding header. If no Content-Transfer-Encoding header is given, this field is 0. Transfer encoding and decoding is done transparently by the module, so this field should be interesting only to applications wishing to do auto conversion of certain transfer encodings.

See Also
MIME.Message->setencoding

MIME.Message.type - The type attribute of the Content-Type header

```cpp
string msg->type;
```

Description
The Content-Type header contains a type, a subtype, and optionally some parameters. This field contains the type attribute extracted from the header. If there is no Content-Type header, the value of this field is "text".

See Also
MIME.Message->subtype and MIME.Message->params

Public methods

MIME.Message.cast - Encode message into byte stream

```cpp
string (string )MIME.Message;
```

Description
Casting the message object to a string will yield a byte stream suitable for transmitting the message over protocols such as ESMTP and NNTP. The body will be encoded using the current transfer encoding, and subparts of a multipart will be collected recursively. If the message is a multipart and no boundary string has been set, one is generated using generate_boundary.

Example
14.11. MIME

> object msg = MIME.Message("Hello, world!",
  ([ "MIME-Version" : "1.0",
    "Content-Type" : "text/plain",
    "Content-Transfer-Encoding" : "base64" ]));
Result: object
> (string )msg;
Result: Content-Type: text/plain
Content-Length: 20
Content-Transfer-Encoding: base64
MIME-Version: 1.0

SGVsbG8sIHdvcmxkIQ==

See Also
MIME.Message->create

MIME.Message.create - Create a Message object

object MIME.Message(void | string message,

Description
There are several ways to call the constructor of the Message class;

•

•

•

Example
> object msg = MIME.Message("Hello, world!", ([ "MIME-Version" : "1.0",
"Content-Type" : "text/plain; charset=iso-8859-1" ])); Result: object
> msg->charset; Result: iso-8859-1

See Also
MIME.Message->cast

MIME.Message.getdata - Obtain raw body data

string getdata();

Description
This method returns the raw data of the message body entity. The type and subtype attributes indicate how this data should be interpreted.

See Also
MIME.Message->getencoded
MIME.Message.getencoded - Obtain encoded body data

```c
string getencoded();
```

**Description**
This method returns the data of the message body entity, encoded using the current transfer encoding. You should never have to call this function.

**See Also**
MIME.Message->getdata

MIME.Message.get_filename - Get supplied filename for body data

```c
string get_filename();
```

**Description**
This method tries to find a suitable filename should you want to save the body data to disk. It will examine the `filename` attribute of the `Content-Disposition` header, and failing that the `name` attribute of the `Content-Type` header. If neither attribute is set, the method returns 0.

**Note**
An interactive application should always query the user for the actual filename to use. This method may provide a reasonable default though.

MIME.Message.is_partial - Identify `message/partial` message

```c
array(string|int) is_partial();
```

**Description**
If this message is a part of a fragmented message (i.e. has a Content-Type of `message/partial`), an array with three elements is returned. The first element is an identifier string. This string should be used to group this message with the other fragments of the message (which will have the same id string). The second element is the sequence number of this fragment. The first part will have number 1, the next number 2 etc. The third element of the array is either the total number of fragments that the original message has been split into, or 0 of this information was not available. If this method is called in a message that is not a part of a fragmented message, it will return 0.

**See Also**
MIME.reconstructpartial

MIME.Message.setboundary - Set boundary parameter

```c
void setboundary(string boundary);
```

**Description**
Sets the `boundary` parameter of the `Content-Type` header. This is equivalent of calling `msg->setparam("boundary", boundary)`.

**See Also**
MIME.Message->setparam
**MIME.Message.setcharset** - Set charset parameter

```c
void setcharset(string charset);
```

**Description**
Sets the charset parameter of the Content-Type header. This is equivalent of calling `msg->setparam("charset", charset)`.

**See Also**
MIME.Message->setparam

**MIME.Message.setdata** - Replace body data

```c
void setdata(string data);
```

**Description**
Replaces the body entity of the data with a new piece of raw data. The new data should comply to the format indicated by the type and subtype attributes. Do not use this method unless you know what you are doing.

**See Also**
MIME.Message->getdata

**MIME.Message.setdisp_param** - Set Content-Disposition parameters

```c
void setdisp_param(string param, string value);
```

**Description**
Set or modify the named parameter of the Content-Disposition header. A common parameters is e.g. filename. It is not allowed to modify the Content-Disposition header directly, please use this function instead.

**See Also**
MIME.Message->setparam and MIME.Message->get_filename

**MIME.Message.setencoding** - Set transfer encoding for message body

```c
void setencoding(string encoding);
```

**Description**
Select a new transfer encoding for this message. The Content-Transfer-Encoding header will be modified accordingly, and subsequent calls to getencoded will produce data encoded using the new encoding. See encode for a list of valid encodings.

**See Also**
MIME.Message->getencoded

**MIME.Message.setparam** - Set Content-Type parameters

```c
void setparam(string param, string value);
```
**Description**
Set or modify the named parameter of the **Content-Type** header. Common parameters include **charset** for text messages, and **boundary** for multipart messages. It is not allowed to modify the **Content-Type** header directly, please use this function instead.

**See Also**
MIME.Message->setcharset, MIME.Message->setboundary and MIME.Message->setdisp_param

### 14.12 Simulate

This module is used to achieve better compatibility with older versions of Pike. It can also be used for convenience, but I would advice against it since some functions defined here are much slower than using similar functions in other modules. The purpose of this section in the manual is to make it easier for the reader to understand code that uses the Simulate module, not to encourage the use of the Simulate module.

Simulate inherits the Array, Stdio, String and Process modules, so importing the Simulate module also imports all identifiers from these modules. In addition, these functions are available:

**Simulate.member_array** - find first occurrence of a value in an array

```c
int member_array(mixed item, array arr);
```

**Description**
Returns the index of the first occurrence of item in array arr. If not found, then -1 is returned. This is the same as `search(arr, item)`.

**Simulate.previous_object** - return the calling object

```c
object previous_object();
```

**Description**
Returns an object pointer to the object that called current function, if any.

**See Also**
backtrace

**Simulate.this_function** - return a function pointer to the current function

```c
function this_function();
```

**Description**
Returns a function pointer to the current function, useful for making recursive lambda-functions.

**See Also**
backtrace
**Simulate.get_function** - fetch a function from an object

```javascript
function get_function(object o, string name);
```

*Description*
Defined as: return o[name];

**Simulate.map_regexp** - filter an array through a regexp

```javascript
array(string) map_regexp(array(string) arr, string reg);
```

*Description*
Returns those strings in arr that matches the regexp in reg.

*See Also*
Regexp

**Simulate.PI** - pi

```javascript
PI;
```

*Description*
This is not a function, it is a constant roughly equal to the mathematical constant π.

**Simulate.all_efuns** - return all 'efuns'

```javascript
mapping all_efuns();
```

*Description*
This function is the same as all_constants.

*See Also*
all_constants

**Simulate.explode** - explode a string on a delimiter

```javascript
string explode(string s, string delimiter);
```

*Description*
This function is really the same as the division operator. It simply divides the string s into an array by splitting s at every occurrence of delimiter.

*See Also*
Simulate.implode

**Simulate.filter_array** - filter an array through a function

```javascript
array filter_array(array arr, function fun, mixed ... args);
array filter_array(array(object) arr, string fun, mixed ... args);
array filter_array(array(function) arr, -1, mixed ... args);
```

*Description*
Filter array is the same function as Array.filter.
**See Also**

Array.filter

**Simulate.implode** - implode an array of strings

```plaintext
string implode(array(string) a, string delimiter);
```

**Description**

This function is the inverse of explode. It concatenates all the strings in `a` with a delimiter in between each.

This function is the same as multiplication.

**Example**

```plaintext
> implode( ({ "foo","bar","gazonk"}), "-" );
Result:  foo-bar-gazonk
> ({ "a","b","c" })*" and ";
Result:  a and b and c
```

**See Also**

Simulate.explode

**Simulate.m_indices** - return all indices from a mapping

```plaintext
array m_indices(mapping m);
```

**Description**

This function is equal to indices.

**See Also**

indices

**Simulate.m_sizeof** - return the size of a mapping

```plaintext
int m_sizeof(mapping m);
```

**Description**

This function is equal to sizeof.

**See Also**

sizeof

**Simulate.m_values** - return all values from a mapping

```plaintext
array m_values(mapping m);
```

**Description**

This function is equal to values.

**See Also**

values
**Simulate.map_array** - map an array over a function

```plaintext
array map_array(array arr, function fun, mixed ... args);
array map_array(array(object) arr, string fun, mixed ... args);
array map_array(array(function) arr, -1, mixed ... arg);
```

**Description**
This function is the same as Array.map.

**See Also**
Array.map

**Simulate.strstr** - find a string inside a string

```plaintext
int strstr(string str1, string str2);
```

**Description**
Returns the position of str2 in str1, if str2 can’t be found in str1-1 is returned.

**See Also**
sscanf, Simulate.explode and search

**Simulate.sum** - add values together

```plaintext
int sum(int ... i);
float sum(float ... f);
string sum(string|float|int ... p);
array sum(array ... a);
mapping sum(mapping ... m);
list sum(multiset ... l);
```

**Description**
This function does exactly the same thing as adding all the arguments together with +. It’s just here so you can get a function pointer to the summation operator.

**Simulate.add_efun** - add an efun or constant

```plaintext
void add_efun(string func_name, mixed function)
void add_efun(string func_name)
```

**Description**
This function is the same as add_constant.

**See Also**
Simulate.add_constant

**Simulate.l_sizeof** - return the size of a multiset

```plaintext
int l_sizeof(multiset m);
```

**Description**
This function is equal to sizeof.
CHAPTER 14. OTHER MODULES

See Also

sizeof

Simulate.listp - is the argument a list? (multiset)

int listp(mixed l);

Description
This function is the same as multisetp.

See Also
Simulate.multisetp

Simulate.mklist - make a multiset

multiset mklist(array a);

Description
This function creates a multiset from an array.

Example

> mklist( [{1,2,3}] ); Result: ( < /* 3 elements */ 1, 2, 3 > )

See Also
aggregate_multiset

Simulate.aggregate_list - aggregate a multiset

multiset aggregate_list(mixed ... args);

Description
This function is exactly the same as aggregate_multiset.

See Also
aggregate_multiset

Simulate.query_host_name - return the name of the host we are running on

string query_host_name();

Description
This function returns the name of the machine the interpreter is running on. This is the same thing that the command 'hostname' prints.

Description
This module enables access to the Mysql database from within Pike.

Mysql is available from www.tcx.se.

Note
$id: mysql.c,v 1.33 2000/01/03 19:02:00 grubba Exp $

See Also
Mysql.mysql, Mysql.result and Sql.sql
14.13 Mysql.mysql

**Mysql.mysql**

*Description*

Mysql.mysql is a pre-compiled Pike program. It enables access to the Mysql database from within Pike. Mysql.mysql is a part of the Mysql module.

Mysql is available from www.tcx.se.

*See Also*

Mysql.result and Sql.sql

**Mysql.mysql.affected_rows**

`int affected_rows()`

*Description*

Returns the number of rows affected by the last query.

**Mysql.mysql.big_query**

`object(Mysql.mysql_result) big_query(string q)`

*Description*

Make an SQL query

*Arguments*

*See Also*

mysql_result

**Mysql.mysql.binary_data**

`int binary_data()`

*Description*

Inform if this version of mysql supports binary data

This function returns non-zero if binary data can be reliably stored and retrieved with this version of the mysql-module.

Usually, there is no problem storing binary data in mysql-tables, but data containing ‘0’ (NUL) couldn’t be fetched with old versions (prior to 3.20.5) of the mysql-library.

**Mysql.mysql.create**

`void create()`

`void create(string host)`

`void create(string host, string database)`

`void create(string host, string database, string user)`

`void create(string host, string database, string user, string password)`
Description
Connects to a Mysql database.

To access the Mysql database, you must first connect to it. This is done with the function mysql().

If you give no argument, or give "" as hostname it will connect with a UNIX-domain socket, which is a big performance gain.

**Mysql.mysql.create_db**

```c
void create_db(string database)
```

*Description*
Create a new database

*Arguments*

*See Also*
select_db and drop_db

**Mysql.mysql.drop_db**

```c
void drop_db(string database)
```

*Description*
Drop a database

*Arguments*

*See Also*
create_db and select_db

**Mysql.mysql.error**

```c
string error()
```

*Description*
Returns a string describing the last error from the Mysql-server.

**Mysql.mysql.host_info**

```c
string host_info()
```

*Description*
Give information about the Mysql-server connection

This function returns a string describing the connection to the Mysql-server.

*See Also*
statistics, server_info and protocol_info
**Mysql.mysql.insert_id**

```c
int insert_id()
```

**Description**

Returns the id of the last INSERT query into a table with an AUTO_INCREMENT field.

**Mysql.mysql.list_dbs**

```c
object(Mysql.mysql_result) list_dbs()
object(Mysql.mysql_result) list_dbs(string wild)
```

**Description**

List databases

Returns a table containing the names of all databases in the Mysql-server. If an argument is specified, only those matching wild are returned.

**See Also**

list_tables, list_fields, list_processes and Mysql.mysql_result

**Mysql.mysql.list_fields**

```c
array(int|mapping(string:mixed)) list_fields(string table)
array(int|mapping(string:mixed)) list_fields(string table, string wild)
```

**Description**

List all fields.

Returns an array of mappings with information about the fields in the specified table.

The mappings contain the following entries:

- **"name"**: string The name of the field.
- **"table"**: string The name of the table.
- **"default"**: string The default value for the field.
- **"type"**: string The type of the field.
- **"length"**: int The length of the field.
- **"max_length"**: int The length of the longest element in this field.
- **"flags"**: multiset(string) Some flags.
- **"decimals"**: int The number of decimal places.

The type of the field can be any of: "decimal", "char", "short", "long", "float", "double", "null", "time", "longlong", "int24", "tiny_blob", "medium_blob", "long_blob", "var_string", "string" or "unknown".

The flags multiset can contain any of "primary_key": This field is part of the primary key for this table. "not_null": This field may not be NULL. "blob": This field is a blob field.
Arguments

Note
Michael Widenius recomends usage of the following query instead:

```
show fields in 'table' like "wild"
```

See Also
list_dbs, list_tables, list_processes and fetch_fields

Mysql.mysql.list_processes

object(Mysql.mysql_result) list_processes()

Description
List all processes in the Mysql-server
Returns a table containing the names of all processes in the Mysql-server.

See Also
list_dbs, list_tables, list_fields and Mysql.mysql_result

Mysql.mysql.list_tables

object(Mysql.mysql_result) list_tables()
object(Mysql.mysql_result) list_tables(string wild)

Description
List tables in the current database

Arguments

See Also
list_dbs, list_fields, list_processes and Mysql.mysql_result

Mysql.mysql.protocol_info

int protocol_info()

Description
Give the Mysql protocol version
This function returns the version number of the protocol the Mysql-server uses.

See Also
statistics, server_info and host_info

Mysql.mysql.reload

void reload()
Description
Reload security tables
This function causes the Mysql-server to reload its access tables.

See Also
shutdown

Mysql.mysql.select_db
void select_db(string database)

Description
The Mysql-server can hold several databases. You select which one you want to
access with this function.

See Also
create, create_db and drop_db

Mysql.mysql.server_info
string server_info()

Description
Give the version number of the Mysql-server
This function returns the version number of the Mysql-server.

See Also
statistics, host_info and protocol_info

Mysql.mysql.shutdown
void shutdown()

Description
Shutdown the Mysql-server
This function shuts down a running Mysql-server.

reload

Mysql.mysql.statistics
string statistics()

Description
Some Mysql-server statistics
This function returns some server statistics.

Example:

int main()
{
    write(mysql()-&gt;statistics());
return(0);
}

See Also
server_info, host_info and protocol_info

14.14 The Pike Crypto Toolkit

14.14.1 Introduction

The Crypto module provides an object-oriented framework for encryption and related functionality. More specifically, its objects can be classified as follows:

14.14.2 Block ciphers

The block ciphers included in the current version are DES, IDEA and CAST128 (note that IDEA is patented, see Ascom Tech* for details). All block ciphers have a common set of methods.

string o->crypt_block(string blocks);
14.14. THE PIKE CRYPTO TOOLKIT

Description
Encrypts or decrypts an even number of block, using the current key. If more than one block is given, they are encrypted/decrypted independently, i.e. in Electronic Code Book (ECB) mode.

query_block_size
int o->query_block_size();

Description
Returns the block size of the cipher, in octets. A typical block size is 8 octets. A string passed to crypt_block() must have a length that is a multiple of the block size.

query_key_length
int o->query_key_length();

Description
Returns the key size of the cipher, in octets. Note that some block ciphers, e.g. CAST128, have a variable key size. In this case, query_key_length returns the recommended key size, although keys of other lengths are accepted by the set_encrypt_key and set_decrypt_key methods. For DES, the real key length is seven octets (56 bits), but the DES standard mandates the use of parity bits. The query_key_length method lies about DES, and says that the key size is eight octets (64 bits). See also des_parity.

set_encrypt_key
object o->set_encrypt_key(string key);

Description
Installs a key, and configures the object for doing encryption. For convenience, this method returns the object itself.

set_decrypt_key
object o->set_decrypt_key(string key);

Description
Installs a key, and configures the object for doing decryption. For convenience, this method returns the object itself. The classes are Crypto.des

Crypto.des

Crypto.idea

Crypto.idea and

Crypto.cast

Crypto.cast. To encrypt the block "Pike des" using the DES-key '0123456789abcdef' (in hex), use
Although most applications will not use the Crypto.des class directly.

### 14.14.3 Stream Ciphers

Currently the only stream cipher in the toolkit is the RC4 cipher (also known as "arcfour").

**Crypto.rc4**

**string Crypto.rc4->crypt(string data);**

*Description*
Encrypts or decrypts a string of data.

**Crypto.rc4.set_encrypt_key**

**object Crypto.rc4->set_encrypt_key(string key);**

*Description*
Installs a key, and configures the object for doing encryption. For convenience, this method returns the object itself.

**Crypto.rc4.set_decrypt_key**

**object Crypto.rc4->set_decrypt_key(string key);**

*Description*
Installs a key, and configures the object for doing decryption. For convenience, this method returns the object itself. Because of the way RC4 works, set_encrypt_key and set_decrypt_key are actually equivalent.

### 14.14.4 Hash Functions

Cryptographic hash functions are essential for many cryptographic applications, and are also useful in other contexts. The Toolkit includes the two most common, *MD5* and *SHA1*. They have the same methods.

**update**

**object o->update(string data);**

*Description*
Processes some more data. For convenience, this method returns the object itself.
14.14. THE PIKE CRYPTO TOOLKIT

**digest**

```c
string o->digest();
```

**Description**

Returns the hash value, or *message digest*, corresponding to all the data that was previously passed to the update method. Also resets the hash object, so that it can be used to process a new message.

**query_digest_size**

```c
int o->query_digest_size();
```

**Description**

Returns the size, in octets, of the digests produced by this hash function. To get the md5 hash of a string `s`, you would use

```c
Crypto.md5()->update(s)->digest()
```

### 14.14.5 Public key algorithms

The only public key algorithm currently in the toolkit is RSA. As the algorithm uses arithmetic on huge numbers, you must also have the GMP library and the corresponding Pike module installed in order to use RSA.

**Crypto.rsa**

**Crypto.rsa.set_public_key**

```c
object rsa->set_public_key(object(Gmp.mpz) modulo, object(Gmp.mpz) e)
```

**Description**

Sets the modulo and the public exponent. For convenience, returns the object itself.

**Crypto.rsa.set_private_key**

```c
object rsa->set_private_key(object(Gmp.mpz) d)
```

**Description**

Sets the private exponent. For convenience, returns the object itself.

**Crypto.rsa.generate_key**

```c
object rsa->generate_key(int bits, function|void random)
```

**Description**

Generates a new rsa key pair, with a modulo of the given bitsize. `random` should be a function that takes one integer argument `n` and returns a string of `n` random
octets. The default function is \texttt{Crypto.randomness.really\_random()}. For convenience, this method returns the object itself.

\texttt{Crypto.rsa.query\_blocksize}

\begin{verbatim}
int rsa->query_block_size()
\end{verbatim}

\textit{Description}

Returns the length of the largest string that can be encrypted in one RSA-operation using the current key.

\texttt{Crypto.rsa.encrypt}

\begin{verbatim}
string rsa->encrypt(string message, function|void random)
\end{verbatim}

\textit{Description}

Encrypts \textit{message} using PKCS#1-style RSA encryption. The function \textit{random} is used to generate random message padding. Padding does not require a strong random number generator. The default \textit{random} function is derived from Pike’s built-in pseudorandom generator \texttt{predef::random}.

\texttt{Crypto.rsa.decrypt}

\begin{verbatim}
string rsa->decrypt(string gibberish)
\end{verbatim}

\textit{Description}

Decrypts a PKCS#1-style RSA-encrypted message. This operation requires knowledge of the private key. Decryption may fail if the input is not a properly encrypted message for this key. In this case, the method returns zero. The PKCS#1 padding method used is vulnerable to a chosen-ciphertext attack discovered by Daniel Bleichenbacher. There are several methods for signature creation and verification. I don’t quite like the interface, so it may very well change in some future version of the Toolkit.

\texttt{Crypto.rsa.sign}

\begin{verbatim}
object(Gmp.mpz) rsa->sign(string message, program hash)
\end{verbatim}

\textit{Description}

Creates a PKCS#1-style signature. This operation requires knowledge of the private key. \textit{hash} should be a hash algorithm with an -\_identifier method which returns a DER-encoded ASN.1 Object Identifier for the hash algorithm. Currently, this is supported only by Crypto.md5. The function returns the signature as a bignum; applications can use

\begin{verbatim}
Standards.ASN1.Types.asn1\_bit\_string(rsa->sign(...))->get\_der()
\end{verbatim}

to convert it a DER-encoded ASN.1 bitstring.

\texttt{Crypto.rsa.verify}

\begin{verbatim}
int verify(string message, program hash, object(Gmp.mpz) signature)
\end{verbatim}
Description
Verifies a PKCS#1-style RSA signature. Returns 1 if the signature is valid, 0 if not.

**Crypto.rsa.sha_sign**

```c
string rsa->sha_sign(string message)
```

Description
Creates an RSA signature using a simpler but non-standard convention.

**Crypto.rsa.sha_verify**

```c
int sha_verify(string message, string signature)
```

Description
Verifies signatures created by sha_sign. Returns 1 if the signature is valid, 0 if not.

### 14.14.6 Combining block cryptos

### 14.15 Locale.Gettext

Description
This module enables access to localization functions from within Pike.

Note
$Id: gettext.c,v 1.3 2000/02/29 20:41:13 neotron Exp $

**Locale.Gettext.bindtextdomain**

```c
string bindtextdomain(string|void domaniname, string|void dirname)
```

Description
The bindtextdomain() function binds the path predicate for a message domain domaniname to the value contained in dirname. If domaniname is a non-empty string and has not been bound previously, bindtextdomain() binds domaniname with dirname.

If domaniname is a non-empty string and has been bound previously, bindtextdomain() replaces the old binding with dirname. The dirname argument can be an absolute or relative pathname being resolved when gettext(), dgettext(), or dgettext() are called. If domaniname is null pointer or an empty string, bindtextdomain() returns 0. User defined domain names cannot begin with the string SYS_. Domain names beginning with this string are reserved for system use.

The return value from bindtextdomain() is a string containing dirname or the directory binding associated with domaniname if dirname is void. If no binding is found, the default locale path is returned. If domaniname is void or an empty string, bindtextdomain() takes no action and returns a 0.
Arguments

See Also
textdomain, gettext, dgettext, dcgettext, setlocale and localeconv

Locale.Gettext.dcgettext

string dcgettext(string domain, string msg, int category)

Description
Return a translated version of msg within the context of the specified domain and current locale for the specified category. Calling dcgettext with category Locale.Gettext.LC_Messages gives the same result as dgettext.

If there is not translation available, msg is returned.

Arguments

See Also
bindtextdomain, textdomain, gettext, dgettext, setlocale and localeconv

Locale.Gettext.dgettext

string dgettext(string domain, string msg)

Description
Return a translated version of msg within the context of the specified domain and current locale.

If there is not translation available, msg is returned.

Arguments

See Also
bindtextdomain, textdomain, gettext, dgettext, setlocale and localeconv

Locale.Gettext.gettext

string gettext(string msg)

Description
Return a translated version of msg within the context of the current domain and locale.

If there is not translation available, msg is returned.

Arguments
See Also
bindtextdomain, textdomain, dgettext, dcgettext, setlocale and localeconv

Locale.Gettext.localeconv

mapping localeconv()

Description
The localeconv() function returns a mapping with settings for the current locale. This mapping contains all values associated with the locale categories LC_NUMERIC and LC_MONETARY.

string decimal_point: The decimal-point character used to format non-monetary quantities.

string thousands_sep: The character used to separate groups of digits to the left of the decimal-point character in formatted non-monetary quantities.

string int_curr_symbol: The international currency symbol applicable to the current locale, left-justified within a four-character space-padded field. The character sequences should match with those specified in ISO 4217 Codes for the Representation of Currency and Funds.

string currency_symbol: The local currency symbol applicable to the current locale.

string mon_decimal_point: The decimal point used to format monetary quantities.

string mon_thousands_sep: The separator for groups of digits to the left of the decimal point in formatted monetary quantities.

string positive_sign: The string used to indicate a non-negative-valued formatted monetary quantity.

string negative_sign: The string used to indicate a negative-valued formatted monetary quantity.

int int_frac_digits: The number of fractional digits (those to the right of the decimal point) to be displayed in an internationally formatted monetary quantity.

int frac_digits: The number of fractional digits (those to the right of the decimal point) to be displayed in a formatted monetary quantity.

int p_cs_precedes: Set to 1 or 0 if the currency_symbol respectively precedes or succeeds the value for a non-negative formatted monetary quantity.

int p_sep_by_space: Set to 1 or 0 if the currency_symbol respectively is or is not separated by a space from the value for a non-negative formatted monetary quantity.

int n_cs_precedes: Set to 1 or 0 if the currency_symbol respectively precedes or succeeds the value for a negative formatted monetary quantity.
int n_sep_by_space: Set to 1 or 0 if the currency_symbol respectively is or is not separated by a space from the value for a negative formatted monetary quantity.

int p_sign_posn: Set to a value indicating the positioning of the positive_sign for a non-negative formatted monetary quantity. The value of p_sign_posn is interpreted according to the following:

0 - Parentheses surround the quantity and currency_symbol.
1 - The sign string precedes the quantity and currency_symbol.
2 - The sign string succeeds the quantity and currency_symbol.
3 - The sign string immediately precedes the currency_symbol.
4 - The sign string immediately succeeds the currency_symbol.

int n_sign_posn: Set to a value indicating the positioning of the negative_sign for a negative formatted monetary quantity. The value of n_sign_posn is interpreted according to the rules described under p_sign_posn.

See Also
bindtextdomain, textdomain, gettext, dgettext, dcgettext and setlocale

Locale.Gettext.setlocale

int setlocale(int category, string locale)

Description
The setlocale() function is used to set the program’s current locale. If locale is "C" or "POSIX", the current locale is set to the portable locale.

If locale is "", the locale is set to the default locale which is selected from the environment variable LANG.

The argument category determines which functions are influenced by the new locale:

Locale.Gettext.LC_ALL for all of the locale.
Locale.Gettext.LC_COLLATE for the functions strcoll() and strxfrm() (used by pike, but not directly accessible).
Locale.Gettext.LC_CTYPE for the character classification and conversion routines.
Locale.Gettext.LC_MONETARY for localeconv().
Locale.Gettext.LC_NUMERIC for the decimal character.
Locale.Gettext.LC_TIME for strftime() (currently not accessible from Pike).

Arguments

Returns
1 if the locale setting succeeded, 0 for failure
See Also
bindtextdomain, textdomain, gettext, dgettext, dcgettext and localeconv

Locale.Gettext.textdomain

string textdomain(string domain|void)
string textdomain(void)

Description
The textdomain() function sets or queries the name of the current domain of the active LC_MESSAGES locale category. The domainname argument is a string that can contain only the characters allowed in legal filenames.

The domainname argument is the unique name of a domain on the system. If there are multiple versions of the same domain on one system, namespace collisions can be avoided by using bindtextdomain(). If textdomain() is not called, a default domain is selected. The setting of domain made by the last valid call to textdomain() remains valid across subsequent calls to setlocale, and gettext().

The normal return value from textdomain() is a string containing the current setting of the domain. If domainname is void, textdomain() returns a string containing the current domain. If textdomain() was not previously called and domainname is void, the name of the default domain is returned.

Arguments

See Also
bindtextdomain, gettext, dgettext, dcgettext, setlocale and localeconv

14.16 Calendar

Description
This module implements calendar calculations, and base classes for time units.

14.16.1 Calendar.time_unit

Calendar.time_unit

Description

Calendar.time_unit.greater

array(string) greater()

Description
Gives a list of methods to get greater (longer) time units from this object. For
a month, this gives back ("year"), thus the method `month->year()` gives the year object.

**Calendar.time_unit.lesser**

array(string) lesser()

*Description*

Gives a list of methods to get lesser (shorter) time units. i.e, for a month, this gives back ("day") and the method `day(mixed n)` gives back that day object. The method `days()` gives back a list of possible argument values to the method `day`. Concurrently, `Array.map(o->days(),o->day)` gives a list of day objects in the object `o`.

i.e:

```
array(string) lesser() - gives back a list of possible xxx's.
object xxxs() - gives back a list of possible n's.
object xxx(mixed n) - gives back xxx n
object xxx(object(Xxx) o) - gives back the corresponing xxx
```

The list of n's (as returned from `xxxs`) are always in order.

There are two n's with special meaning, 0 and -1. 0 always gives the first `xxx`, equal to `my_obj->xxx(my_obj->xxxs()[0])`, and -1 gives the last, equal to `my_obj->xxx(my_obj->xxxs()[-1])`.

To get all `xxx`s in the object, do something like `Array.map(my_obj->xxxs(),my_obj->xxx)`. `xxx(object)` may return zero, if there was no corresponding `xxx`.

**Calendar.time_unit.‘+’, Calendar.time_unit.‘-’, Calendar.time_unit.next, Calendar.time_unit.prev**

object next()

object prev()

object ‘+(int n)

object ‘-(int n)

object ‘-(object x)

*Description*

next and prev gives the logical next and previous object. The + operator gives that logical relative object, i.e `my_day+14` gives 14 days ahead. - works the same way, but can also take an object of the same type and give the difference as an integer.

14.16.2  **Calendar.Gregorian**
Description
time units: Year, Month, Week, Day

Calendar. Gregorian.

Calendar. Gregorian.

Calendar. Gregorian. Year

Description
A Calendar.time_unit.

Lesser units: Month, Week, Day
Greater units: none

Calendar. Gregorian. Year.parse

object parse(string fmt, string arg)

Description
parse a date, create relevant object fmt is in the format "abc%edef..." where
abc and def is matched, and %x is one of those time units: %Y absolute year
%y year (70-99 is 1970-1999, 0-69 is 2000-2069) %M month (number, name or
short name) (needs %y) %W week (needs %y) %D date (needs %y, %m) %a
day (needs %y) %e weekday (needs %y, %w) %h hour (needs %d, %D or %W)
%m minute (needs %h) %s second (needs %s)

function datetime(int—void unix_time) Replacement for localtime.

function datetime_name(int—void unix_time) Replacement for ctime.

function datetime_short_name(int—void unix_time) Replacement for ctime.

Calendar. Gregorian. Stardate

Description
time unit: TNGDate

Calendar. Gregorian. Stardate. TNGDate

Description
implements ST:TNG stardates can be used as create argument to Day
14.17 Parser

14.18 Math

14.18.1 Math.Matrix

**Math.Matrix**

**Description**

This class holds matrix capabilities, and supports a range of matrix operations.

It can only - for speed and simplicity - hold floating point numbers and are only in 2 dimensions.

**Math.Matrix.+**

**Math.Matrix."+**

**Math.Matrix.add**

Object `+(object with)`

Object `"+(object with)`

Object `add(object with)`

**Description**

Add this matrix to another matrix. A new matrix is returned. The matrices must have the same size.

**Math.Matrix.cast**

Array(array(float)) cast(string to what)

Array(array(float)) cast(string to what)

**Description**

This is to be able to get the matrix values. (array) gives back a double array of floats. `m->vect()` gives back an array of floats.

**Math.Matrix.cast**

Array(array(float)) cast(string to what)

**Description**

This is to be able to get the matrix values. This gives back a double array of floats.

**Math.Matrix.create**

void create(array(array(int|float)))

void create(array(int|float))

void create(int n, int m)

void create(int n, int m, string type)

void create(int n, int m, float|int init)

void create("identity", int size)

void create("rotate", int size, float|rads, Matrix|axis)

void create("rotate", int size, float|rads, float|x, float|y, float|z)
Description
This method initializes the matrix. It is illegal to create and hold an empty matrix.

The normal operation is to create the matrix object with a double array, like
\texttt{Math.Matrix(\{\{1,2\},\{3,4\}\})}.

Another use is to create a special type of matrix, and this is told as third argument.

Currently there are only the "identity" type, which gives a matrix of zeroes except in the diagonal, where the number one (1.0) is. This is the default, too.

The third use is to give all indices in the matrix the same value, for instance zero or 42.

The forth use is some special matrixes. First the square identity matrix again, then the rotation matrix.


\begin{itemize}
\item object '×(object,with)
\item object "×(object,with)
\item object cross(object,with)
\end{itemize}

Description
Matrix cross-multiplication.


\begin{itemize}
\item object "∗(object,with)
\item object "∗(object,with)
\item object mult(object,with)
\end{itemize}

Description
Matrix multiplication.


\begin{itemize}
\item float norm()
\item float norm2()
\item object normv()
\end{itemize}

Description
Norm of the matrix, and the square of the norm of the matrix. (The later method is because you may skip a square root sometimes.)

This equals —A— or sqrt( A^0_2 + A^1_2 + ... + A^n_2 ).

It is only usable with 1xn or nx1 matrices.
m-\|v\| is equal to m*(1.0/m-\|\|), with the exception that the zero vector will still be the zero vector (no error).

\begin{itemize}
  \item \texttt{Math.Matrix.\_}
  \item \texttt{Math.Matrix.\"\_\}
  \item \texttt{Math.Matrix.sub}
\end{itemize}

\textbf{Description}
Subtracts this matrix from another. A new matrix is returned. \(-m\) is equal to \(-1^*m\).

\texttt{Math.Matrix.transpose}

\textbf{Description}
Transpose of the matrix as a new object.

\section*{14.19 Calendar.time_unit}

\texttt{Calendar.time_unit}

\textbf{Description}

\texttt{Calendar.time_unit.greater}

\textbf{Description}
Gives a list of methods to get greater (longer) time units from this object. For a month, this gives back \{"year"\}, thus the method \texttt{month->year()} gives the year object.

\texttt{Calendar.time_unit.lesser}

\textbf{Description}
Gives a list of methods to get lesser (shorter) time units. ie, for a month, this gives back \{"day"\} and the method \texttt{day(mixed n)} gives back that day object. The method \texttt{days()} gives back a list of possible argument values to the method \texttt{day}. Concurrently, \texttt{Array.map(o->days(), o->day)} gives a list of day objects in the object \(o\).
array(string) lesser() - gives back a list of possible xxx's.
object xxxs() - gives back a list of possible n's.
object xxx(mixed n) - gives back xxx n
object xxx(object(Xxx) o) - gives back the corresponding xxx

The list of n's (as returned from xxxs) are always in order.

There are two n's with special meaning, 0 and -1. 0 always gives the first xxx, equal to my_obj->xxx(my_obj->xxxs()[0]), and -1 gives the last, equal to my_obj->xxx(my_obj->xxxs()[[-1]]).

To get all xxxs in the object, do something like Array.map(my_obj->xxxs(), my_obj->xxx).

xxx(object) may return zero, if there was no corresponding xxx.

```
Calendar.time_unit.'+',
Calendar.time_unit.'-',
Calendar.time_unit.next,
Calendar.time_unit.prev
```

Description
next and prev gives the logical next and previous object. The + operator gives that logical relative object, ie my_day+14 gives 14 days ahead. - works the same way, but can also take an object of the same type and give the difference as an integer.

14.20 Calendar.Gregorian

Description
time units: Year, Month, Week, Day

14.20.1 Calendar.Gregorian.

Calendar.Gregorian.

14.20.2 Calendar.Gregorian.Year

Calendar.Gregorian.Year
CHAPTER 14. OTHER MODULES

Description
A Calendar.time unit.

Lesser units: Month, Week, Day
Greater units: none

Calendar.Gregorian.Year.parse

object parse(string fmt, string arg)

Description
Parse a date, create relevant object fmt is in the format "abc%edef..." where abc and def is matched, and %x is one of those time units: %Y absolute year
%y year (70-99 is 1970-1999, 0-69 is 2000-2069) %M month (number, name or short name) (needs %y) %W week (needs %y) %D date (needs %y, %m) %a day (needs %y) %e weekday (needs %y, %w) %h hour (needs %d, %D or %W) %m minute (needs %h) %s second (needs %s)

function datetime(int—void unix_time) Replacement for localtime.
function datetime_name(int—void unix_time) Replacement for ctime.
function datetime_short_name(int—void unix_time) Replacement for ctime.

14.20.3 Calendar.Gregorian.Stardate

Description
Time unit: TNGDate

Calendar.Gregorian.Stardate.TNGDate

Description
Implements ST:TNG stardates can be used as create argument to Day

14.21 Crypto.randomness

Description
Assorted stronger or weaker randomnumber generators.

14.21.1 Crypto.randomness.pike_random

Crypto.randomness.pike_random

Description
A pseudo random generator based on the ordinary random() function.

Crypto.randomness.pike_random.read

string read(int len)
14.22. GEOGRAPHICAL POSITION

Description
Returns a string of length len with pseudo random values.

14.21.2  Crypto.randomness.arcfour_random

Crypto.randomness.arcfour_random

Description
A pseudo random generator based on the arcfour crypto.

void create(string_secret)

Description
Initialize and seed the arcfour random generator.

string read(int_len)

Description
Return a string of the next len random characters from the arcfour random generator.

14.22  Geographical.Position

Geographical.Position

Description
This class contains a geographical position, i.e., a point on the earth's surface.

variable float lat  variable float long
Longitude (W–E) and latitude (N–S) of the position, float value in degrees. Positive number is north and east, respectively. Negative number is south and west, respectively.

Geographical.Position.cast

array cast(“array”)

Description
It is possible to cast the position to an array, ({float lat, float long}).

Geographical.Position.create

void create(float lat, float long)
void create(string_lat, string_long)
void create(string both)

Description
Constructor for this class. If fed with strings, it will perform a dwim scan on the strings. If they fail to be understood, there will be an exception.
**Geographical(Position).latitude**,  
**Geographical(Position).longitude**

```plaintext
string latitude(void|int n)
string longitude(void|int n)
```

**Description**  
Returns the nicely formatted latitude or longitude.

<table>
<thead>
<tr>
<th>n</th>
<th>format</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>17°42.19’N 42°22.2’W</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>17.703°N   42.37°W</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>17°42.18°N 42°22.2°W</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>17°42°10.4”N 42°22°12”W</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>17.703      -42.37</td>
<td></td>
</tr>
</tbody>
</table>
```

**Geographical(Position).‘==** ,  
**Geographical(Position).‘¿** ,  
**Geographical(Position).‘¿’¡** ,  
**Geographical(Position).‘¿’hash**

```plaintext
int .hash()
int ‘==(Position pos)
int ‘>(Position pos)
int ‘<(Position pos)
```

**Description**  
These exist solely for purpose of detecting equal positions, for instance when used as keys in mappings.

### 14.23 Geographical.Countries

**Description**  
subclass Country variable string iso2 ISO 2-character code aka domain name variable string name variable array(string) aka Country name and as-known-as, if any variable int former Flag that is set if this isn’t a country anymore (Like USSR.) constant array(Country) countries All known countries.

**Geographical.Countries.cast**

```plaintext
string cast("string")
```

**Description**  
It is possible to cast a country to a string, which will be the same as performing country->name;.
Geographical.Countries.from_domain

Country from_domain(string_domain)

Description
Look up a country from a domain name. Returns zero if the domain doesn’t map to a country. Note that there are some valid domains that doesn’t:

And that USA has three domains, Great Britain two:
Description
Look up a country from its name or aka. The search is case-insensitive but regards whitespace and interpunctuation.

\texttt{Geographical.Countries.[[]],}
\texttt{Geographical.Countries.'.'},

mixed '[](string.what)
mixed '->(string.what)

Description
Convenience function for getting a country the name-space way; it looks up whatever it is in the name- and domain-space and

\begin{verbatim}
> Geographical.Countries.se;
Result: Country(Sweden)
> Geographical.Countries.djibouti;
Result: Country(Djibouti)
> Geographical.Countries.com;
Result: Country(United States)
> Geographical.Countries.wallis_and_futuna_islands->iso2;
Result: "WF"
\end{verbatim}

Returns
that country if possible:

Description
$Id: module.pmod,v 1.8 2000/03/22 18:12:19 peter Exp$

\texttt{Image.load},
\texttt{Image.load},
\texttt{Image.load}

object(Image.Image) load()
object(Image.Image) load(object_,file)
object(Image.Image) load(string_filename)
object(Image.Layer) load_layer()
object(Image.Layer) load_layer(object_,file)
object(Image.Layer) load_layer(string_filename)
mapping _load()
mapping _load(object_,file)
mapping _load(string_filename)

Description
Helper function to load an image from a file. If no filename is given, Stdio.stdin is used. The result is the same as from the decode functions in Image.ANY.

Note
All data is read, ie nothing happens until the file is closed. Throws upon error.
Parser.SGML

Description
This is a handy simple parser of SGML-like syntax like HTML. It doesn't do anything advanced, but finding the corresponding end-tags.

It's used like this:

```cpp
array res=Parser.SGML()->feed(string)->finish()->result();
```

The resulting structure is an array of atoms, where the atom can be a string or a tag. A tag contains a similar array, as data.

Example: A string "<gat>&nbsp;<gurka>&nbsp;</gurka>&nbsp;<banan>&nbsp;<kiwi>&nbsp;</gat>" results in

```cpp
({
tag "gat" object with data:
  ({
tag "gurka" object with data:
    ({
      " 
    })
tag "banan" object with data:
    ({
      " 
tag "kiwi" object with data:
      ({
        " 
      })
    })
  })
})
```

ie, simple "tags" (not containers) are not detected, but containers are ended implicitly by a surrounding container _with_ an end tag.

The 'tag' is an object with the following variables:

- string name; - name of tag
- mapping args; - argument to tag
- int line, char, column; - position of tag
string file;     - filename (see create)
array(SGMLatom) data;     - contained data

Parser.SGML.create
void create()
void create(string filename)

*Description*
This object is created with this filename. It's passed to all created tags, for
debug and trace purposes.

*Note*
No, it doesn't read the file itself. See feed.

Parser.SGML.result,
Parser.SGML.finish,
Parser.SGML.feed
object feed(string s)
array finish()
array result(string s)

*Description*
Feed new data to the object, or finish the stream. No result can be used until
finish() is called.

Both finish() and result() returns the computed data.
feed() returns the called object.

14.25 Protocols.HTTP

Protocols.HTTP.delete_url
object(Protocols.HTTP.Query) delete_url(string url)
object(Protocols.HTTP.Query) delete_url(string url, mapping query variables)
object(Protocols.HTTP.Query) delete_url(string url, mapping query variables, mapping request headers)

*Description*
Sends a HTTP DELETE request to the server in the URL and returns the
created and initialized Query object. 0 is returned upon failure.

Protocols.HTTP.get_url
object(Protocols.HTTP.Query) get_url(string url)
object(Protocols.HTTP.Query) get_url(string url, mapping query variables)
object(Protocols.HTTP.Query) get_url(string url, mapping query variables, mapping request headers)
Description
Sends a HTTP GET request to the server in the URL and returns the created and initialized Query object. 0 is returned upon failure.

**Protocols.HTTP.get_url_nice, Protocols.HTTP.get_url_data**

array(string) get_url_nice(string_url)
array(string) get_url_nice(string_url, mapping_query_variables)
array(string) get_url_nice(string_url, mapping_query_variables, mapping_request_headers)
string get_url_data(string_url)
string get_url_data(string_url, mapping_query_variables)
string get_url_data(string_url, mapping_query_variables, mapping_request_headers)

Description
Returns an array of (content_type,data) and just the data string respective, after calling the requested server for the information. 0 is returned upon failure.

**Protocols.HTTP.http_encode_query**

string http_encode_query(mapping_variables)

Description
Encodes a query mapping to a string; this protects odd - in http perspective - characters like '&' and '#' and control characters, and packs the result together in a HTTP query string.

Example:

```> Protocols.HTTP.http_encode_query( ["anna":"eva","lilith":"blue"] );
Result: "lilith=blue&anna=eva"
> Protocols.HTTP.http_encode_query( ["&":"&","'="":"] );
Result: "%26amp%3b=%26&%27%3d%00%00"
```

**Protocols.HTTP.http_encode_string**

string http_encode_string(string_in)

Description
This protects all odd - see http_encode_query - characters for transfer in HTTP.

Do not use this function to protect URLs, since it will protect URL characters like '/' and '?'.

**Protocols.HTTP.post_url_data, Protocols.HTTP.post_url**
**Protocols.HTTP.post_url_nice**

array(string) post_url_nice(string url, mapping query_variables)
array(string) post_url_nice(string url, mapping query_variables, mapping request_headers)
string post_url_data(string url, mapping query_variables)
string post_url_data(string url, mapping query_variables, mapping request_headers)
object(Protocols.HTTP.Query) post_url(string url, mapping query_variables)
object(Protocols.HTTP.Query) post_url(string url, mapping query_variables, mapping request_headers)

*Description*

Similar to the get_url class of functions, except that the query variables is sent as a post request instead of a get.

**Protocols.HTTP.put_url**

object(Protocols.HTTP.Query) put_url(string url)
object(Protocols.HTTP.Query) put_url(string url, string file)
object(Protocols.HTTP.Query) put_url(string url, string file, mapping query_variables)
object(Protocols.HTTP.Query) put_url(string url, string file, mapping query_variables, mapping request_headers)

*Description*

Sends a HTTP PUT request to the server in the URL and returns the created and initialized Query object. 0 is returned upon failure.

**Protocols.HTTP.unentity**

string unentity(string s)

*Description*

Helper function for replacing HTML entities with the corresponding iso-8859-1 characters.

*Note*

All characters isn’t replaced, only those with corresponding iso-8859-1 characters.

### 14.25.1 Protocols.HTTP.Query

**Protocols.HTTP.Query**

*Description*

Open and execute a HTTP query.

**Protocols.HTTP.Query.set_callbacks**,
Protocols.HTTP.Query.async_request

object set_callback(function okhttp, function onfail, mixed ...extra)
object async_request(string server, int port, string query);
object async_request(string server, int port, string query,
mapping headers, void|string data);

Description
Setup and run an asynchronous request, otherwise similar to thread_request.
request_ok(object httpquery,...extra args) will be called when connection is complete,
and headers are parsed.
request_fail(object httpquery,...extra args) is called if the connection fails.
variable int ok Tells if the connection is successful. variable int errno Errno
copied from the connection.
variable mapping headers Headers as a mapping. All header names are in lower
case, for convenience.
variable string protocol Protocol string, ie "HTTP/1.0".
variable int status variable string status_desc Status number and description (ie,
200 and "ok").
variable mapping hostname_cache Set this to a global mapping if you want to
use a cache, prior of calling *request().
variable mapping async_dns Set this to an array of Protocols.DNS.async_clients,
if you wish to limit the number of outstanding DNS requests. Example: async_dns=allocate(20,Protocols.DNS.async_client);

Returns
the called object

Protocols.HTTP.Query.cast
array cast("array")

Description
Gives back (–mapping headers,string data, string protocol,int status,string status_desc);

Protocols.HTTP.Query.cast
mapping cast("mapping")

Description
Gives back headers — ("protocol":protocol, "status":status number, "status_desc":status
description, "data":data);
CHAPTER 14. OTHER MODULES

**Description**
Gives back the answer as a string.

**Protocols.HTTP.Query.data**

```cpp
string data()
```

**Description**
Gives back the data as a string.

**Protocols.HTTP.Query.downloaded_bytes**

```cpp
int downloaded_bytes()
```

**Description**
Gives back the number of downloaded bytes.

**Protocols.HTTP.Query.thread_request**

```cpp
object thread_request(string_server, int_port, string_query);
object thread_request(string_server, int_port, string_query,
  mapping_headers, void|string_data);
```

**Description**
Create a new query object and begin the query.

The query is executed in a background thread; call '()' in this object to wait for
the request to complete.

'query' is the first line sent to the HTTP server; for instance ”GET /index.html
HTTP/1.1”.

headers will be encoded and sent after the first line, and data will be sent after
the headers.

**Returns**
the called object

**Protocols.HTTP.Query.total_bytes**

```cpp
int total_bytes()
```

**Description**
Gives back the size of a file if a content-length header is present and parsed at
the time of evaluation. Otherwise returns -1.

```cpp
object(pseudofile) file() object(pseudofile) file(mapping newheaders, void—
  mapping removeheaders) object(pseudofile) datafile();
```

**Description**
datafile() doesn’t give the complete request, just the data.

newheaders, removeheaders is applied as: (oldheaders|newheaders)—removeheadersMake
sure all new and remove-header indices are lower case.

```cpp
void async_fetch(function done, callback);
```

Fetch all data in background.
Protocols.HTTP.Query

int `()`

Description
Wait for connection to complete.

Returns
1 on successful connection, 0 if failed

14.26 Protocols.LysKOM


Protocols.LysKOM.Session

Description
variable user This variable contains the person that are logged in.

Protocols.LysKOM.Session.create

void create(string server)
void create(string server, mapping options)

Description
Initializes the session object, and opens a connection to that server.

options is a mapping of options,

```
({
    "login" : int|string login as this person number
            (get number from name)
    "create" : string create a new person and login with it
    "password" : string send this login password
    "invisible" : int(0..1) if set, login invisible
              advanced
    "port" : int(0..65535) server port (default is 4894)
    "whoami" : string present as this user
              (default is from uid/getpwent and hostname)
})
```

See Also
Connection

Protocols.LysKOM.Session.create_person

object create_person(string name, string password)
Description
Create a person, which will be logged in.

Returns
the new person object

**Protocols.LysKOM.Session.create_text**

```plaintext
object create_text(string subject, string body, mapping options)
object create_text(string subject, string body, mapping options, function callback, mixed ...extra)
```

Description
Creates a new text.

If "callback" are given, this function will be called when the text is created, with the text as first argument. Otherwise, the new text is returned.

options is a mapping that may contain:

```plaintext
{
    "recpt" : Conference|array(Conference)
        recipient conferences
    "cc" : Conference|array(Conference)
        cc-recipient conferences
    "bcc" : Conference|array(Conference)
        bcc-recipient conferences
    "comm_to" : Text|array(Text)
        what text(s) is commented
    "foot_to" : Text|array(Text)
        what text(s) is footnoted
    "anonymous" : int(0..1)
        send text anonymously
}
```

**Note**
The above marked with a '*' is only available on a protocol 10 server. A LysKOM error will be thrown if the call fails.

See Also
Conference.create_text, Text.comment and Text.footnote

**Protocols.LysKOM.Session.login**

```plaintext
object login(int user_no, string password)
object login(int user_no, string password, int invisible)
```

Description
Performs a login. Returns 1 on success or throws a lyskom error.
14.26. PROTOCOLS.LYSKOM

Returns the called object

Protocols.LysKOM.Session.logout

object logout()

Description
Logouts from the server.

Returns the called object

Protocols.LysKOM.Session.send_message

object send_message(string message, mapping options)

Description
Sends a message.

options is a mapping that may contain:

([ "recpt" : Conference recipient conference ])

Protocols.LysKOM.Session.try_complete_person

description
Runs a LysKOM completion on the given string, returning an array of confzinfos of the match.

14.26.2 Protocols.LysKOM.Connection

Protocols.LysKOM.Connection

Description
This class contains nice abstraction for calls into the server. They are named "call", "async_call" or "async_cb_call", depending on how you want the call to be done.

mixed /call/(mixed,...args)
object async_/call/(mixed,...args)
object async_cb_/call/(function, callback, mixed,...args)
CHAPTER 14. OTHER MODULES

Description
Do a call to the server. This really clones a request object, and initialises it. /call/ is to be read as one of the calls in the lyskom protocol. (’-’ is replaced with ’_’) (ie, logout, async_login or async_cb_get_conf_stat.)

The first method is a synchronous call. This will send the command, wait for the server to execute it, and then return the result.

The last two is asynchronous calls, returning the initialised request object.

variable int protocol_level
variable string session
software variable string software
variable string software

version

Description of the connected server.

Protocols.LysKOM.Connection.create

void create(string server)

void create(string server, mapping options)

Description

{
    "login" : int|string login as this person number
    (get number from name)
    "password" : string send this login password
    "invisible" : int(0..1) if set, login invisible
    advanced
    "port" : int(0..65535) server port (default is 4894)
    "whoami" : string present as this user
    (default is from uid/getpwent and hostname)
}

14.26.3 Protocols.LysKOM.Request

Description
This class contains nice abstraction for calls into the server. They are named "call", "async_call" or "async_cb_call", depending on how you want the call to be done.

Protocols.LysKOM.Request._Request

Protocols.LysKOM.Request._Request

Description
This is the main request class. All lyskom request classes inherits this class.

Protocols.LysKOM.Request._Request.sync

Protocols.LysKOM.Request._Request.async

void async(mixed... args)
mixed sync(mixed... args)
**Description**
initialise an asynchronous or a synchronous call, the latter is also evaluating the result. This calls 'indata' in itself, to get the correct arguments to the lyskom protocol call.

```plaintext
Protocols.LysKOM.Request._Request._reply,
Protocols.LysKOM.Request._Request.reply
```

mixed _reply(object|array what)
mixed reply(object|array what)

**Description**
_reply is called as callback to evaluate the result, and calls reply in itself to do the real work.

```plaintext
Protocols.LysKOM.Request._Request._
```
mixed ()()

**Description**
wait for the call to finish.

variable int ok tells if the call is executed ok variable object error how the call failed The call is completed if (ok — error).

```plaintext
Protocols.LysKOM.Request._Request._async,
Protocols.LysKOM.Request._Request._sync
```

void _async(int call, _mixed data)
mixed _sync(int call, _mixed data)

**Description**
initialise an asynchronous or a synchronous call, the latter is also evaluating the result. These are called by async and sync respectively.

### 14.27 Protocols.DNS

**Description**

### 14.27.1 Protocols.DNS.client

**Protocols.DNS.client**

**Description**
Synchronous DNS client.
Protocols.DNS.client.create

void create()
void create(void|string|array server, void|int|array domain)

Description

Protocols.DNS.client.gethostbyname, Protocols.DNS.client.gethostbyaddr

array gethostbyname(string hostname)
array gethostbyaddr(string hostip)

Description
Querys the host name or ip from the default or given DNS server. The result is a mapping with three elements,

{{
  string hostname [0] hostname
  array(string) ip [1] ip number(s)
  array(string) ip [2] dns name(s)
}}

Protocols.DNS.client.get_primary_mx

string get_primary_mx(string hostname)

Description
Querys the primary mx for the host.

Returns
the hostname of the primary mail exchanger
Chapter 15

The preprocessor

Pike has a built-in C-style preprocessor. The preprocessor reads the source before it is compiled and removes comments and expands macros. The preprocessor can also remove code depending on an expression that is evaluated when you compile the program. The preprocessor helps to keep your programming abstract by using defines instead of writing the same constant everywhere.

It currently works similar to old C preprocessors but has a few extra features. This chapter describes the different preprocessor directives. This is what it can do:
Chapter 16

Builtin functions

This chapter is a reference for all the builtin functions in Pike. They are listed in alphabetical order.

_disable_threads - temporarily disable threads

object _disable_threads();

Description
This function first posts a notice to all threads that it is time to stop. It then waits until all threads actually *have* stopped, and then then returns an object. All other threads will be blocked from running until that object has been freed/destroyed. This function can completely block Pike if used incorrectly. Use with extreme caution.

_do_call_outs - do all pending call_outs

void _do_call_out();

Description
This function runs all pending call_outs that should have been run if Pike returned to the backend. It should not be used in normal operation.

As a side-effect, this function sets the value returned by time(1) to the current time.

See Also
call_out, find_call_out and remove_call_out

_exit - Really exit

void _exit(int returncode);

Description
This function does the same as exit, but doesn’t bother to clean up the Pike interpreter before exiting. This means that no destructors will be called, caches
will not be flushed, file locks might not be released, and databases might not be
closed properly. Use with extreme caution.

See Also
exit

_locate_references - locate where an object is referenced from

mapping(string:int) _locate_references(string|array|mapping|multiset|function|object|program o);

Description
This function is mostly intended for debugging. It will search through all data
structures in Pike looking for $o$ and print the locations on stderr. $o$ can be
anything but int or float.

_memory_usage - check memory usage

mapping(string:int) _memory_usage();

Description
This function is mostly intended for debugging. It delivers a mapping with
information about how many arrays/mappings/strings etc. there are currently
allocated and how much memory they use. Try evaluating the function in hilfe
to see precisely what it returns.

See Also
_verify_internals

_next - find the next object/array/whatever

mixed _next(mixed p);

Description
All objects, arrays, mappings, multisets, programs and strings are stored in
linked lists inside Pike. This function returns the next object/array/mapping/string/etc
in the linked list. It is mainly meant for debugging Pike but can also be used
to control memory usage.

See Also
next_object and _prev

_prev - find the previous object/array/whatever

mixed _prev(mixed p);

Description
This function returns the 'previous' object/array/mapping/etc in the linked list.
It is mainly meant for debugging Pike but can also be used to control memory
usage. Note that this function does not work on strings.

See Also
_next
_refs - find out how many references a pointer type value has

int _refs(string|array|mapping|multiset|function|object|program o);

*Description*
This function checks how many references the value o has. Note that the number of references will always be at least one since the value is located on the stack when this function is executed. _refs() is mainly meant for debugging Pike but can also be used to control memory usage.

*See Also*
_next and _prev

_verify internals - check Pike internals

void _verify internals();

*Description*
This function goes through most of the internal Pike structures and generates a fatal error if one of them is found to be out of order. It is only used for debugging.

_abs - absolute value

float abs(float f); int abs(int f); object abs(object f);

*Description*
Return the absolute value for f. f can be a Gmp-object.

_acos - trigonometrical inverse cosine

float acos(float f);

*Description*
Return the arcus cosine value for f. The result will be in radians.

*See Also*
_cos and _asin

_add constant - add new predefined functions or constants

void add constant(string name, mixed value);
void add constant(string name);

*Description*
This function adds a new constant to Pike, it is often used to add builtin functions. All programs compiled after add constant function is called can access 'value' by the name given by 'name'. If there is a constant called 'name' already, it will be replaced by by the new definition. This will not affect already compiled programs.

Calling add constant without a value will remove that name from the list of constants. As with replacing, this will not affect already compiled programs.
Example
add_constant("true",1);
add_constant("false",0);
add_constant("PI",4.0);
add_constant("sqr",lambda(mixed x) { return x * x; });
add_constant("add_constant");

See Also
all_constants

void add_include_path(string path);

Description
This function adds another directory to the search for include files. This is the same as the command line option -I. Note that the added directory will only be searched when using \"\" to quote the included file.

See Also
remove_include_path and #include

void add_module_path(string path);

Description
This function adds another directory to the search for modules. This is the same as the command line option -M. For more information about modules, see chapter 7.4.

See Also
remove_module_path

void add_program_path(string path);

Description
This function adds another directory to the search for programs. This is the same as the command line option -P. For more information about programs, see section 4.2.3.

See Also
remove_program_path

array aggregate(mixed ... elems);
({ elem1, elem2, ... });

Description
Construct an array with the arguments as indices. This function could be
written in Pike as:

```
array aggregate(mixed ... elems) – return elems;
```

**Note**
Arrays are dynamically allocated there is no need to declare them like int a[10]=allocate(10); (and it isn’t possible either) like in C, just array(int) a=allocate(10); will do.

**See Also**
sizeof, arrayp and allocate

### aggregate_mapping - construct a mapping

```
mapping aggregate_mapping(mixed ... elems);
([ key1:val1, key2:val2, ... ]);)
```

**Description**
Groups the arguments together two and two to key-index pairs and creates a mapping of those pairs. The second syntax is always preferable.

**See Also**
sizeof, mappingp and mkmapping

### aggregate_multiset - construct a multiset

```
multiset aggregate_multiset(mixed ... elems);
(< elem1, elem2, ... >);
```

**Description**
Construct a multiset with the arguments as indices. This function could be written in Pike as:

```
multiset aggregate(mixed ... elems) – return mkmultiset(elems); “
```

The only problem is that mkmultiset is implemented using aggregate_multiset...

**See Also**
sizeof, multisetp and mkmultiset

### alarm - set an alarm clock for delivery of a signal

```
int alarm(int seconds);
```

**Description**
alarm arranges for a SIGALRM signal to be delivered to the process in seconds seconds.

If seconds is zero, no new alarm is scheduled.

In any event any previously set alarm is canceled.

**Returns**
alarm returns the number of seconds remaining until any previously scheduled alarm was due to be delivered, or zero if there was no previously scheduled alarm.
See Also

all_constants - return all predefined constants

mapping (string:mixed) all_constant();

Description
Returns a mapping containing all constants, indexed on the names of the constant, and with the value of the efun as argument.

See Also
add_constant

allocate - allocate an array

array allocate(int size);

Description
Allocate an array of size elements and initialize them to zero.

Example
array a=allocate(17);

Note
Arrays are dynamically allocated there is no need to declare them like int a[10]=allocate(10); (and it is not possible either) like in C, just array(int) a=allocate(10); will do.

See Also
sizeof, aggregate and arrayp

arrayp - is the argument an array?

int arrayp(mixed arg);

Description
Returns 1 if arg is an array, zero otherwise.

See Also
allocate, intp, programp, floatp, stringp, objectp, mappingp, multisetp and functionp

array_sscanf - sscanf to an array

array array_sscanf(string data, string format);

Description
This function works just like sscanf, but returns the matched results in an array instead of assigning them to lvalues. This is often useful for user-defined sscanf strings.

See Also
sscanf and '/
**asin** - trigonometrical inverse sine

```c
float asin(float f);
```

*Description*

Returns the arcus sinus value for `f`.

*See Also*

- `sin` and `acos`

**atan** - trigonometrical inverse tangent

```c
float atan(float f);
```

*Description*

Returns the arcus tangent value for `f`.

*See Also*

- `atan2`, `tan`, `asin` and `acos`

**atan2** - trigonometrical inverse tangent

```c
float atan2(float f1, float f2);
```

*Description*

Returns the arcus tangent value for `f1/f2`.

*See Also*

- `atan`, `tan`, `asin` and `acos`

**atexit** - schedule a callback for when Pike exits

```c
void atexit(function callback);
```

*Description*

This function puts the `callback` in a queue of callbacks to call when Pike exits. Please note that atexit callbacks are not called if Pike exits abnormally.

*See Also*

- `exit`

**backtrace** - get a description of the call stack

```c
array(array) backtrace();
```

*Description*

This function returns a description of the call stack at this moment. The description is returned in an array with one entry for each call in the stack. Each entry has this format:

```c
(–
  file,       /* a string with the filename if known, else zero */
  line,       /* an integer containing the line if known, else zero */
  function,   /* The function pointer to the called function */
  mixed—void ..., /* The arguments the function was called with */
)
The current call frame will be last in the array, and the one above that the last but one and so on.

See Also
catch and throw

basename - get the base of a filename

string basename(string filename);

Description
This function returns the base of a filename, for instance the base of "/home/hubbe/bin/pike" would be "pike".

See Also
dirname and explode_path

call_function - call a function with arguments

mixed call_function(function fun,mixed ... args);
mixed fun (mixed ... args);

Description
This function takes a function pointer as first argument and calls this function with the rest of the arguments as arguments. Normally, you will never have to write call_function(), because you will use the second syntax instead.

See Also
backtrace and Simulate.get_function

call_out - make a delayed call to a function

mixed call_out(function f, float|int delay, mixed ... args);

Description
Call_out places a call to the function f with the argument args in a queue to be called in about delay seconds. The return value identifies this call out. The return value can be sent to find_call_out or remove_call_out to remove the call out again.

See Also
remove_call_out, find_call_out and call_out_info

call_out_info - get info about all call outs

array(array) call_out_info();

Description
This function returns an array with one entry for each entry in the call out queue. The first in the queue will be in index 0. Each index contains an array that looks like this:

(-}
See Also
call_out, find_call_out and remove_call_out

catch

```c
catch { commands }
catch ( expression );
```

**Description**
catch traps exceptions such as run time errors or calls to throw() and returns the argument given to throw. For a run time error, this value is (~"error message", backtrace ~)

See Also
throw

cd - change directory

```c
int cd(string s);
```

**Description**
Change the current directory for the whole Pike process, return 1 for success, 0 otherwise.

See Also
getcwd

ceil - truncate a number upward

```c
float ceil(float f);
```

**Description**
Return the closest integer value higher or equal to f.

**Note**
`ceil()` does not return an int, merely an integer value stored in a float.

See Also
floor and round

chmod - change mode of a file in the filesystem

```c
void chmod(string filename, int mode);
```

**Description**
Sets the protection mode of the given file. It will throw if it fails.
CHAPTER 16. BUILTIN FUNCTIONS

See Also
Stdio.File->open and errno

c**lone** - clone an object from a program

```php
object clone(program p,mixed ...  args);
```

**Description**
c**lone**() creates an object from the program p. Or in C++ terms: It creates an instance of the class p. This clone will first have all global variables initialized, and then create() will be called with args as arguments.

See Also
new, destruct, compile_string and compile_file

c**olumn** - extract a column

```php
array column(array data,mixed index)
```

**Description**
This function is exactly equivalent to:

```php
map_array(data, lambda(mixed x,mixed y) { return x[y]; }, index)
```

Except of course it is a lot shorter and faster. That is, it indices every index in the array data on the value of the argument index and returns an array with the results.

**Example**

```text
> column( ({ ({1,2}), ({3,4}), ({5,6}) }), 1)
Result: ({2, 4, 6})
```

See Also
rows

c**ombine_path** - concatenate paths

```php
string combine_path(string absolute, string relative);
```

**Description**
Concatenate a relative path to an absolute path and remove any "/", "../" or "/." to produce a straightforward absolute path as a result.

**Example**

```text
> combine_path("/foo/bar/","..");
Result: /foo
> combine_path("/foo/bar/","../apa.c");
Result: /foo/apa.c
> combine_path("/foo/bar","./sune.c");
```
Result: /foo/bar/sune.c

See Also
getcwd and Stdio.append_path

**compile** - compile a string to a program

program compile(string program);

Description

*compile* takes a piece of Pike code as a string and compiles it into a clonable program. Note that *prog* must contain the complete source for a program. You can not compile a single expression or statement. Also note that *compile* does not preprocess the program. To preprocess the program you can use *compile_string* or call the preprocessor manually by calling *cpp*.

See Also
clone, compile_string, compile_file and cpp

**compile_file** - compile a file to a program

program compile_file(string filename);

Description

This function will compile the file *filename* to a Pike program that can later be used for cloning. It is the same as doing *compile_string*(Stdio.read_file(filename), filename).

See Also
clone and compile_string

**compile_string** - compile a string to a program

program compile_string(string prog, string name);

Description

Equal to *compile*(cpp(prog, name));

See Also

compile_string and clone

**copy_value** - copy a value recursively

mixed copy_value(mixed value);

Description

Copy value will copy the value given to it recursively. If the result value is changed destructively (only possible for multisets, arrays and mappings) the copied value will not be changed. The resulting value will always be equal to the copied (tested with the efun equal), but they may not be the same value. (tested with ==)

See Also
equal
**COS** - trigonometrical cosine

```c
float cos(float f);
```

*Description*
Returns the cosine value for `f`.

*See Also*
`acos` and `sin`

**cpp** - run the preprocessor on a string

```c
string cpp ( string source, string filename);
```

*Description*
This function runs the Pike preprocessor on a string. The second argument `filename` will be used for inserting `#line` statements into the result.

*See Also*
`compile`, `compile_string` and `compile_file`

**crypt** - crypt a password

```c
string crypt(string password);
int crypt(string typed_password, string crypted_password);
```

*Description*
This function crypts and verifies a short string. (normally only the first 8 characters are significant) The first syntax crypts the string `password` into something that is hopefully hard to decrypt, and the second function crypts the first string and verifies that the crypted result matches the second argument and returns 1 if they matched, 0 otherwise.

*Example*
To crypt a password use:

```

```

To see if the same password was used again use:

```

```

**ctime** - convert time int to readable date string

```c
string ctime(int current_time);
```
Convert the output from a previous call to time() into a readable string containing the current year, month, day and time.

Example

> ctime(time());
Result: Wed Jan 14 03:36:08 1970

See Also
time, localtime, mktime and gmtime

decode_value - decode a value from a string

mixed decode_value(string coded_value);

Description

This function takes a string created with encode_value() or encode_value_canonic() and converts it back to the value that was coded.

See Also
encode_value and encode_value_canonic
describe_backtrace - make a backtrace readable

string describe_backtrace(array(array) backtrace);

Description

Returns a string containing a readable message that describes where the backtrace was made. The argument 'backtrace' should normally be the return value from a call to backtrace()

See Also
backtrace and describe_error
describe_error - get the error message from a backtrace

string describe_error(array(array) backtrace);

Description

Returns only the error message in a backtrace. If there is no message in it, a fallback message is returned.

See Also
backtrace and describe_backtrace
destruct - destruct an object

void destruct(object o);

Description

Destruct marks an object as destructed, all pointers and function pointers to this object will become zero. The destructed object will be freed from memory as soon as possible. This will also call o->destroy.
See Also
clone

dirname - find the directory part of a path

string dirname(string path);

Description
This function returns the directory part of a path. For example, the directory
part of "/home/hubbe/bin/pike" would be "/home/hubbe/bin".

See Also
basename and explode_path

encode_value - code a value into a string

string encode_value(mixed value);

Description
This function takes a value, and converts it to a string. This string can then be
saved, sent to another Pike process, packed or used in any way you like. When
you want your value back you simply send this string to decode_value() and it
will return the value you encoded.

Almost any value can be coded, mappings, floats, arrays, circular structures etc.
At present, objects, programs and functions cannot be saved in this way. This
is being worked on.

See Also
decode_value, sprintf and encode_value_canonic

encode_value_canonic - code a value into a string on canonical form

string encode_value_canonic(mixed value)

Description
Takes a value and converts it to a string on canonical form, much like en-
code_value(). The canonical form means that if an identical value is encoded,
it will produce exactly the same string again, even if it’s done at a later time
and/or in another Pike process. The produced string is compatible with de-
code_value().

Note that this function is more restrictive than encode_value() with respect to
the types of values it can encode. It will throw an error if it can’t encode to a
canonical form.

See Also
encode_value and decode_value

enumerate - create an array with an enumeration

array(int) enumerate(int n);
array enumerate(int n, void|mixed step, void|mixed
start, void|function operator);
Description
Create an enumeration, useful for initializing arrays or as first argument to map or foreach.

For instance, `enumerate(4)` gives `{0,1,2,3}`.

*Advanced use:* the resulting array is calculated like this:

```php
array enumerate(int n, mixed step, mixed start, function operator)
{
    array res=({});
    for (int i=0; i<n; i++)
    {
        res+=({start});
        start=operator(start, step);
    }
    return res;
}
```

The default values is step=1, start=0, operator=add.

*See Also*
map and foreach

`equal` - check if two values are equal or not

```php
int equal(mixed a, mixed b);
```

*Description*
This function checks if the values a and b are equal. For all types but arrays, multisets and mappings, this operation is the same as doing `a == b`. For arrays, mappings and multisets however, their contents are checked recursively, and if all their contents are the same and in the same place, they are considered equal.

*Example*
```bash
> ({{ 1 }}) == ({{ 1 }});
Result: 0
> equal( ({{ 1 }}), ({{ 1 }}) );
Result: 1
>
```

*See Also*
copy_value

`errno` - return system error number

```php
int errno();
```

*Description*
This function returns the system error from the last file operation. Note that
CHAPTER 16. BUILTIN FUNCTIONS

you should normally use the function errno in the file object instead.

See Also
Stdio.File->errno and strerror

### exece

```pike
int exece(string file, array(string) args);
int exece(string file, array(string) args, mapping(string:string) env);
```

Description
This function transforms the Pike process into a process running the program specified in the argument 'file' with the argument 'args'. If the mapping 'env' is present, it will completely replace all environment variables before the new program is executed. This function only returns if something went wrong during exece(), and in that case it returns zero.

Note
The Pike driver _dies_ when this function is called. You must use fork() if you wish to execute a program and still run the Pike driver.

Example
```pke
exece("/bin/ls", ({"-l"}));
exece("/bin/sh", ({"-c", "echo $HOME"}), ("HOME":"/not/home"));
```

See Also
fork and Stdio.File->pipe

### explode_path - Split a path into components

```pke
array(string) explode_path(string path);
```

Description
This function divides a path into its components. This might seem like it could be done by dividing the string on "/", but that would not work on other operating systems.

Example
```pke
> explode_path("/home/hubbe/bin/pike"); Result: ("home", "hubbe", "bin", "pike")
```

### exit - exit Pike interpreter

```pke
void exit(int returncode);
```

Description
This function exits the whole Pike program with the return code given. Using exit() with any other value than 0 indicates that something went wrong during execution. See your system manuals for more information about return codes.

### exp - natural exponent

```pke
float exp(float f);
```
Description
Return the natural exponent of \( f \).

See Also
pow and log

**file_stat** - stat a file

```php
array(int) file_stat(string file);
array(int) file_stat(string file, 1);
array(int) file->stat();
```

Description
file_stat returns an array of integers describing some properties about the file. Currently file_stat returns 7 entries:

```php
(
    int mode [0] file mode, protection bits etc. etc.
    int size [1] file size for regular files,
        -2 for dirs,
        -3 for links,
        -4 for otherwise
    int atime [2] last access time
    int mtime [3] last modify time
    int ctime [4] last status time change
    int uid [5] The user who owns this file
    int gid [6] The group this file belongs to
)
```

If you give 1 as a second argument, file_stat does not follow links.
You can never get -3 as size if you don’t give a second argument.

If there is no such file or directory, zero is returned.

See Also
get_dir

**file_truncate** - truncate a file

```php
int file_truncate(string file, int length);
```

Description
Truncates a file to that length. Returns 1 if ok, 0 if failed.

**filter** - map a function over elements and filter

```php
array filter(array arr, function fun, mixed ...extra);
mixed filter(mixed arr, void|mixed fun, void|mixed ...extra);
```

Description
Calls the given function for all elements in `arr`, and keeps the elements in `arr` that resulted in a non-zero value from the function.

<table>
<thead>
<tr>
<th>arr</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>keep = map(arr, fun, @extra); for (i=0; i&lt;sizeof(arr); i++) ~if (keep[i]) res+ = (-arr[i]) ;</td>
</tr>
<tr>
<td>multiset</td>
<td>(multiset) filter((array) arr, fun, @extra);</td>
</tr>
<tr>
<td>mapping</td>
<td>ind = indices(arr), val = values(arr);</td>
</tr>
<tr>
<td>program</td>
<td>keep = map(val, fun, @extra);</td>
</tr>
<tr>
<td>function</td>
<td>for (i=0; i&lt;sizeof(keep); i++) ~if (keep[i]) res[ind[i]] = val[i];</td>
</tr>
<tr>
<td>string</td>
<td>(string) filter((array) arr, fun, @extra);</td>
</tr>
<tr>
<td>object</td>
<td>if arr-&gt;cast: ~try filter((array) arr, fun, @extra); ~try filter((mapping) arr, fun, @extra); ~try filter((multiset) arr, fun, @extra);</td>
</tr>
</tbody>
</table>

*Returns*
the same datatype as given, the exception are program and function that gives a mapping back

*See Also*
map and foreach

### find_call_out - find a call out in the queue

```
int find_call_out(function f);
int find_call_out(mixed id);
```

*Description*
This function searches the call out queue. If given a function as argument, it looks for the first call out scheduled to that function. The argument can also be a call out id as returned by call_out, in which case that call_out will be found. (Unless it has already been called.) find_call_out will then return how many seconds remains before that call will be executed. If no call is found, zero_type(find_call_out(f)) will return 1.

*See Also*
call_out, remove_call_out and call_out_info

### floatp - is the argument a float?

```
int floatp(mixed arg);
```

*Description*
Returns 1 if `arg` is a float, zero otherwise.

*See Also*
intp, programp, arrayp, stringp, objectp, mappingp, multisetp and functionp
**floor** - truncate a number downward

```php
float floor(float f);
```

**Description**
Return the closest integer value lower or equal to \( f \).

**Note**
`floor()` does **not** return an int, merely an integer value stored in a float.

**See Also**
`ceil` and `round`

---

**fork** - fork the process in two

```php
int fork();
```

**Description**
Fork splits the process in two, and for the parent it returns the pid of the child. Refer to your Unix manual for further details.

**Note**
This function cause endless bugs if used without proper care.

Some operating systems have problems if this function is used together with threads.

**See Also**
`Process.exec` and `Stdio.File->pipe`

---

**function_name** - return the name of a function, if known

```php
string function_name(function f);
```

**Description**
This function returns the name of the function \( f \). If the function is a pre-defined function in the driver, zero will be returned.

**See Also**
`function_object` and `Simulate.get_function`

---

**function_object** - return what object a function is in

```php
object function_object(function f);
```

**Description**
Function_object will return the object the function \( f \) is in. If the function is a predefined function from the driver, zero will be returned.

**See Also**
`function_name` and `Simulate.get_function`

---

**functionp** - is the argument a function?

```php
int functionp(mixed arg);
```
**Description**

Returns 1 if `arg` is a function, zero otherwise.

**See Also**

`intp`, `programp`, `arrayp`, `stringp`, `objectp`, `mappingp`, `multisetp` and `floatp`

---

### gc - do garbage collection

```c
int gc();
```

**Description**

This function checks all the memory for cyclic structures such as arrays containing themselves and frees them if appropriate. It also frees up destructed objects. It then returns how many arrays/objects/programs/etc. it managed to free by doing this. Normally there is no need to call this function since Pike will call it by itself every now and then. (Pike will try to predict when 20% of all arrays/object/programs in memory is 'garbage' and call this routine then.)

---

### get_dir - read a directory

```c
array(string) get_dir(string dirname);
```

**Description**

Returns an array of all filenames in the directory `dirname`, or zero if no such directory exists.

**See Also**

`mkdir` and `cd`

---

### get_profiling_info - get profiling information

```c
array(int|mapping(string:array(int))) get_profiling_info(program prog);
```

**Description**

Returns an array with two elements, the first of which is the number of times the program `prog` has been cloned. The second element is a `mapping(string:array(int))` from function name to an `array(int)` with two elements. The first element of this array is the number of times the function has been called, while the second element is the total time (in milliseconds) spent in the function so far.

**Note**

This function is only available if Pike was compiled with the option ‘with-profiling’.

---

### getcwd - return current working directory

```c
string getcwd();
```

**Description**

getcwd returns the current working directory.
See Also

cd

getenv - get an environment variable

string getenv(string varname);

Description

Returns the value of the environment variable with the name varname, if no such environment variable exists, zero is returned.

Note

This function is provided by master.pike.

getpid - get the process id of this process

int getpid();

Description

This returns the pid of this process. Useful for sending signals to yourself.

See Also

kill, fork and signal

glob - match strings against globs

int glob(string glob, string str); or
array(string) glob(string glob, array(string) arr);

Description

This function matches "globs". In a glob string a question sign matches any character and an asterisk matches any string. When given two strings as argument a true/false value is returned which reflects if the str matches glob. When given an array as second argument, an array containing all matching strings is returned.

See Also

sscanf and Regexp

gmtime - break down time() into intelligible components

mapping(string:int) gmtime(int time);

Description

This function works like localtime but the result is not adjusted for the local time zone.

See Also

localtime, time, ctime and mktime

has index - does the index exist?

int has_index(string haystack, int index);
int has_index(array haystack, int index);
int has_index(mapping haystack, mixed index);
Description
Search for `index` in `haystack`. Returns `1` if `index` is in the index domain of `haystack`, or `0` if not found. The `has_index` function is equivalent (but sometimes faster) to:

```Plain
search(indices(haystack), index) != -1
```

Note
A negative index in strings and arrays as recognized by the index operators `[]` and `[]=` is not considered a proper index by `has_index`.

See Also
`has_value`, `indices`, `search`, `values` and `zero_type`

### has_value - does the value exist?

```Plain
int has_value(string haystack, int value);
int has_value(array haystack, int value);
int has_value(mapping haystack, mixed value);
```

Description
Search for `value` in `haystack`. Returns `1` if `value` is in the value domain of `haystack`, or `0` if not found. The `has_value` function is in all cases except for strings equivalent (but sometimes faster) to:

```Plain
search(values(haystack), value) != -1
```

For strings, `has_value` is equivalent to:

```Plain
search(haystack, value) != -1
```

See Also
`has_index`, `indices`, `search`, `values` and `zero_type`

### hash - hash a string

```Plain
int hash(string s);
int hash(string s, int max);
```

Description
This function will return an `int` derived from the string `s`. The same string will always hash to the same value. If a second argument is given, the result will be `<= 0` and lesser than that argument.

### indices - return an array of all index possible for a value

```Plain
array indices(string|array|mapping|multiset|object foo);
```

Description
`indices` returns an array of all values you can use as index when indexing `foo`. For strings and arrays this is simply an array of the ascending numbers. For mappings and multisets, the array may contain any kind of value. For objects, the result is an array of strings.
See Also
values

_is_absolute_path_ - Is the given pathname relative or not?
int is_absolute_path(string path);

Description
Returns 1 if path is an absolute path, 0 otherwise.

_intp_ - is the argument an int?

array intp(mixed arg);

Description
Returns 1 if arg is an int, zero otherwise.

See Also
arrayp, programp, floatp, stringp, objectp, mappingp, multisetp and functionp

_kill_- send signal to other process
int kill(int pid, int signal);

Description
Kill sends a signal to another process. If something goes wrong -1 is returned, 0 otherwise.

Some signals and their supposed purpose:

- **SIGHUP**  
  Hang-up, sent to process when user logs out

- **SIGINT**  
  Interrupt, normally sent by ctrl-c

- **SIGQUIT**  
  Quit, sent by ctrl-"\

- **SIGILL**  
  Illegal instruction

- **SIGTRAP**  
  Trap, mostly used by debuggers

- **SIGABRT**  
  Aborts process, can be caught, used by Pike whenever something goes seriously wrong.

- **SIGBUS**  
  Bus error

- **SIGFPE**  
  Floating point error (such as division by zero)

- **SIGKILL**  
  Really kill a process, cannot be caught

- **SIGUSR1**  
  Signal reserved for whatever you want to use it for.

- **SIGSEGV**  
  Segmentation fault, caused by accessing memory where you shouldn’t. Should never happen to Pike.

- **SIGUSR2**  
  Signal reserved for whatever you want to use it for.

- **SIGALRM**  
  Signal used for timer interrupts.

- **SIGTERM**  
  Termination signal

- **SIGSTKFLT**  
  Stack fault

- **SIGCHLD**  
  Child process died

- **SIGCONT**  
  Continue suspended

- **SIGSTOP**  
  Stop process

- **SIGTSTP**  
  Suspend process

- **SIGTTIN**  
  tty input for background process
CHAPTER 16. BUILTIN FUNCTIONS

SIGTTOU       tty output for background process
SIGXCPU       Out of CPU
SIGXFSZ       File size limit exceeded
SIGPROF       Profile trap
SIGWINCH      Window change signal

Note that you have to use signame to translate the name of a signal to its number.

See Also
signal, signum, signame and fork

load_module - load a binary module

int load_module(string module_name);

Description
This function loads a module written in C or some other language into Pike. The module is initialized and any programs or constants defined will immediately be available.

When a module is loaded the functions init_module_efuns and init_module_programs are called to initialize it. When Pike exits exit_module is called in all dynamically loaded modules. These functions _must_ be available in the module.

Please see the source and any examples available at ftp://www.idonex.se/pub/pike for more information on how to write modules for Pike in C.

Bugs
Please use "./name.so" instead of just "foo.so" for the module name. If you use just "foo.so" the module will not be found.

localtime - break down time() into intelligible components

mapping(string:int) localtime(int time);

Description
Given a time represented as second since 1970, as returned by the function time(), this function returns a mapping with the following components:

{(  "sec" : int(0..59)   seconds over the minute  
  "min" : int(0..59)   minutes over the hour 
  "hour" : int(0..59)  what hour in the day 
  "mday" : int(1..31)  day of the month 
  "mon" : int(0..11)  what month 
  "year" : int(0..)   years since 1900 
  "wday" : int(0..6)  day of week (0=Sunday) 
  "yday" : int(0..365) day of year 
  "isdst" : int(0..1) is daylight saving time 
  "timezone" : int difference between local time and UTC
)}
Note
The ’timezone’ might not be available on all platforms.

See Also
Calendar, gmtime, time, ctime and mktime

log - natural logarithm

float log(float f);

Description
Return the natural logarithm of f.

See Also
pow and exp

lower_case - convert a string to lower case

string lower_case(string s);

Description
Returns a string with all capital letters converted to lower case.

See Also
upper_case

map - map a function over elements

array map(array arr,function fun,mixed ...extra);
mixed map(mixed arr,void|mixed fun,void|mixed ...extra);

Description
simple use: map() loops over all elements in arr and call the function fun with
the element as first argument, with all ”extra” arguments following. The result
is the same datatype as ”arr”, but all elements is the result from the function
call of the corresponding element.

advanced use is a wide combination of types given as ”arr” or ”fun”.

<table>
<thead>
<tr>
<th>arr</th>
<th>fun</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>function</td>
<td>array ret; ret[i]=fun(arr[i],@extra);</td>
</tr>
<tr>
<td></td>
<td>program</td>
<td></td>
</tr>
<tr>
<td></td>
<td>object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>array</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>multiset</td>
<td>ret = rows(fun,arr)</td>
</tr>
<tr>
<td></td>
<td>mapping</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>string</td>
<td>array ret; ret[i]=arr[i]<a href="@extra">fun</a>;</td>
</tr>
<tr>
<td>array</td>
<td>void—int(0)</td>
<td>array ret; ret=arr[@extra]</td>
</tr>
<tr>
<td>mapping</td>
<td>*</td>
<td>mapping ret = mkmapping(indices(arr),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>map(values(arr),fun,@extra));</td>
</tr>
<tr>
<td>multiset</td>
<td>*</td>
<td>multiset ret = (multiset)(map(indices(arr),fun,@extra));</td>
</tr>
<tr>
<td>string</td>
<td>*</td>
<td>string ret = (string)map((array)arr,fun,@extra);</td>
</tr>
<tr>
<td>arr</td>
<td>fun</td>
<td>result</td>
</tr>
<tr>
<td>-------------</td>
<td>-----</td>
<td>------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| object *    |     | if arr-\&\& sizeof && arr-\&\&"[]
|             |     | --array ret; ret[i]=arr[i];                                            |
|             |     | --ret=map(ret,fun,@extra);                                             |

Returns
the same datatype as given, but with the subtype set to the return value of the function; the exception are program and function that gives a mapping back

See Also
filter, enumerate and foreach

Note
You may get unexpected errors if you feed the function with illegal values; for instance if fun is an array of non-callables.

\texttt{m\_delete} - remove an index from a mapping

\begin{verbatim}
mixed m_delete(mapping map, mixed index);
\end{verbatim}

Description
Removes the entry with index \texttt{index} from mapping \texttt{map} destructively. If the mapping does not have an entry with index \texttt{index}, nothing is done. Note that \texttt{m\_delete} changes map destructively and only returns the mapping for compatibility reasons.

This function returns the value from the key-value pair that was removed from the mapping.

See Also
\texttt{mappingp}

\texttt{mappingp} - is the argument a mapping?

\begin{verbatim}
int mappingp(mixed arg);
\end{verbatim}

Description
Returns 1 if \texttt{arg} is a mapping, zero otherwise.

See Also
\texttt{intp, programp, arrayp, stringp, objectp, multisetp, floatp and functionp}

\texttt{master} - return the master object

\begin{verbatim}
object master();
\end{verbatim}

Description
Master is added by the master object to make it easier to access it.
**max** - return the greatest value

mixed max(mixed ...arg)

*Description*

Returns the greatest value of its args.

*Example*

```c
> man( 1,2,3 );
Result: 3
```

*See Also*

min

**min** - return the smallest value

mixed min(mixed ...arg)

*Description*

Returns the smallest value of its args.

*Example*

```c
> min( 1,2,3 );
Result: 1
```

*See Also*

max

**mkdir** - make directory

int mkdir(string dirname, void|int mode);

*Description*

Create a directory, return zero if it fails and nonzero if it successful. If a mode is given, it's used for the new directory after being &'ed with the current umask (on OS'es that supports this).

*See Also*

rm, cd and Stdio.mkdirhier

**mkmapping** - make a mapping from two arrays

mapping mkmapping(array ind, array val);

*Description*

Makes a mapping ind[x]:val[x], 0¡=x¡sizeof(ind). *ind* and *val* must have the same size. This is the inverse operation of *indices* and *values.*
CHAPTER 16. BUILTIN FUNCTIONS

See Also
indices and values

`mkmultiset` - make a multiset

```
multiset mkmultiset(array a);
```

Description
This function creates a multiset from an array.

Example

```
> mkmultiset( {{1,2,3}} );
Result: ( < /* 3 elements */
   1,
   2,
   3
> )
```

See Also
`aggregate_multiset`

`mktime` - convert date and time to seconds

```
int mktime(mapping tm)
int mktime(int sec, int min, int hour, int mday, int mon, int year, int isdst, int tz)
```

Description
This function converts information about date and time into an integer which contains the number of seconds since the beginning of 1970. You can either call this function with a mapping containing the following elements:

- `year`: The number of years since 1900
- `mon`: The month
- `mday`: The day of the month.
- `hour`: The number of hours past midnight
- `min`: The number of minutes after the hour
- `sec`: The number of seconds after the minute
- `isdst`: If this is 1, daylight savings time is assumed
- `tm`: The timezone (-12 ¡= tz ¡= 12)

Or you can just send them all on one line as the second syntax suggests.

See Also
`time`, `ctime`, `localtime` and `gmtime`

`multisetp` - is the argument a multiset?

```
int multisetp(mixed arg);
```
Description
Returns 1 if arg is a multiset, zero otherwise.

See Also
intp, programp, arrayp, stringp, objectp, mappingp, floatp and functionp

mv - move a file (may handle directories as well)

int mv(string from, string to);

Description
Rename or move a file between directories. If the destination file already exists, it will be overwritten. Returns 1 on success, 0 otherwise.

See Also
rm

new - clone an object from a program

object new(program p, mixed ... args);

Description
new() creates an object from the program p. Or in C++ terms: It creates an instance of the class p. This clone will first have all global variables initialized, and then create() will be called with args as arguments.

See Also
clone, destruct, compile_string and compile_file

next object - get next object

object next_object(object o);
object next_object();

Description
All objects are stored in a linked list, next_object() returns the first object in this list, and next_object(o) the next object in the list after o.

Example

/* This example calls shutting_down() in all cloned objects */
object o;
for(o=next_object(); o; o=next_object(o))
o->shutting_down();

See Also
clone and destruct

Note
This function is not recommended to use.

**object_program** - get the program associated with the object

```c
program object_program(object o);
```

**Description**
This function returns the program from which \( o \) was cloned. If \( o \) is not an object or has been destructed \( o \) zero is returned.

**See Also**
clone and new

**object_variablep** - find out if an object identifier is a variable

```c
int object_variablep(object o, string var);
```

**Description**
This function return 1 if \( \text{var} \) exists is a non-static variable in \( o \), zero otherwise.

**See Also**
indices and values

**objectp** - the argument an object?

```c
int objectp(mixed arg);
```

**Description**
Returns 1 if \( \text{arg} \) is an object, zero otherwise.

**See Also**
intp, programp, floatp, stringp, arrayp, mappingp, multisetp and functionp

**pow** - raise a number to the power of another

```c
float pow(float|int n, float|int x);
```

**Description**
Return \( n \) raised to the power of \( x \).

**See Also**
exp and log

**programp** - is the argument a program?

```c
int programp(mixed arg);
```

**Description**
Returns 1 if \( \text{arg} \) is a program, zero otherwise.

**See Also**
intp, multisetp, arrayp, stringp, objectp, mappingp, floatp and functionp
**putenv** - put environment variable

```c
void putenv(string varname, string value);
```

**Description**
This function sets the environment variable `varname` to `value`.

**See Also**
`getenv` and `exece`

**query_host_name** - return the name of the host we are running on

```c
string query_host_name();
```

**Description**
This function returns the name of the machine the interpreter is running on. This is the same thing that the command `hostname` prints.

**query_num_arg** - find out how many arguments were given

```c
int query_num_arg();
```

**Description**
`query_num_arg` returns the number of arguments given when this function was called. This is only useful for varargs functions.

**See Also**
`call_function`

**random** - return a random number

```c
int random(int max);
```

**Description**
This function returns a random number in the range 0 - max-1.

**See Also**
`random_seed`

**random_seed** - seed random generator

```c
void random_seed(int seed);
```

**Description**
This function sets the initial value for the random generator.

**Example**
Pike v1.0E-13 Running Hilfe v1.2 (Hubbe’s Incremental Pike Front-End)
> random_seed(17);
Result: 0
> random(1000);
Result: 732
> random(1000);
Result: 178
> random(1000);
Result: 94
> random_seed(17);
Result: 0
> random(1000);
Result: 732
> random(1000);
Result: 178
> random(1000);
Result: 94
>
See Also
random

**remove_call_out** - remove a call out from the call out queue

```c
int remove_call_out(function f);
int remove_call_out(function id);
```

*Description*
This function finds the first call to the function *f* in the call out queue and removes it. The time left to that call out will be returned. If no call out was found, `zero_type(remove_call_out(f))` will return 1. You can also give a call out id as argument. (as returned by call_out)

See Also
call_out_info, call_out and find_call_out

**remove_include_path** - remove a directory to search for include files

```c
void remove_include_path(string path);
```

*Description*
This function removes a directory from the list of directories to search for include files. It is the opposite of add_include_path.

See Also
add_include_path and #include

**remove_module_path** - remove a directory to search for modules

```c
void remove_module_path(string path);
```

*Description*
This function removes a directory from the list of directories to search for modules. It is the opposite of add_module_path. For more information about modules, see chapter 7.4.

See Also
add_module_path
**remove_program_path** - remove a directory to search for modules

```c
void remove_program_path(string path);
```

**Description**
This function removes a directory from the list of directories to search for program. It is the opposite of `add_program_path`. For more information about programs, see section 4.2.3.

**See Also**
`add_program_path`

**replace** - generic replace function

```c
string replace(string s, string from, string to);
string replace(string s, array(string) from, array(string) to);
array replace(array a, mixed from, mixed to);
mapping replace(mapping a, mixed from, mixed to);
```

**Description**
This function can do several kinds replacement operations, the different syntaxes do different things as follow:

---

**replace_master** - replace the master object

```c
void replace_master(object o);
```

**Description**
This function replaces the master object with the argument you specify. This will let you control many aspects of how Pike works, but beware that master.pike may be required to fill certain functions, so it is probably a good idea to have your master inherit the original master and only re-define certain functions.

**reverse** - reverse a string, array or int

```c
string reverse(string s);
array reverse(array a);
int reverse(int i);
```

**Description**
This function reverses a string, char by char, an array, value by value or an int,
bit by bit and returns the result. Reversing strings can be particularly useful for parsing difficult syntaxes which require scanning backwards.

See Also

`sscanf`

`rint` - round a number

```plaintext
float rint(float f);
```

Description

This function is named `round` in Pike.

Note

This function does not exist! Use `round`!

See Also

`round`

`rm` - remove file or directory

```plaintext
int rm(string f);
```

Description

Remove a file or directory, return 0 if it fails. Nonzero otherwise.

See Also

`mkdir` and `recursive_rm`

`round` - round a number

```plaintext
float round(float f);
```

Description

Return the closest integer value to `f`.

Note

`round()` does not return an int, merely an integer value stored in a float.

See Also

`floor` and `ceil`

`rows` - select a set of rows from an array

```plaintext
array rows(mixed data, array index);
```

Description

This function is exactly equivalent to:

```plaintext
map_array(index,lambda(mixed x,mixed y) – return y[x]; ”,data)
```

Except of course it is a lot shorter and faster. That is, it indices data on every index in the array index and returns an array with the results.

See Also

`column`
**rusage** - return resource usage

```php
array(int) rusage();
```

**Description**
This function returns an array of ints describing how much resources the interpreter process has used so far. This array will have at least 29 elements, of which those values not available on this system will be zero. The elements are as follows:


Don’t ask me to explain these values, read your system manuals for more information. (Note that all values may not be present though)

**See Also**
time

**search** - search for a value in a string, array or mapping

```php
int search(string haystack, string needle, [ int start ]);  
int search(array haystack, mixed needle, [ int start ]);  
mixed search(mapping haystack, mixed needle, [ mixed start ]);  
```

**Description**
Search for `needle` in `haystack`. Return the position of `needle` in `haystack` or -1 if not found. If the optional argument `start` is present search is started at this position. Note that when `haystack` is a string `needle` must be a string, and the first occurrence of this string is returned. However, when `haystack` is an array, `needle` is compared only to one value at a time in `haystack`.

When the `haystack` is a mapping, `search` tries to find the index connected to the data `needle`. That is, it tries to lookup the mapping backwards. If `needle` isn’t present in the mapping, zero is returned, and zero_type() will return 1 for this zero.

**See Also**
indices, values and zero_type

**sgn** - check the sign of a value

```php
int sgn(mixed value);  
int sgn(mixed value, mixed base);  
```

**Description**
This function returns -1 if `value` is less than zero, 1 if `value` is greater than zero and 0 otherwise. If `base` is given, -1 is returned if `value` is lesser than `base`, 1 if `value` is greater than `base` and 0 otherwise.
CHAPTER 16. BUILTIN FUNCTIONS

See Also
abs

signal - trap signals

void signal(int sig, function(int:void) callback);
void signal(int sig);

Description
This function allows you to trap a signal and have a function called when the
process receives a signal. Although it IS possible to trap SIGBUS, SIGSEGV
etc. I advice you not to. Pike should not receive any such signals and if it does
it is because of bugs in the Pike interpreter. And all bugs should be reported,
no matter how trifle.

The callback will receive the signal number as the only argument. See the
document for the function 'kill' for a list of signals.

If no second argument is given, the signal handler for that signal is restored to
the default handler.

If the second argument is zero, the signal will be completely ignored.

See Also
kill, signame and signum

signame - get the name of a signal

string signame(int sig);

Description
Returns a string describing the signal.

Example
> signame(9);
Result: SIGKILL

See Also
kill, signame and signal

signum - get a signal number given a descriptive string

int signum(string sig);

Description
This function is the opposite of signame.

Example
> signum("SIGKILL");
Result: 9

See Also
signame, kill and signal
\begin{verbatim}
sin - trigonometrical sine
float sin(float f);

Description
Returns the sinus value for f.

See Also
asin and cos

sizeof - return the size of an array, string, multiset or mapping
int sizeof(string|multiset|mapping|array|object a);

Description
This function returns the number of indices available in the argument given to it. It replaces older functions like strlen, m_sizeof and size.

sleep - let interpreter doze off for a while
void sleep(float|int s, int|void foo);

Description
This function makes the program stop for s seconds. Only signal handlers can interrupt the sleep. Other callbacks are not called during sleep.

See Also
signal

sort - sort an array destructively
array sort(array(mixed) index, array(mixed) ... data);

Description
This function sorts the array 'index' destructively. That means that the array itself is changed and returned, no copy is created. If extra arguments are given, they are supposed to be arrays of the same size. Each of these arrays will be modified in the same way as 'index'. I.e. if index 3 is moved to position 0 in 'index' index 3 will be moved to position 0 in all the other arrays as well.

Sort can sort strings, integers and floats in ascending order. Arrays will be sorted first on the first element of each array.

Sort returns its first argument.

See Also
reverse

sprintf - print the result from sprintf
string sprintf(string format, mixed arg, ...);  

Description
The format string is a string containing a description of how to output the data in the rest of the arguments. This string should generally speaking have
\end{verbatim}
one `%i modifiers` operator (examples: `%s`, `%0d`, `%-=20s`) for each of the rest arguments.

Modifiers:

- 0: Zero pad numbers (implies right justification)
- !: Toggle truncation
- (space): pad positive integers with a space
- +: pad positive integers with a plus sign
- -: left adjusted within field size (default is right)
- =: centered within field size
- /: column mode if strings are greater than field size
- #: table mode, print a list of ‘‘n’’ separated word (top-to-bottom order)
- $: Inverse table mode (left-to-right order)
- n: (where n is a number or *) a number specifies field size
- .n: set precision
- :n: set field size & precision
- :n: Set column width
- *: if n is a * then next argument is used for precision/field size
- 'X': Set a pad string. ’ cannot be a part of the pad string (yet)
- -: Get pad string from argument list.
- i: Use same arg again
- @: repeat this on every line produced
- @: do this format for each entry in argument array
- @: Put the string at the bottom end of column instead of top
- : Set width to the length of data

Operators:

- `%%` percent
- `%b` signed binary int
- `%d` signed decimal int
- `%o` signed octal int
- `%x` lowercase signed hexadecimal int
- `%X` uppercase signed hexadecimal int
- `%c` char (or short with `%2c`, `%3c` gives 3 bytes etc.)
- `%f` float
- `%g` heuristically chosen representation of float
- `%G` like `%g`, but uses uppercase E for exponent
- `%e` exponential notation float
- `%E` like `%e`, but uses uppercase E for exponent
- `%F` binary IEEE representation of float (%4F gives single precision, %8F gives double precision.)
- `%s` string
- `%O` any type (debug style)
- `%n` nop
- `%t` type of argument
- `%i modifiers;` format do a format for every index in an array.
Example

Pike v0.7 release 1 running Hilfe v2.0 (Incremental Pike Frontend)
> int screen_width=70;
 Result: 70
> mixed sample;
> write(sprintf("fish: %c\n", 65));
 fish: A
 Result: 8
> write(sprintf("num: %d\n", 10));
 num: 10
 Result: 8
> write(sprintf("num: %+10d\n", 10));
 num: +10
 Result: 16
> write(sprintf("num: %010d\n", 5*2));
 num: 000000010
 Result: 16
> write(sprintf("num: %10d\n", 20/2));
 num: 10
 Result: 16
> write(sprintf("%s\n", screen_width,"THE NOT END"));
 THE NOT END
 Result: 71
> write(sprintf("%s\n",screen_width, "fun with penguins\n"));
 fun with penguins
 Result: 71
> write(sprintf("%*s\n",screen_width,{{ "fish", 9, "gumbies", 2 }}));
 (/* 4 elements */
 "fish",
 9,
 "gumbies",
 2)
 Result: 426
> write(sprintf("%*s\n", screen_width,
 "This will wordwrap the specified string within the "+
 "specified field size, this is useful say, if you let "+
 "users specify their screen size, then the room "+
 "descriptions will automagically word-wrap as appropriate.\n"
 + "slosh-n's will of course force a new-line when needed.\n"));
This will wordwrap the specified string within the specified field size, this is useful say, if you let users specify their screen size, then the room descriptions will automagically word-wrap as appropriate.
slosh-n's will of course force a new-line when needed.
 Result: 355
> write(sprintf("%*s %*s\n", screen_width/2,
"This will wordwrap the specified string within the "+
 "specified field size, this is useful say, if you let "+
 "users specify their screen size, then the room "+
 "descriptions will automagically word-wrap as appropriate.\n"
 + "slosh-n's will of course force a new-line when needed.\n"));
This will wordwrap the specified string within the specified field size, this is useful say, if you let users specify their screen size, then the room descriptions will automagically word-wrap as appropriate.
slosh-n's will of course force a new-line when needed.
 Result: 355
> write(sprintf("%*s %*s\n", screen_width/2,
 This will wordwrap the specified string within the specified field size, this is useful say, if you let users specify their screen size, then the room descriptions will automagically word-wrap as appropriate.
slosh-n's will of course force a new-line when needed.
 Result: 355
"Two columns next to each other (any number of columns will "+
"of course work) independently word-wrapped, can be useful.",
screen_width/2-1,
"The - is to specify justification, this is in adherence "+
"to std sprintf which defaults to right-justification, "+
"this version also supports center and right justification.");

Two columns next to each other (any number of columns will of course work) independently word-wrapped, can be useful.

Result: 426

> write(sprintf("%-s
", screen_width,
"Given a list of slosh-n separated 'words', this option
creates a table out of them the number of columns be forced
by specifying a precision. The most obvious use is for formatted ls output.");

Given a list of slosh-n separated 'words', this option creates a table out of them the number of columns be forced by specifying a precision. The most obvious use is for formatted ls output.

Result: 312

> sample = ("bing","womble","wuff","gul"));

Result: (4 elements */

"bing",
"womble",
"wuff",
"gul"
}

> write(sprintf("This will apply the format strings between the procent-braces to all elements in the array:
"%{gurksallad: %s
%},
sample));

This will apply the format strings between the procent-braces to all elements in the array:
gurksallad: bing
gurksallad: womble
gurksallad: wuff
gurksallad: gul

Result: 162

> write(sprintf("Of course all the simple printf options "+
"are supported: \%s, \%!s, \%!o, \%!c\n",
"65 as decimal, hex, octal and a char",
65, 65, 65, 65));

Of course all the simple printf options are supported: 65 as decimal, hex, octal and a char: 65 41 101 A

Result: 106
> write(sprintf("%s\n", screen_width, "THE END"));

THE END

Result: 71
> quit
Exiting.

See Also
sscanf

sqrt - square root

float sqrt(float f);
int sqrt(int i);

Description
Returns the square root of f, or in the second case, the square root truncated to the closest lower integer.

See Also
pow, log, exp and floor

strerror - return a string describing an error

string strerror(int errno);

Description
This function returns a description of an error code. The error code is usually obtained from the file-errno() call.

Note
This function may not be available on all platforms.

stringp - is the argument a string?

int stringp(mixed arg);

Description
Returns 1 if arg is a string, zero otherwise.

See Also
intp, multisetp, arrayp, programp, objectp, mappingp, floatp and functionp

string_to_unicode - convert a string to an UTF16 stream

string string_to_unicode(string s);

Description
Converts a string into an UTF16 compliant byte-stream.

Throws an error if characters not legal in an UTF16 stream are encountered. Valid characters are in the range 0x00000 - 0x10ffff, except for characters 0xfffe and 0xffff.
Characters in range 0x010000 - 0x10ffff are encoded using surrogates.

See Also
Locale.Charset.decode, string_to_utf8, unicode_to_string and utf8_to_string

string_to_utf8 - convert a string to an UTF8 stream

string string_to_utf8(string s); string string_to_utf8(string s, int extended);

Description
Converts a string into an UTF8 compilant byte-stream.

Throws an error if characters not valid in an UTF8 stream are encountered. Valid characters are in the range 0x00000000 - 0x7fffffff.

If extended is 1, characters in the range 0x80000000-0xfffffffff will also be accepted, and encoded using a non-standard UTF8 extension.

See Also
Locale.Charset.decode, string_to_unicode, unicode_to_string and utf8_to_string

strlen - return the length of a string

int strlen(string s);

Description
This function is equal to sizeof.

See Also
sizeof

tan - trigonometrical tangent

float tan(float f);

Description
Returns the tangent value for f.

See Also
atan, sin and cos
this_object - return the object we are evaluating in currently

object this_object();

Description
This function returns the object we are currently evaluating in.

throw - throw a value to catch or global error handling

void throw(mixed value);
**Description**  
This function throws a value to a waiting catch. If no catch is waiting global error handling will send the value to handle_error in the master object. If you throw an array with where the first index contains an error message and the second index is a backtrace, (the output from backtrace() that is) then it will be treated exactly like a real error by overlying functions.

**See Also**  
catch

**time** - return the current time

```c
int time();
int time(1);
float time(int t);
```

**Description**  
This function returns the number of seconds since 1 Jan 1970. The function ctime() converts this integer to a readable string.

The second syntax does not call the system call time() as often, but is only updated in the backed. (when Pike code isn’t running)

The third syntax can be used to measure time more precisely than one second. It return how many seconds has passed since \( t \). The precision of this function varies from system to system.

**See Also**  
ctime, localtime, mktime and gmtime

**trace** - change debug trace level

```c
int trace(int t);
```

**Description**  
This function affects the debug trace level. (also set by the -t command line option) The old level is returned. Trace level 1 or higher means that calls to Pike functions are printed to stderr, level 2 or higher means calls to built-in functions are printed, 3 means every opcode interpreted is printed, 4 means arguments to these opcodes are printed as well. See the command lines options for more information

**typeof** - check return type of expression

```c
typeof ( expression );
```

**Description**  
This is a not really a function even if it looks like it, it returns a human readable (almost) representation of the type that the expression would return without actually evaluating it. The representation is in the form of a string.

**Example**  
> typeof('sizeof);
Result: function(object | mapping | array | multiset | string : int)
> typeof(sizeof({}));
Result: int
>

*ualarm* - set an alarm clock for delivery of a signal

```c
int ualarm(int useconds);
```

**Description**

ualarm arranges for a SIGALRM signal to be delivered to the process in useconds micro seconds.

If useconds is zero, no new alarm is scheduled.

In any event any previously set alarm is canceled.

**Returns**

ualarm returns the number of microseconds seconds remaining until any previously scheduled alarm was due to be delivered, or zero if there was no previously scheduled alarm.

**See Also**

signal

*unicode_to_string* - convert an UTF16 stream to a string

```c
string unicode_to_string(string s);
```

**Description**

Converts an UTF16 byte-stream into a string.

**Note**

This function does not decode surrogates.

**See Also**

Locale.Charset.decode, string_to_unicode, string_to_utf8 and utf8_to_string

*upper_case* - convert a string to upper case

```c
string upper_case(string s);
```

**Description**

Returns a copy of the string s with all lower case character converted to upper case character.

**See Also**

lower_case

*utf8_to_string* - convert an UTF8 stream to a string

```c
string utf8_to_string(string s); string utf8_to_string(string s, int extended);
```
Description
Converts an UTF8 byte-stream into a string.
Throws an error if the stream is not a legal UTF8 byte-stream.
Accepts and decodes the extension used by string_to_utf8(), if extended is 1.

See Also
Locale.Charset.decode, string_to_unicode, string_to_utf8 and unicode_to_string

values - return an array of all possible values from indexing
array values(string|multiset|mapping|array|object foo);

Description
Values return an array of all values you can get when indexing the value foo.
For strings, an array of int with the ascii values of the characters in the string
is returned. For a multiset, an array filled with ones is return. For mappings,
objects and arrays, the returned array may contain any kind of value.

See Also
indices

version - return version info
string version();

Description
This function returns a brief information about the Pike version.

Example

> version();
Result: "Pike v0.7 release 1"

write - write text to stdout
int write(string text);

Description
Added by the master, it directly calls write in a Stdio.stdout.

See Also
Stdio.werror

zero_type - return the type of zero
int zero_type(mixed a);

Description
There are many types of zeros out there, or at least there are two. One
is returned by normal functions, and one returned by mapping lookups and
find_call_out() when what you looked for wasn’t there. The only way to separate these two kinds of zeros is zero_type. When doing a find_call_out or mapping lookup, zero_type on this value will return 1 if there was no such thing present in the mapping, or no such call_out could be found. If the argument to zero_type is a destructed object or a function in a destructed object, 2 will be returned. Otherwise zero_type will return zero.

If the argument is not an int, zero will be returned.

See Also
find_call_out
Chapter 17

Pike internals - how to extend Pike

The rest of this book describes how Pike works and how to extend it with your own functions written in C or C++. Even if you are not interested in extending Pike, the information in this section can make you understand Pike better and thus make you a better Pike programmer. From this point on I will assume that the reader knows C or C++.

17.1 The master object

Pike is a very dynamic language. Sometimes that is not enough, sometimes you want to change the way Pike handles errors, loads modules or start scripts. All this and much more can be changed by modifying the master object. The master object is a Pike object like any other object, but it is loaded before anything else and is expected to perform certain things for the Pike executable. The Pike executable cannot function without a master object to take care of these things. Here is a list of the methods needed in the master object:
Aside from the above functions, which are expected from the Pike binary, the master object is also expected to provide functions used by Pike scripts. The current master object adds the following global functions:

There are at least two ways to change the behavior of the master object. (Except for editing it directly, which would cause other Pike scripts not to run in most cases.) You can either copy the master object, modify it and use the command line option \texttt{-m} to load your file instead of the default master object. However, since there might be more functionality added to the master object in the future I do not recommend this.

A better way is to write an object that inherits the master and then calls replace_master with the new object as argument. This should be far more future-safe. Although I can not guarantee that the interface between Pike and the master object will not change in the future, so be careful if you do this.

Let’s look an example:

``` Pike
#!/usr/local/bin/pike

class new_master {
    inherit "/master";

    void create()
    {
        /* You need to copy the values from the old master to the new */
        /* NOTE: At this point we are still using the old master */
        object old_master = master();
        object new_master = this_object();

        foreach(indices(old_master), string varname)
        {
            /* The catch is needed since we can’t assign constants */
            catch { new_master[varname] = old_master[varname]; };
        }
    }
}
```
void handle_error(array trace)
{
    Stdio.write_file("error log", describe_backtrace(trace));
}

int main(int argc, array(string) argv)
{
    replace_master(new master());
    /* Run rest of program */
    exit(0);
}

This example installs a master object which logs run time errors to file instead of writing them to stderr.

17.2 Data types from the inside

This section describes the different data types used inside the Pike interpreter. It is necessary to have at least a basic understanding of these before you write Pike extensions.

17.2.1 Basic data types

First, we must come to know the basic data types pike uses.

17.2.2 struct svalue

An svalue is the most central data structure in the Pike interpreter. It is used to hold values on the stack, local variables, items in arrays and mappings and a lot more. Any of the data types described in chapter 3.4 can be stored in an svalue.
A **struct svalue** has three members:

Of course there are a whole bunch of functions for operating on svalues:

**free_svalue** - free the contents of an svalue

```c
void free_svalue(struct svalue *s);
```

**Description**
This function is actually a macro, it will the contents of `s`. It does not however free `s` itself. After calling `free_svalue`, the contents of `s` is undefined, and you should not be surprised if your computer blows up if you try to access the it’s contents. Also note that this doesn’t nessecarily free whatever the svalue is pointing to, it only frees one reference. If that reference is the last one, the object/array/mapping/whatever will indeed be freed.

**Note**
This function will *not* call Pike code or `error()`.

**free_svalues** - free many svalues

```c
void free_svalues(struct svalue *s, INT32 howmany, TYPE_FIELD type_hint);
```

**Description**
This function does the same as `free_svalue` but operates on several svalues. The `type_hint` is used for optimization and should be set to `BIT_MIXED` if you don’t know exactly what types are beeing freed.

**Note**
This function will *not* call Pike code or `error()`.

**See Also**
`free_svalue` and `TYPE_FIELD`

**assign_svalue** - copy an svalue to another svalue

```c
void assign_svalue(struct svalue *to, sstruct svalue *from);
```

**Description**
This function frees the contents of `to` and then copies the contents of `from` into `to`. If the value in `from` uses refcounts, they will be increased to reflect this copy.

**Note**
This function will *not* call Pike code or `error()`.
See Also
free_svalue and assign_svalue_no_free

assign_svalue_no_free - copy an svalue to another svalue

void assign_svalue_no_free(struct svalue *to, struct svalue *from);

Description
This function does the same as assign_svalue() but does not free the contents of to before overwriting it. This should be used when to has not been initialized yet. If this function is incorrectly, memory leaks will occur. On the other hand, if you call assign_svalue on an uninitialized svalue, a core dump or bus error will most likely occur.

Note
This function will *not* call Pike code or error().

See Also
assign_svalue and free_svalue

IS_ZERO - check if an svalue is true or false

int IS_ZERO(struct svalue *s);

Description
This macro returns 1 if s is false and 0 if s is true.

Note
This macro will evaluate s several times.
This macro may call Pike code and/or error().

See Also
is_eq

is_eq - check if two svalues contains the same value

int is_eq(struct svalue *a, struct svalue *b);

Description
This function returns 1 if a and b contain the same value. This is the same as the ‘==’ operator in pike.

Note
This function may call Pike code and/or error().

See Also
IS_ZERO, is_lt, is_gt, is_le, is_ge and is_equal

is_equal - check if two svalues are equal

int is_equal(struct svalue *a, struct svalue *b);

Description
This function returns 1 if a and b contains equal values. This is the same as the function equal in pike.
Note
This function may call Pike code and/or error().

See Also
equal and is_eq

is_lt - compare the contents of two svalues

int is_lt(struct svalue *a, struct svalue *b);
int is_le(struct svalue *a, struct svalue *b);
int is_gt(struct svalue *a, struct svalue *b);
int is_ge(struct svalue *a, struct svalue *b);

Description
These functions are equal to the pike operators ‘<’, ‘<=’, ‘>’, ‘>=’ respectively. For instance is_lt will return 1 if the contents of a is lesser than the contents of b.

Note
This function may call Pike code and/or error(). For instance, it will call error() if you try to compare values which cannot be compared such as comparing an integer to an array.

See Also
IS_ZERO and is_eq

17.2.3 struct pike_string

A struct pike_string is the internal representation of a string. Since Pike relies heavily on string manipulation, there are quite a few features and quirks to using this data structure. The most important part is that strings are shared. This means that after a string has been entered into the shared string table it must never be modified. Since some other thread might be using the very same string, it is not even permitted to change a shared string temporarily and then change it back.

A struct pike_string has these members:
General string management
Since pike strings are shared, you can compare them by using `==`. FIXME – add more here.

`STR0` - Get a pointer to a `char`

```c
p wchar0 *STR0(struct pike_string *s);
p wchar1 *STR1(struct pike_string *s);
p wchar2 *STR2(struct pike_string *s);
```

**Description**
These macros return raw C pointers to the data in the string `s`. Note that you may only use `STR0` on strings where `size_shift` is 0, `STR1` on strings where `size_shift` is 1 and `STR2` on strings where `size_shift` is 2. When compiled with `DEBUG` these macros will call `fatal` if used on strings with the wrong `size_shift`.

**Note**
All pike strings have been zero-terminated for your convenience.
The zero-termination is not included in the length of the string.

`free_string` - Free a reference to a pike_string

```c
void free_string(struct pike_string *s);
```

**Description**
This function frees one reference to a pike string and if that is the last reference, it will free the string itself. As with all refcounting functions you should be careful about how you use it. If you forget to call this when you should, a memory leak will occur. If you call this function when you shouldn’t Pike will most likely crash.

`make_shared_string` - Make a new shared string

```c
struct pike_string *make_shared_string(char *str);
```

**Description**
This function takes a null terminated C string as argument and returns a `pike_string` with the same contents. It does not free or change `str`. The returned string will have a reference which will be up to you to free with `free_string` unless you send the string to a function such as `push_string` which eats the reference for you.

**See Also**
`free_string`, `push_string`, `begin_shared_string`, `make_shared_binary_string`, `make_shared_string1` and `make_shared_string2`

`make_shared_binary_string` - Make a new binary shared string

```c
struct pike_string *make_shared_binary_string(char *str, INT32 len);
```
CHAPTER 17. PIKE INTERNALS - HOW TO EXTEND PIKE

Description
This function does essentially the same thing as make_shared_string, but you give it the length of the string str as a second argument. This allows for strings with zeros in them. It is also more efficient to call this routine if you already know the length of the string str.

See Also
free_string, push_string, begin_shared_string, make_shared_string, make_shared_binary_string and make_shared_binary_string2

begin_shared_string - Start building a shared string

struct pike_string *begin_shared_string(INT32 len);

Description
This function is used to allocate a new shared string with a specified length which has not been created yet. The returned string is not yet shared and should be initialized with data before calling end_shared_string on it.

If after calling this function you decide that you do not need this string after all, you can simply call free on the returned string to free it. It is also possible to call free_string(end_shared_string(s)) but that would be much less efficient.

Example

// This is in effect equal to s=make_shared_string("test")
struct pike_string *s=begin_shared_string(4);
STRO(s)[0] = 't';
STRO(s)[1] = 'e';
STRO(s)[2] = 's';
STRO(s)[3] = 't';
s=end_shared_string(s);

See Also
begin_shared_string, free_string, push_string, make_shared_string and end_shared_string

end_shared_string - Insert a pre-allocated string into the shared string table

struct pike_string *end_shared_string(struct pike_string *s);

Description
This function is used to finish constructing a pike string previously allocated with begin_shared_string or begin-wide_shared_string. It will insert the string into the shared string table. If there already is such a string in the shared string table then s will be freed and that string will be returned instead. After calling this function, you may not modify the string any more. As with make_shared_string this function returns a string with a reference which it is your responsibility to free.

See Also
begin_shared_string and begin-wide_shared_string

begin-wide_shared_string - Start building a wide shared string

struct pike_string *begin-wide_shared_string(INT32 len, int size_shift);
Description
This function is a more generic version of \texttt{begin\_shared\_string}. It allocates space for a string of length \textit{len} where each character is $1 \ll \text{size\_shift}$ bytes. As with \texttt{begin\_shared\_string} it is your responsibility to initialize the string and to call \texttt{end\_shared\_string} on it.

Example
\begin{verbatim}
struct pike\_string *s=begin\_wide\_shared\_string(1,2); STR2(s)[0]=4711;
s=end\_shared\_string(s);
\end{verbatim}

See Also
\texttt{begin\_shared\_string}, \texttt{end\_shared\_string}, \texttt{make\_shared\_string}, \texttt{make\_shared\_string1} and \texttt{make\_shared\_string2}

\textit{make\_shared\_string1} - Make a wide shared string
\begin{verbatim}
struct pike\_string *make\_shared\_string1(p\_wchar1 *str);
struct pike\_string *make\_shared\_binary\_string1(p\_wchar1 *str, \text{INT32} len);
struct pike\_string *make\_shared\_string2(p\_wchar2 *str);
struct pike\_string *make\_shared\_binary\_string2(p\_wchar2 *str, \text{INT32} len);
\end{verbatim}

Description
These functions are the wide string equivalents of \texttt{make\_shared\_string} and \texttt{make\_shared\_binary\_string}. The functions ending in 1 use 2-byte characters and the ones ending in 2 use 4-byte characters.

See Also
\texttt{make\_shared\_string}, \texttt{make\_shared\_binary\_string} and \texttt{begin\_wide\_shared\_string}

17.2.4 \textit{struct array}

Internally Pike uses a \texttt{struct array} to represent the type \texttt{array}. As with strings, arrays are used in many different ways, so they have many supporting functions for making them easier to manipulate. Usually you will not have to construct array structures yourself, but it is often necessary to read data from arrays.

A \texttt{struct array} has these members:
Here is an example function which will print the type of each value in an array:

```c
void prtypes(struct array *a)
{
  INT e;
  for(e=0; e<a->size; e++)
    printf("Element %d is of type %d\n", e, a->item[e].type);
}
```

allocate_array
free_array
array_index
array_index_no_free
simple_array_index_no_free
array_set_index
push_array_items
aggregate_array
f_aggregate_array
append_array
explode
slice_array
add_arrays
copy_array

17.2.5 \textit{struct mapping}

\textit{struct mapping} is used to represent a mapping, for the most part you should be able to write modules and understand Pike internals without actually touching the internals of a mapping. It also helps that mappings are very well abstracted, so you can almost always use the supporting functions instead of fiddling around with the contents of a \textit{struct mapping} directly.

This is the contents of a \textit{struct mapping}:
Mappings are allocated as two separate blocks of memory. One is the `struct mapping` which holds pointers into the second memory block. The second memory block contains the hash table and all key-value pairs. A key-value pair is represented as a `struct keypair` which has the following members:

Please note that the free list is separate for each mapping. Also, when there are no more free key-value pairs the whole memory block is re-allocated and the mapping is re-hashed with a larger hash table.

Below is an illustration which shows an example of a small mapping with hash table, free list and key-index pairs.
As you can see, mappings uses a linked list for each bucket in the hash table. Also, the current implementation moves key-value pairs to the top of the hash chain everytime a match is found, this can greatly increase performance in some situations. However, because of this the order of elements in a mapping can change every time you access it. Also, since mappings can be re-allocated any time you add an element to it you can never trust a pointer to a pointer to a `struct keypair` if any Pike code has a chance to execute.

- `m_sizeof`
- `m_ind_types`
- `m_val_types`
- `MAPPING_LOOP`
- `free_mapping`
- `allocate_mapping`
- `mapping_insert`
- `mapping_get_item_ptr`
17.3. THE INTERPRETER

map_delete
---
low_mapping_lookup
---
low_mapping_string_lookup
---
simple_mapping_string_lookup
---
mapping_string_insert
---
mapping_indices
---
mapping_values
---
mapping_to_array
---
mapping_replace
---
mkmapping
---
copy_mapping
---

17.2.6 struct object

17.2.7 struct program

17.3 The interpreter

•
•
•

Functional overview  Overview of the Pike source
•
•
•
•
•
•
•
Appendix A

Terms and jargon
Appendix B

Register program

Here is a complete listing of the example program from chapter 2.

```pike
#!/usr/local/bin/pike

mapping records(string:array(string)) = ([
  "Star Wars Trilogy" : ({
    "Fox Fanfare",
    "Main Title",
    "Princess Leia's Theme",
    "Here They Come",
    "The Asteroid Field",
    "Yoda's Theme",
    "The Imperial March",
    "Parade of the 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=Stdio.Readline()->read("Record name: ");
    records[record_name]=({});
    write("Input song names, one per line. End with '.' on its own line.\n");
    while(1)
    {
        string song;
        song=Stdio.Readline()->read(sprintf("Song %2d: ",
                                             sizeof(records[record_name])+1));

        if(song==".") return;
        records[record_name]+=({song});
    }
}

void save(string file_name)
{
    string name, song;
    Stdio.File o=Stdio.File();

    if(!o->open(file_name,"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_name)
{
string name="ERROR";
string file_contents,line;

Stdio.File o=Stdio.File();
if(!o->open(file_name,"r"))
{
    write("Failed to open file.\n");
    return;
}

file_contents=o->read();
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, array(string) argv)
{
    string cmd;
    while(cmd=Stdio.Readline()->read("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);
            break;

        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 B. REGISTER PROGRAM
Appendix C

Reserved words

These are words that have special meaning in Pike and can not be used as variable or function names.

array break case catch continue default do else float for foreach function gauge if inherit inline int lambda mapping mixed multiset nomask object predef private program protected public return sscanf static string switch typeof varargs void while
Appendix D

BNF for Pike

BNF is short for "Backus Naur Form". It is a precise way of describing syntax.
This is the BNF for Pike:

```plaintext
program ::= definition
definition ::= import — inheritance — function_declaration — function_definition — variables — constant — class_def
import ::= modifiers import ( constant_identifier — string ) ;
inheritance ::= modifiers inherit program_specifier [ ; ; identifier ] ;
function_declaration ::= modifiers type identifier ( arguments ) ;
function_definition ::= modifiers type identifier ( arguments ) block
variables ::= modifiers type variable_names ;
variable_names ::= variable_name — , , variable_name
variable_name ::= * identifier [ = expression2 ]
constant ::= modifiers constant constant_names ;
constant_names ::= constant_name — , , constant_name
constant_name ::= identifier = expression2
class_def ::= modifiers class [ ; ; ]
class ::= class [ identifier ] ; ; program ; ;
modifiers ::= static — private — nomask — public — protected — inline
block ::= statement
statement ::= expression2 ; — cond — while — do_while — for — switch — case — default — return — block — foreach — break — continue ; ;
cond ::= if statement [ else statement ]
while ::= while ( expression ) statement
do_while ::= do statement while ( expression ) ; ;
for ::= for ( [ expression ] ; ; [ expression ] ; ; [ expression ] ) statement
switch ::= switch ( expression ) block
case ::= case expression [ ; ; expression ] ; ;
default ::= default ; ;
foreach ::= foreach ( expression ; ; expression6 ) statement
break ::= break ; ;
continue ::= continue ; ;
```

413
414

APPENDIX D. BNF FOR PIKE

expression

::=

expression2

::=

expression3

::=

expression4

::=

expression5

::=

expression6

::=

number
float
catch
gauge

::=
::=
::=
::=

sscanf

::=

lvalue

::=

lambda
constant identifier
call
index
array
multiset

::=
::=
::=
::=
::=
::=

mapping

::=

arrow
parenthesis
expression list
splice expression

::=
::=
::=
::=

type

::=

function type
arguments
argument
program specifier
string

::=
::=
::=
::=
::=

identifier

::=

letter
digit

::=
::=

expression2 – ”,” expression2 ˝
– lvalue ( ”=” — ”+=” — ”*=” — ”/=” — ”&=”
— ”—=” — ”ˆ=” — ”¡¡=” — ”¿¿=” — ”%=” ) ˝
expression3
expression4 ’ ?’ expression3 ”:” expression3
– expression5 ( ”——” — ”&&” — ”—” — ”ˆ” — ”&”
— ”==” — ”!=” — ”¿” — ”¡” — ”¿=” — ”¡=” — ”¡¡”
— ”¿¿” — ”+” — ”*” — ”/” — ”%” ) ˝ expression5
expression6 — ”(” type ”)” expression5 — ”–” expression6 — ”++” expression6 — expression6 ”–” — expression6 ”++” — ”˜” expression5 — ”-” expression5
string — number — float — catch — gauge — typeof
— sscanf — lambda — class — constant identifier —
call — index — mapping — multiset — array — parenthesis — arrow
digit – digit ˝ — ”0x” – digits ˝ — ”’” character ”’”
digit – digit ˝ ”.” – digit ˝
catch ( ”(” expression ”)” — block )
gauge ( ”(” expression ”)” — block )
sscanf ”(” expression2 ”,” expression2 – ”,” lvalue ˝
”)”
expression6 — type identifier — ”[” [ lvalue – ”,” lvalue
˝ [ ”,” ] ] ”]”
lambda ”(” arguments ”)” block
[”.”] identifier – ”.” identifier ˝
expression6 ”(” expression list ”)”
expression6 ”[” expression [ ”..” expression ] ”]”
”(–” expression list ”˝)”
”(¡” expression list ”¿)”
”([” [ expression : expression – ”,” expression ”:” expression ˝ ] [ ”,” ] ”])”
expression6 ”-¿” identifier
”(” expression ”)”
[ splice expression – ”,” splice expression ˝ ] [ ”,” ]
[ ”@” ] expression2
( int — string — float — program — object [ ”(”
program specifier ”)” ] — mapping [ ”(” type ”:” type
”)” — array [ ”(” type ”)” ] — multiset [ ”(” type
”)” ] — function [ function type ] ) – ”*” ˝
”(” [ type – ”,” type ˝ [ ”...” ] ”)”
[ argument – ”,” argument ˝ ] [”,”]
type [ ”...” ] [ identifier ]
string constant — constant identifier
string literal – string literal ˝
letter – letter — digit ˝ — ”‘+” — ”‘/” — ”‘%” —
”‘*” — ”‘&” — ”‘—” — ”‘ˆ” — ”‘˜” — ”‘¡” — ”‘¡¡”
— ”‘¡=” — ”‘¿” — ”‘¿¿” — ”‘¿=” — ”‘==” — ”‘!=”
— ”‘!” — ”‘()” — ”‘-” — ”‘-¿” — ”‘-¿=” — ”‘[]” —
”‘[]=”
”a”-”z” — ”A”-”Z” — ” ”
”0”-”9”


Appendix E

How to install Pike

To install Pike, you need a C compiler, a couple of Mb of disk space, the source for Pike, and a bit of patience. The latest version of Pike is always available from the Pike home page*. Lists of mirror sites and binary releases should also be available there. Pike should compile and install nicely on almost any UNIX platform. It has been tested on the following:

- 
- 
- 
- 
- 
- 
- 
- 
- 
- 

After obtaining the Pike source you need to unpack it. To unpack Pike you need gzip, which is available from any GNU mirror site. You also need tar, which is a part of UNIX. If you got Pike-v0.7.1.tar.gz, simply unpack it by typing:

```
$ gunzip -d Pike-v0.7.1.tar.gz
$ tar xvf Pike-v0.7.1.tar
```

Now you have a directory called Pike-v0.7.1. Please read the README file in the new directory since newer versions can contain information not available at the time this book was written.

Now, to compile Pike, the following three commands should be enough.
$ cd Pike-v0.7.1/src
$ ./configure --prefix=/dir/to/install/pike
$ make

They will (in order) change directory to the source directory. Configure will then find out what features are available on your system and construct makefiles. You will see a lot of output after you run configure. Do not worry, that is normal. It is usually not a good idea to install Pike anywhere but in /usr/local (the default) since Pike scripts written by other people will usually assume that's where Pike is. However, if you do not have write access to /usr/local you will have to install Pike somewhere else on your system.

After that make will actually compile the program. After compilation it is a good idea to do make verify to make sure that Pike is 100% compatible with your system. Make verify will take a while to run and use a lot of CPU, but it is worth it to see that your compilation was successful. After doing that you should run make install to install the Pike binaries, libraries and include files in the directory you selected earlier.

You are now ready to use Pike.
Appendix F

How to convert from old versions of Pike

This appendix contains some information about keywords, functions and old behavior. But there are only information about migration from old an old Pike to a new one, not the other way around.
APPENDIX F. HOW TO CONVERT FROM OLD VERSIONS OF PIKE
Appendix G

Image Layer modes

All channels are calculated separately, if nothing else is specified.

\[ D = \frac{L \cdot a_L + S \cdot (1 - a_L) \cdot a_S}{a_L + (1 - a_L) \cdot a_S} \]

\[ a_D = (a_L + (1 - a_L) \cdot a_S) \]
### APPENDIX G. IMAGE LAYER MODES

<table>
<thead>
<tr>
<th>Mode</th>
<th>Formula</th>
<th>Alpha Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add</strong></td>
<td>D = L + S</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>Subtract</strong></td>
<td>D = L - S</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>Multiply</strong></td>
<td>D = L * S</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>Divide</strong></td>
<td>D = L / S</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>Modulo</strong></td>
<td>D = L % S</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>InvSubtract</strong></td>
<td>D = S - L</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>InvDivide</strong></td>
<td>D = S / L</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>InvModulo</strong></td>
<td>D = S % L</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>Difference</strong></td>
<td>D = abs(L - S)</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>Max</strong></td>
<td>D = max(L, S)</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>Min</strong></td>
<td>D = min(L, S)</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>Bitwise And</strong></td>
<td>D = L &amp; S</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>Bitwise Or</strong></td>
<td>D = L</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>Bitwise Xor</strong></td>
<td>D = L ^ S</td>
<td>Normal</td>
</tr>
</tbody>
</table>
D=(L*aL+S*(1-aL)*aS), aD=aS
Dr=(Lr*aLr+Sr*(1-aLr)*aSr), Dgb=Sgb, aD=aS
Dg=(Lg*aLg+SG*(1-aLg)*aSg), Drb=Srb, aD=aS
Db=(Lb*aLb+Sb*(1-aLb)*aSb), Drg=Srg, aD=aS

Dhsv=(Lhsv*aLrgb+Shsv*(1-aLrgb)*aSrgb) / (aLrgb+(1-aLrgb)*aSrgb), aD=aS
Dh=(Lh*aLr+Sh*(1-aLr)*aSr), Dsv=Lsv, aD=aS
Ds=(Ls*aLg+Ss*(1-aLg)*aSg), Dhv=Lhv, aD=aS
Dv=(Lv*aLb+Sv*(1-aLb)*aSb), Dhs=Lhs, aD=aS

Dv=min(Lv,Sv), Dhs=Lhs, aD=aS
Dv=max(Lv,Sv), Dhs=Lhs, aD=aS
Ds=max(Ls,Ss), Dhv=Lhv, aD=aS
Ds=min(Ls,Ss), Dhv=Lhv, aD=aS
i=random 0 or 1, D=i?L:S, aD=i+aS
### Appendix G. Image Layer Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>behind</td>
<td>$D = (S \ast, aS + L \cdot (1-aS) \cdot aL) / (aS + (1-aS) \cdot aL)$</td>
<td>$aD = (aS + (1-aS) \cdot aL)$ simply swap $S$ and $L$</td>
</tr>
<tr>
<td>erase</td>
<td>$D = S$, $aD = aS \cdot (1-aL)$</td>
<td></td>
</tr>
<tr>
<td>screen</td>
<td>$1-(1-S) \cdot (1-L)$</td>
<td>apply alpha as &quot;normal&quot;</td>
</tr>
<tr>
<td>overlay</td>
<td>$(1-(1-a) \cdot (1-b) - a \cdot b) \cdot a + a \cdot b$</td>
<td>apply alpha as &quot;normal&quot;</td>
</tr>
<tr>
<td>burn_alpha</td>
<td>$aD = aL$, $D = L + S$; experimental, may change or be removed</td>
<td></td>
</tr>
<tr>
<td>equal</td>
<td>each channel $D = \max$ if $L = S$, $0$ otherwise, apply with alpha</td>
<td></td>
</tr>
<tr>
<td>not_equal</td>
<td>each channel $D = \max$ if $L \neq S$, $0$ otherwise, apply with alpha</td>
<td></td>
</tr>
<tr>
<td>less</td>
<td>each channel $D = \max$ if $L &lt; S$, $0$ otherwise, apply with alpha</td>
<td></td>
</tr>
<tr>
<td>more</td>
<td>each channel $D = \max$ if $L &gt; S$, $0$ otherwise, apply with alpha</td>
<td></td>
</tr>
<tr>
<td>less_or_equal</td>
<td>each channel $D = \max$ if $L \leq S$, $0$ otherwise, apply with alpha</td>
<td></td>
</tr>
<tr>
<td>more_or_equal</td>
<td>each channel $D = \max$ if $L \geq S$, $0$ otherwise, apply with alpha</td>
<td></td>
</tr>
<tr>
<td>logic_equal</td>
<td>logic: $D =$ white and opaque if $L = S$, black and transparent otherwise</td>
<td></td>
</tr>
<tr>
<td>logic_not_equal</td>
<td>logic: $D =$ white and opaque if any $L \neq S$, black and transparent otherwise</td>
<td></td>
</tr>
<tr>
<td>logic_strict_less</td>
<td>logic: $D =$ white and opaque if all $L &lt; S$, black and transparent otherwise</td>
<td></td>
</tr>
</tbody>
</table>
logic: $D = \text{white and opaque}$ if all $L \leq S$, black and transparent otherwise

logic: $D = \text{white and opaque}$ if all $L = L$, black and transparent otherwise

logic: $D = \text{white and opaque}$ if all $L = S$, black and transparent otherwise
Appendix H

Image. Color colors
<table>
<thead>
<tr>
<th>Color Name</th>
<th>Color Bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>peachpuff2</td>
<td><img src="image1" alt="" /></td>
</tr>
<tr>
<td>seashell2</td>
<td><img src="image2" alt="" /></td>
</tr>
<tr>
<td>darkorange</td>
<td><img src="image3" alt="" /></td>
</tr>
<tr>
<td>peachpuff4</td>
<td><img src="image4" alt="" /></td>
</tr>
<tr>
<td>peachpuff</td>
<td><img src="image5" alt="" /></td>
</tr>
<tr>
<td>peachpuff1</td>
<td><img src="image6" alt="" /></td>
</tr>
<tr>
<td>linen</td>
<td><img src="image7" alt="" /></td>
</tr>
<tr>
<td>bisque3</td>
<td><img src="image8" alt="" /></td>
</tr>
<tr>
<td>burlywood4</td>
<td><img src="image9" alt="" /></td>
</tr>
<tr>
<td>burlywood3</td>
<td><img src="image10" alt="" /></td>
</tr>
<tr>
<td>burlywood2</td>
<td><img src="image11" alt="" /></td>
</tr>
<tr>
<td>burlywood1</td>
<td><img src="image12" alt="" /></td>
</tr>
<tr>
<td>bisque4</td>
<td><img src="image13" alt="" /></td>
</tr>
<tr>
<td>orange4</td>
<td><img src="image14" alt="" /></td>
</tr>
<tr>
<td>bisque2</td>
<td><img src="image15" alt="" /></td>
</tr>
<tr>
<td>orange3</td>
<td><img src="image16" alt="" /></td>
</tr>
<tr>
<td>bisque</td>
<td><img src="image17" alt="" /></td>
</tr>
<tr>
<td>orange</td>
<td><img src="image18" alt="" /></td>
</tr>
<tr>
<td>orange1</td>
<td><img src="image19" alt="" /></td>
</tr>
<tr>
<td>antiquewhite3</td>
<td><img src="image20" alt="" /></td>
</tr>
<tr>
<td>tan</td>
<td><img src="image21" alt="" /></td>
</tr>
<tr>
<td>antiquewhite2</td>
<td><img src="image22" alt="" /></td>
</tr>
<tr>
<td>antiquewhite1</td>
<td><img src="image23" alt="" /></td>
</tr>
<tr>
<td>antiquewhite4</td>
<td><img src="image24" alt="" /></td>
</tr>
<tr>
<td>navajowhite4</td>
<td><img src="image25" alt="" /></td>
</tr>
<tr>
<td>navajowhite3</td>
<td><img src="image26" alt="" /></td>
</tr>
<tr>
<td>antiquewhite</td>
<td><img src="image27" alt="" /></td>
</tr>
<tr>
<td>navajowhite2</td>
<td><img src="image28" alt="" /></td>
</tr>
<tr>
<td>navajowhite</td>
<td><img src="image29" alt="" /></td>
</tr>
<tr>
<td>navajowhite1</td>
<td><img src="image30" alt="" /></td>
</tr>
<tr>
<td>blanchedalmond</td>
<td><img src="image31" alt="" /></td>
</tr>
<tr>
<td>darkgoldenrod4</td>
<td><img src="image32" alt="" /></td>
</tr>
<tr>
<td>darkgoldenrod</td>
<td><img src="image33" alt="" /></td>
</tr>
<tr>
<td>darkgoldenrod3</td>
<td><img src="image34" alt="" /></td>
</tr>
<tr>
<td>papayawhip</td>
<td><img src="image35" alt="" /></td>
</tr>
<tr>
<td>darkgoldenrod2</td>
<td><img src="image36" alt="" /></td>
</tr>
<tr>
<td>darkgoldenrod1</td>
<td><img src="image37" alt="" /></td>
</tr>
</tbody>
</table>
APPENDIX H. IMAGE.COLOR COLORS

<table>
<thead>
<tr>
<th>lemonchiffon2</th>
<th>lemonchiffon1</th>
<th>lemonchiffon</th>
<th>lemonchiffon4</th>
<th>lightyellow4</th>
<th>lightgoldenrod4</th>
<th>lightyellow3</th>
<th>lightyellow2</th>
<th>lightyellow1</th>
<th>lightyellow</th>
<th>beige</th>
<th>ivory4</th>
<th>ivory3</th>
<th>ivory2</th>
<th>ivory1</th>
<th>ivory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>olivedrab4</td>
<td>olivedrab</td>
<td>olivedrab3</td>
<td>yellowgreen</td>
<td>olivedrab2</td>
<td>olivedrab1</td>
<td>greenyellow</td>
<td>darkolivegreen</td>
<td>darkolivegreen4</td>
<td>darkolivegreen3</td>
<td>darkolivegreen2</td>
<td>darkolivegreen1</td>
<td>chartreuse3</td>
<td>chartreuse</td>
<td>chartreuse1</td>
<td>chartreuse4</td>
</tr>
<tr>
<td>Color</td>
<td>Hex Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>forestgreen</td>
<td>#228B22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>limegreen</td>
<td>#00FF00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>palegreen4</td>
<td>#77FF00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>palegreen3</td>
<td>#778800</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>palegreen2</td>
<td>#776400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightgreen</td>
<td>#77FF77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>palegreen1</td>
<td>#77FF99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>palegreen</td>
<td>#77FFCC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkseagreen4</td>
<td>#4D4D4D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkseagreen</td>
<td>#4D4D4D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkseagreen3</td>
<td>#4D4D4D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkseagreen2</td>
<td>#4D4D4D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkseagreen1</td>
<td>#4D4D4D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>honeydew4</td>
<td>#DADEDE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>honeydew3</td>
<td>#DADEDE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>honeydew2</td>
<td>#DADEDE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>honeydew1</td>
<td>#DADEDE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>seagreen4</td>
<td>#777777</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>seagreen</td>
<td>#777777</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>seagreen3</td>
<td>#777777</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>seagreen2</td>
<td>#777777</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>seagreen1</td>
<td>#777777</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mediumseagreen</td>
<td>#777777</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>springgreen4</td>
<td>#99FF00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>springgreen3</td>
<td>#99FF00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>springgreen2</td>
<td>#99FF00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>springgreen</td>
<td>#99FF00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>springgreen1</td>
<td>#99FF00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pikeygreen</td>
<td>#99CC99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mintcream</td>
<td>#99CC99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mediumspringgreen</td>
<td>#99CC99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aquamarine4</td>
<td>#00FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aquamarine3</td>
<td>#00FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mediumaquamarine</td>
<td>#00FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aquamarine2</td>
<td>#00FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aquamarine</td>
<td>#00FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aquamarine1</td>
<td>#00FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>turquoise</td>
<td>#40E0D0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightseagreen</td>
<td>#E5D9B8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color</td>
<td>Hex Color</td>
<td>RGB Color</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------</td>
<td>-------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cyan4</td>
<td>#0f7fcf</td>
<td>(15, 127, 252)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkcyan</td>
<td>#0056b3</td>
<td>(0, 86, 179)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cyan3</td>
<td>#0084ff</td>
<td>(0, 132, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cyan2</td>
<td>#00aaff</td>
<td>(0, 170, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cyan1</td>
<td>#00bfff</td>
<td>(0, 191, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mediumturquoise</td>
<td>#00ccff</td>
<td>(0, 204, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkturquoise</td>
<td>#0064ff</td>
<td>(0, 100, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>turquoise4</td>
<td>#009bff</td>
<td>(0, 155, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>turquoise3</td>
<td>#008aff</td>
<td>(0, 139, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>turquoise2</td>
<td>#007aff</td>
<td>(0, 120, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>turquoise1</td>
<td>#006aff</td>
<td>(0, 100, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkslategrey</td>
<td>#282828</td>
<td>(40, 40, 40)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkslategrey4</td>
<td>#222222</td>
<td>(34, 34, 34)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkslategrey3</td>
<td>#333333</td>
<td>(51, 51, 51)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkslategrey2</td>
<td>#4d4d4d</td>
<td>(77, 77, 77)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkslategrey1</td>
<td>#6b6b6b</td>
<td>(107, 107, 107)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>paleturquoise4</td>
<td>#e5f9ff</td>
<td>(229, 249, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>paleturquoise3</td>
<td>#eff4ff</td>
<td>(255, 244, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>paleturquoise2</td>
<td>#eff4f1</td>
<td>(255, 244, 225)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>paleturquoise</td>
<td>#eff2f2</td>
<td>(255, 242, 242)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>paleturquoise1</td>
<td>#eff0f0</td>
<td>(255, 240, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightcyan4</td>
<td>#e0f8ff</td>
<td>(224, 248, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightcyan3</td>
<td>#edeaff</td>
<td>(237, 242, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightcyan2</td>
<td>#e7e7f1</td>
<td>(231, 231, 241)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightcyan1</td>
<td>#e7e7e7</td>
<td>(231, 231, 231)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>azure4</td>
<td>#64faff</td>
<td>(100, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>azure3</td>
<td>#6affff</td>
<td>(102, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>azure2</td>
<td>#6affac</td>
<td>(102, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>azure1</td>
<td>#6affa9</td>
<td>(102, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>azure</td>
<td>#6affa3</td>
<td>(102, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cadetblue4</td>
<td>#64faff</td>
<td>(100, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cadetblue3</td>
<td>#6affff</td>
<td>(102, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cadetblue2</td>
<td>#6affac</td>
<td>(102, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cadetblue1</td>
<td>#6affa9</td>
<td>(102, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>powderblue</td>
<td>#e6faff</td>
<td>(230, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deepskyblue4</td>
<td>#64faff</td>
<td>(100, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deepskyblue3</td>
<td>#6affff</td>
<td>(102, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deepskyblue2</td>
<td>#6affac</td>
<td>(102, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deepskyblue1</td>
<td>#6affa9</td>
<td>(102, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deepskyblue</td>
<td>#6affa3</td>
<td>(102, 255, 255)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color Name</td>
<td>Color Example</td>
<td>Color Example</td>
<td>Color Example</td>
<td>Color Example</td>
<td>Color Example</td>
<td>Color Example</td>
<td>Color Example</td>
<td>Color Example</td>
<td>Color Example</td>
<td>Color Example</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>royalblue4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>royalblue3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>royalblue2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>royalblue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>royalblue1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>navyblue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>navy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>blue4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkblue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mediumblue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>blue3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>blue2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>blue1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>midnightblue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lavender</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ghostwhite</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slateblue4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkslateblue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slateblue3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slateblue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slateblue2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mediumslateblue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slateblue1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightslateblue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mediumpurple4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mediumpurple3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mediumpurple</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mediumpurple2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mediumpurple1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>purple4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>purple3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>purple2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>blueviolet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>purple1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>purple</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkorchid4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkorchid3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkorchid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkorchid2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>darkorchid1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color Name</td>
<td>Color Sample</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hotpink</td>
<td><img src="image" alt="hotpink sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hotpink1</td>
<td><img src="image" alt="hotpink1 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>violetred1</td>
<td><img src="image" alt="violetred1 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>violetred3</td>
<td><img src="image" alt="violetred3 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>violetred2</td>
<td><img src="image" alt="violetred2 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hotpink2</td>
<td><img src="image" alt="hotpink2 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hotpink3</td>
<td><img src="image" alt="hotpink3 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maroon</td>
<td><img src="image" alt="maroon sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>palevioletred4</td>
<td><img src="image" alt="palevioletred4 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>palevioletred3</td>
<td><img src="image" alt="palevioletred3 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>palevioletred2</td>
<td><img src="image" alt="palevioletred2 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lavanderblush4</td>
<td><img src="image" alt="lavanderblush4 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>palevioletred1</td>
<td><img src="image" alt="palevioletred1 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lavanderblush2</td>
<td><img src="image" alt="lavanderblush2 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lavanderblush3</td>
<td><img src="image" alt="lavanderblush3 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lavanderblush1</td>
<td><img src="image" alt="lavanderblush1 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lavanderblush</td>
<td><img src="image" alt="lavanderblush sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pink4</td>
<td><img src="image" alt="pink4 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pink3</td>
<td><img src="image" alt="pink3 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pink2</td>
<td><img src="image" alt="pink2 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pink1</td>
<td><img src="image" alt="pink1 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pink</td>
<td><img src="image" alt="pink sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightpink</td>
<td><img src="image" alt="lightpink sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightpink4</td>
<td><img src="image" alt="lightpink4 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightpink3</td>
<td><img src="image" alt="lightpink3 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightpink2</td>
<td><img src="image" alt="lightpink2 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lightpink1</td>
<td><img src="image" alt="lightpink1 sample" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Index

Calendar.Gregorian., 307
Geographical.Countries.‘., 316
Image.Color.Color., 191

/#
/\call/
Protocols.LysKOM.Connection
/\call/., 230
#
#string, 331
1
1, 6
1.1, 7
10.1, 113
11.1, 123
12.1, 137
12.26.1, 219
12.7.1, 188
13.1, 223
13.1.1, 225
13.2.1, 227
13.2.3.1, 231
13.3.1, 232
14.1, 233
14.10.1, 267
14.11.1, 273
14.11.2.1, 280
14.14.1, 296
14.16.1, 305
14.16.2.1, 307
14.16.2.3.1, 307
14.18.1, 308
14.20.1, 311
14.20.3.1, 312
14.21.1, 312
14.25.1, 320
14.26.1, 323
14.26.3.1, 326
14.27.1, 327
17.1, 379
17.2.1, 381
2.1, 17
2.1.1, 17
2.2.1, 19
2.3.1, 21
3.1, 25
3.1.1, 25
3.2.1, 27
3.3.1, 29
4.1, 33
4.1.1, 33
4.2.1, 38
5.1, 53
6.1, 65
7.1, 75
8.1, 80
9.1, 83
9.6.1, 99
introduction.1, 4
1, 7
2, 8
3, 9
4, 10
5, 11
6, 12
7, 12
437
10, 112
12.10, 194
14.10, 266
6.10, 73
9.10, 105
1, 113
2, 114
3, 119
11, 121
12.11, 194
14.11, 272
1, 123
2, 125
12, 133
12.12, 196
14.12, 286
1, 137
10, 194
11, 194
12, 196
13, 197
14, 204
15, 205
16, 206
17, 207
18, 210
19, 212
2, 171
20, 212
21, 213
22, 214
23, 215
24, 216
25, 218
26, 219
1, 219
2, 221
27, 221
3, 180
4, 184
5, 186
6, 186
7, 186
1, 188
8, 191
9, 193
13, 222
12.13, 197
14.13, 290
1, 223
1, 225
2, 227
1, 227
2, 229
3, 230
1, 231
3, 231
1, 232
14, 232
12.14, 204
14.14, 296
1, 233
10, 266
1, 267
2, 269
3, 271
11, 272
1, 273
2, 279
1, 280
2, 282
12, 286
13, 290
14, 296
1, 296
2, 296
3, 298
4, 298
5, 299
6, 301
15, 301
16, 305
1, 305
2, 306
1, 307
2, 307
3, 307
1, 307
17, 307
18, 308
1, 308
19, 310
2, 239
20, 311
1, 311
2, 311
3, 312
1, 312
21, 312
1, 312
2, 313
22, 313
23, 314
24, 317
25, 318
1, 320
26, 323
1, 323
2, 325
3, 326
1, 326
27, 327
1, 327
3, 244
4, 246
5, 248
6, 250
7, 254
8, 256
9, 260
15, 328
12.15, 205
14.15, 301
16, 331
12.16, 206
14.16, 305
17, 378
12.17, 207
14.17, 307
1, 379
2, 381
1, 381
2, 381
3, 384
4, 387
5, 388
6, 391
7, 391
3, 391
18
12.18, 210
14.18, 308
19
12.19, 212
14.19, 310
2
2, 13
1.2, 8
10.2, 114
11.2, 125
12.2, 171
12.26.2, 221
13.2, 227
13.2.2, 229
14.10.2, 269
14.11.2, 279
14.11.2.2, 282
14.14.2, 296
14.16.2, 306
14.16.2.2, 307
14.2, 239
14.20.2, 311
14.21.2, 313
14.26.2, 325
<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.6, 98</td>
<td>6.9, 72</td>
</tr>
<tr>
<td>1, 65</td>
<td>9.9, 105</td>
</tr>
<tr>
<td>10, 73</td>
<td>1, 83</td>
</tr>
<tr>
<td>2, 65</td>
<td>10, 105</td>
</tr>
<tr>
<td>3, 66</td>
<td>2, 93</td>
</tr>
<tr>
<td>4, 66</td>
<td>3, 94</td>
</tr>
<tr>
<td>5, 67</td>
<td>4, 94</td>
</tr>
<tr>
<td>6, 70</td>
<td>5, 96</td>
</tr>
<tr>
<td>7, 71</td>
<td>6, 98</td>
</tr>
<tr>
<td>8, 71</td>
<td>1, 99</td>
</tr>
<tr>
<td>9, 72</td>
<td>2, 100</td>
</tr>
<tr>
<td>7</td>
<td>7, 100</td>
</tr>
<tr>
<td>7, 74</td>
<td>8, 101</td>
</tr>
<tr>
<td>1.7, 12</td>
<td>9, 105</td>
</tr>
<tr>
<td>12.7, 186</td>
<td></td>
</tr>
<tr>
<td>14.7, 254</td>
<td></td>
</tr>
<tr>
<td>17.2.7, 391</td>
<td></td>
</tr>
<tr>
<td>5.7, 60</td>
<td></td>
</tr>
<tr>
<td>6.7, 71</td>
<td></td>
</tr>
<tr>
<td>9.7, 100</td>
<td></td>
</tr>
<tr>
<td>1, 75</td>
<td></td>
</tr>
<tr>
<td>2, 76</td>
<td></td>
</tr>
<tr>
<td>3, 77</td>
<td></td>
</tr>
<tr>
<td>4, 77</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>8, 77</td>
<td></td>
</tr>
<tr>
<td>12.8, 191</td>
<td></td>
</tr>
<tr>
<td>14.8, 256</td>
<td></td>
</tr>
<tr>
<td>5.8, 62</td>
<td></td>
</tr>
<tr>
<td>6.8, 71</td>
<td></td>
</tr>
<tr>
<td>9.8, 101</td>
<td></td>
</tr>
<tr>
<td>1, 80</td>
<td></td>
</tr>
<tr>
<td>2, 81</td>
<td></td>
</tr>
<tr>
<td>3, 81</td>
<td></td>
</tr>
<tr>
<td>4, 82</td>
<td></td>
</tr>
<tr>
<td>5, 82</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>9, 82</td>
<td></td>
</tr>
<tr>
<td>12.9, 193</td>
<td></td>
</tr>
<tr>
<td>14.9, 260</td>
<td></td>
</tr>
<tr>
<td>5.9, 63</td>
<td></td>
</tr>
</tbody>
</table>

APPENDIX H. IMAGE.COLOR COLORS

Image.TTF.', 219
Image.TTF.Face.', 221
Protocols.HTTP.Query.', 227
Protocols.LysKOM.Request
!Request.', 231
!=, 56
%, 53
&., 57
Image.Image.', 168
Image.Image.'&., 168
'()', 77
'*', 53
Image.Colortable.'*', 174
Image.Image.'*', 169
Math.Matrix.'*', 309
'+', 53
Calendar.time_unit.'+', 306
Image.Colortable.'+', 179
Image.Image.'+', 170
Math.Matrix.'+', 308
'-', 53
Calendar.time_unit.'-', 306
ADT
Table, 260
ASCII
encode, 265
table
'==, 264
'[', 264
ADT.Table.table.'[', 264
Geographical.Countries.'[', 316
'[]=, 264
'[, 57
"*
Image.Colortable."*, 174
Math.Matrix."*, 309
"+
Math.Matrix."+, 308
"-
Math.Matrix."-, 310
"×
Math.Matrix."×, 309
'—, 57
Image.Image.'—, 170
'=', 57
'×
APPENDIX H. IMAGE.COLOR COLORS

rename, 262
reverse, 262
row, 264
rsort, 262
select, 263
sizeof, 261
sort, 262
type, 262
values, 261
where, 263
ADT.Table.table.append_bottom, 265
append_path
ADT.Table.table.append_right, 265
apply_matrix
ADT.Table.table.append_right, 265
apply_max
ADT.Table.table.append_right, 265
arcfour_random

Array, 125
diff, 125
diff_compare_table, 126
diff_longest_sequence, 127
every_nth, 128
filter, 128
longest_ordered_sequence, 129
map, 130
permute, 130
reduce, 130
rreduce, 131
search_array, 131
shuffle, 131
sort_array, 132
splice, 132
sum_arrays, 133
uniq, 133
array_index, 388
array_index_no_free, 388
array_set_index, 388
array_scnsnf, 388
arrayyp, 388
asin, 388
assign
Stdio.File.assign, 90
assign_svalue, 382
assign_svalue_no_free, 383
async

Protocols.LysKOM.Request
_.Request.async, 231
_.async

Protocols.LysKOM.Request
_.Request_.async, 231
async_.call/

Protocols.LysKOM.Connection
_.async_.call/, 230
async_cb_.call/

Protocols.LysKOM.Connection
_.async_cb_.call/, 230
async_request

Protocols.HTTP.Query.async_request, 225
atan, 339
atan2, 339
atexit, 339
autocrop

Image.Image.autocrop, 139
Image.Layer.autocrop, 181
available_modes

Image.Layer.available_modes, 181
average

Image.Image.average, 139
AVS

Image.AVS, 194

B

B, 394
backtrace, 339
baseline

Image.Font.baseline, 184
basename, 340
begin_shared_string, 386
begin_wide_shared_string, 386
big_query

Mysql.mysql.big_query, 291
binary_data

Mysql.mysql.binary_data, 291
bind

Stdio.Port.bind, 94

Stdio.UDP.bind, 96
Yp.YpDomain.bind, 257
bindtextdomain

Locale.Gettext.bindtextdomain, 301
bitscale

Image.Image.bitscale, 140
BMP

Image.BMP, 194
body_parts

MIME.Message.body_parts, 280
boundary

MIME.Message.boundary, 280
box

Image.Image.box, 140
bright

Image.Color.Color.bright, 189
broadcast

Thread.Condition.broadcast, 117
C

C, 399
Calendar, 305
Gregorian, 306

, 307
Stardate, 307
TNGDate, 307
Year, 307
parse, 307
time_unit, 305
'+' , 306
'-' , 306
greater, 305
lesser, 306
next, 306
prev, 306
call_function, 340
call_out, 340
call_out_info, 340
capitalize
String.capitalize, 124
cast, 61
Crypto.cast, 297
  Geographical.Countries.cast, 314
Geographical.Position.cast, 313
Gmp.mpz.cast, 247
Image.Color.Color.cast, 189
Image.Colortable.cast, 172
Image.Image.cast, 141
Math.Matrix.cast, 308
MIME.Message.cast, 282
  Protocols.HTTP.Query.cast, 225
catch, 76
cd, 341
ceil, 341
change, color
  Image.Image.change_color, 141
charset
  MIME.Message.charset, 280
chmod, 341
chroot, 233
_chunk
  Image.PNG._chunk, 208
circle
  Image.Image.circle, 141
clear
  Image.Image.clear, 142
client
  Protocols.DNS.client, 232
clone, 342
  Image.Image.clone, 143
  Image.Layer.clone, 182
close
  Gdbm.gdbm.close, 249
Stdio.File.close, 84
cmyk
  Image.Color.cmyk, 188
  Image.Color.Color.cmyk, 189
col
  ADT.Table.table.col, 264
Color
  Image.Color, 186
  Image.Color.Color, 188
  Image.Image.color, 143
colors, 413
Colortable
  Image.Colortable, 171
colortable
  Image.colortable, 186
column, 342
combine, path, 342
commit
  Yabu.transaction.commit, 271
common, prefix
  String.common_prefix, 124
compile, 343
compile_file, 343
compile_string, 343
Condition
  Thread.Condition, 116
connect
  Stdio.File.connect, 91
Connection
  Protocols.LysKOM.Connection, 230
copy
  Image.Image.copy, 144
copy_array, 388
copy_mapping, 391
copy_value, 343
corners
  Image.Colortable.corners, 172
cos, 343
count
  String.count, 125
Countries
  Geographical.Countries, 314
cpp, 344
crc32
create
ADT.Table.table.create, 261
Crypto.randomness.arcfour_random
create, 313
Gdbm.gdbm.create, 248
Geographical.Position.create,
313
Gmp.mpz.create, 246
Gz.deflate.create, 255
Gz.inflate.create, 255
Image.Color.Color.create, 190
Image.Colortable.create, 171
Image.Font.create, 185
Image.Image.create, 145
Image.Layer.create, 182
Image.TTF.FaceInstance.create,
221
Math.Matrix.create, 308
MIME.Message.create, 283
Mysql.mysql.create, 291
Parser.SGML.create, 318
Protocols.DNS.client.create,
232
Protocols.LysKOM.Connection
/create, 230
Protocols.LysKOM.Session
/create, 227
Regexp.create, 245
Stdio.File.create, 83
Stdio.Port.create, 95
Stdio.Readline.create, 100
Stdio.Terminfo.Termcap.create,
100
Thread.Fifo.create, 118
Yabu.db.create, 268
create_db
Mysql.mysql.create_db, 292
create_person
Protocols.LysKOM.Session
/create_person, 228
create_process
Process.create_process, 239
create_text
Protocols.LysKOM.Session
/create_text, 228
crop
Image.Layer.crop, 182
cross
Math.Matrix.cross, 309
crypt, 344
Crypto.rc4.crypt, 298
crypt_block, 296
Crypto, 312
cast, 297
des, 297
idea, 297
randomness, 312
arcfour_random, 313
create, 313
read, 313
pike_random, 312
read, 312
rc4, 298
crypt, 298
set_decrypt_key, 298
set_encrypt_key, 298
rsa, 299
decrypt, 300
encrypt, 300
generate_key, 299
query_blocksize, 300
set_private_key, 299
set_public_key, 299
sha_sign, 301
sha_verify, 301
sign, 300
verify, 300
crypto, 296
cftime, 344
cubicles
Image.Colortable.cubicles, 172
D
D, 401
dark
   Image.Color.Color.dark, 189
data
   Protocols.HTTP.Query.data,
dct
degettext
   Locale.Gettext.degettext, 302
dct
   Image.Image.dct, 145
decode
   Image.GIF._decode, 203
   Image.ILBM._decode, 206
   Image.PNG._decode, 209
   Image.XCF._decode, 215
decode
   Image.XCF.__decode, 215
decode
   Image.ANY._decode, 193
   Image.AVS._decode, 194
   Image.BMP._decode, 194
   Image.GD._decode, 196
   Image.GIF._decode, 201
   Image.HRZ._decode, 204
   Image.ILBM._decode, 206
   Image.JPG._decode, 216
   Image.PCX._decode, 207
   Image.PNG._decode, 209
   Image.TGA._decode, 213
   Image.TIFF._decode, 219
   Image.XBM._decode, 213
   Image.XCF._decode, 214
   Image.XWD._decode, 216
decode
   ADT.Table.table.decode, 264
   Image.ANY._decode, 193
   Image.AVS.decode, 194
   Image.BMP.decode, 194
   Image.GD.decode, 196
   Image.GIF.decode, 197
   Image.HRZ.decode, 204
   Image.ILBM.decode, 205
   Image.JPG.decode, 216
   Image.PCX.decode, 207
   Image.PNG.decode, 208
   Image.PNM.decode, 211
   Image.TGA.decode, 212
   Image.TIFF.decode, 218
   Image.XBM.decode, 213
   Image.XCF.decode, 214
   Image.XFace.decode, 221
   Image.XWD.decode, 216
   MIME.decode, 273
decode_alpha
   Image.ANY.decode_alpha, 193
   Image.GD.decode_alpha, 196
decode_base64
   MIME.decode_base64, 274
decode_header
   Image.BMP.decode_header, 194
   Image.GD.decode_header, 196
   Image.JPG.decode_header, 216
   Image.XFace.decode_header, 222
   Image.XWD.decode_header, 216
decode_layer
   Image.GIF.decode_layer, 197
decode_layers
   Image.GIF.decode_layers, 197
   Image.XCF.decode_layers, 214
decode_map
   Image.GIF.decode_map, 198
decode_pseudocolor
   Image.X.decode_pseudocolor, 192
decode_qp
   MIME.decode_qp, 274
decode_truecolor
   Image.X.decode_truecolor, 192
decode_truecolor_masks
Image.X.decode_truecolor_masks, 192
decode_uue
   MIME.decode_uue, 274
decode_value, 345
decode_word
   MIME.decode_word, 274
decrypt
   Crypto.rsa.decrypt, 300
default_yp_domain
   Yp.default_yp_domain, 257
define, 329
deflate
   Gz.deflate, 254
   Gz.deflate.deflate, 255
delete
   Gdbm.gdbm.delete, 249
   Yabu.table.delete, 270
delete_url
   Protocols.HTTP.delete_url, 223
des
   Crypto.des, 297
describe_backtrace, 345
describe_error, 345
destruct, 345
dgettext
   Locale.Gettext.dgettext, 302
diff
   Array.diff, 125
diff_compare_table
   Array.diff_compare_table, 126
diff_longest_sequence
   Array.diff_longest_sequence, 127
digest, 298
digits
   Gmp.mpz.digits, 247
dirname, 346
   _disable_threads, 333
disp_params
   MIME.Message.disp_params, 280
disposition
   MIME.Message.disposition, 280
distancesq
   Image.Image.distancesq, 146
distinct
   ADT.Table.table.distinct, 263
DNS
   Protocols.DNS, 232
   _do_call_outs, 333
   downloaded_bytes
   Protocols.HTTP.Query.downloaded_bytes, 226
drop_db
   Mysql.mysql.drop_db, 292
dull
   Image.Color.Color.dull, 189
dup
   Stdio.File.dup, 90
dup2
   Stdio.File.dup2, 90
E
   E, 405
   else, 330
   elseif, 330
   enable_broadcast
   Stdio.UDP.enable_broadcast, 97
   _encode
   Image.GIF._encode, 202
   Image.PCX._encode, 207
   Image.TIFF._encode, 218
   encode
   ADT.Table.ASCII.encode, 265
   ADT.Table.table.encode, 264
   Image.AVS.encode, 194
   Image.BMP.encode, 195
   Image.GD.encode, 196
   Image.GIF.encode, 198
   Image.HRZ.encode, 204
APPENDIX H. IMAGE.COLOR COLORS

Image.ILBM.encode, 205
Image.JPEG.encode, 217
Image.PNG.encode, 207
Image.PCX.encode, 208
Image.PNM.encode, 211
Image.TGA.encode, 212
Image.TIFF.encode, 218
Image.XBM.encode, 213
Image.XFace.encode, 222
MIME.encode, 275
encode_ascii
Image.PNM.encode_ascii, 211
encode_base64
MIME.encode_base64, 275
encode_binary
Image.PNM.encode_binary, 211
encode_P1
Image.PNM.encode_P1, 211
encode_P2
Image.PNM.encode_P2, 211
encode_P3
Image.PNM.encode_P3, 211
encode_P4
Image.PNM.encode_P4, 211
encode_P5
Image.PNM.encode_P5, 211
encode_P6
Image.PNM.encode_P6, 211
encode_pseudocolor
Image.X.encode_pseudocolor, 192
encode_qp
MIME.encode_qp, 276
encode_trans
Image.GIF.encode_trans, 198
encode_truecolor
Image.X.encode_truecolor, 192
encode_truecolor_masks
Image.X.encode_truecolor_masks, 192
encode_uue
MIME.encode_uue, 276
encode_value, 346
encode_value_canonic, 346
encode_word
MIME.encode_word, 276
encrypt
Crypto.rsa.encrypt, 300
end_block
Image.GIF.end_block, 199
end_shared_string, 386
endif, 330
enumerate, 346
equal, 347
errno, 347
Stdio.File.errno, 87
Stdio.Port.errno, 96
error, 330
Mysql.mysql.error, 292
everynth
Array.everynth, 128
exceptions, 76
exec
Process.exec, 243
exece, 348
Process.exece, 242
exist
Stdio.exist, 101
.exit, 333
exit, 348
exp, 348
explode, 388
Simulate=explode, 287
explode_path, 348
F
F, 406
f_aggregate_array, 388
Face
Image.TTF.Face, 220
FaceInstance
Image.TTF.FaceInstance, 221
fd
Process.Spawn.fd, 243
feed
Parser.SGML.feed, 318
fetch
Gdbm.gdbm.fetch, 249
Fifo
Thread.Fifo, 117
FILE
Stdio.FILE, 93
File
Stdio.File, 83
file_size
Stdio.file_size, 101
file_stat, 349
file_truncate, 349
fill
Image.Layer.fill, 183
fill_alpha
Image.Layer.fill_alpha, 183
filter, 349
Array.filter, 128
filter_array
Simulate.filter_array, 287
find_all_options
Getopt.find_all_options, 251
find_autocrop
Image.Image.find_autocrop, 139
Image.Layer.find_autocrop, 181
find_call_out, 350
find_max
Image.Image.find_max, 147
find_min
Image.Image.find_min, 147
find_option
Getopt.find_option, 250
finish
Parser.SGML.finish, 318
firstkey
Gdbm.gdbm.firstkey, 249
FLOAT_TYPE, 2??
floatp, 350
floor, 350
floyd_steinberg
Image.Colortable.floyd_steinberg, 173
flush
Image.TTF.Face.flush, 220
Font
Image.Font, 184
foreach, 28
fork, 351
free_array, 388
free_mapping, 390
free_string, 385
free_svalue, 382
free_svalues, 382
from_domain
Geographical.Countries.from_domain, 314
full
Image.Colortable.full, 174
function_name, 351
function_object, 351
functionp, 351
functions, 331
fuzzymatch
String.fuzzymatch, 123
G
G, 408
gamma
Image.Image.gamma, 147
gauge, 77
gc, 352
gcd
Gmp.mpz.gcd, 247
_gce_block
Image.GIF._gce_block, 203
GD
Image.GD, 196
Gdbm, 248
gdbm
APPENDIX H. IMAGE.COLOR COLORS

close, 249
create, 248
delete, 249
fetch, 249
firstkey, 249
nextkey, 249
reorganize, 250
store, 249
sync, 250
generate_boundary
MIME.generate_boundary, 277
generate_key
Crypto.rsa.generate_key, 299
Geographical, 313
Countries, 314
‘
, 316
‘[], 316
cast, 314
from_domain, 314
Position, 313
‘==, 314
‘\., 314
‘\_hash, 314
‘\;\; 314
cast, 313
create, 313
latitude, 313
longitude, 313
get
 Thread.thread_thread_local.thread_local.get, 119
 Yabu.table.get, 270
get_args
Getopt.get_args, 254
get_dit, 352
get_filename
MIME.Message.get_filename, 284
get_function
Simulate.get_function, 286
get_m misc_value
 Image.Layer.get_m misc_value,
183
get_primary_mx
 Protocols.DNS.client.get_primary_mx,
232
get_profiling_info, 352
get_url
 Protocols.HTTP.get_url, 223
get_url_data
 Protocols.HTTP.get_url_data,
223
get_url_n ice
 Protocols.HTTP.get_url_n ice,
223
getchar
 Stdio.FILE.getchar, 93
cwd, 352
data
 MIME.Message.getdata, 283
getegid, 233
getencoded
 MIME.Message.getencoded,
283
getenv, 353
getuid, 234
getFallbackTerm
 Stdio.Termino.getFallbackTerm,
98
getgid, 234
groups, 234
gethostbyaddr, 234
 Protocols.DNS.client.gethostbyaddr,
232
gethostbyname, 234
 Protocols.DNS.client.gethostbyname,
232
gethostname, 235
Getopt, 250
find_all_options, 251
find_option, 250
get_args, 254
getpgrp, 235
 getpid, 235
 getpixel
   Image.Image.getpixel, 148
 getppid, 236
 gets
   Stdio.FILE.gets, 93
 getTerm
   Stdio.Terminfo.getTerm, 99
 getTermcap
   Stdio.Terminfo.getTermcap, 99
 getTerminfo
   Stdio.Terminfo.getTerminfo, 99
 Gettext
   Locale.Gettext, 301
 gettext
   Locale.Gettext.gettext, 302
 getuid, 236
 GIF
   Image.GIF, 197
 glob, 353
 Gmp, 246
 mpz, 246
   cast, 247
   create, 246
   digits, 247
   gcd, 247
 powm, 247
 probably_prime_p, 247
 size, 248
 sqrt, 247
 gmtime, 353
 gradients
   Image.Image.gradients, 148
 greater
   Calendar.time_unit.greater, 305
 Gregorian
   Calendar.Gregorian, 306
 grey
   Image.Color.Color.grey, 190
   Image.Image.grey, 148
 greylevel
   Image.Color.Color.greylevel, 189
   Image.Color.greylevel, 188
   group
     ADT.Table.table.group, 263
 guess
   Image.Color.guess, 188
 guess_subtype
   MIME.guess_subtype, 277
 Gz, 254
   crc32, 256
   deflate, 254
     create, 255
     deflate, 255
     inflate, 255
       create, 255
     inflate, 256
 H
   H, 413
 hardlink, 236
 has_index, 353
 has_value, 354
 hash, 354
 header_block
   Image.GIF.header_block, 199
 headers
   MIME.Message.headers, 281
 height
   Image.Font.height, 185
 hex
   Image.Color.Color.hex, 190
 host_info
   Mysql.mysql.host_info, 292
 HRZ
   Image.HRZ, 204
 hsv
   Image.Color.Color.hsv, 189
   Image.Color hsv, 188
 hsv_to_rgb
   Image.Image.hsv_to_rgb, 149
 html
Image.Color.Color.html, 190
Image.Color.html, 188
HTTP
Protocols.HTTP, 223
http_encode_query
Protocols.HTTP.http_encode_query, 319
http_encode_string
Protocols.HTTP.http_encode_string, 319
I
idea
Crypto.idea, 297
if, 330
ILBM
Image.ILBM, 205
Image, 133
Image.Image, 137
ANY, 193
_decode, 193
decode, 193
decode_alpha, 193
AVS, 194
_decode, 194
decode, 194
encode, 194
BMP, 194
_decode, 194
decode, 194
decode_header, 194
encode, 195
Color, 186
cmyk, 188
Color, 188
, 191
"==", 191
bright, 189
cast, 189
cmyk, 189
create, 190
dark, 189
dull, 189
grey, 190
greylevel, 189
hex, 190
hsv, 189
html, 190
light, 189
name, 190
neon, 189
rgb, 189
s, 191
greylevel, 188
guess, 188
hsv, 188
html, 188
indices, 188
rgb, 188
values, 188
Colortable, 171
"*", 174
"+", 179
"-", 180
"*", 174
add, 171
cast, 172
corners, 172
create, 171
cubicles, 172
floyd_steinberg, 173
full, 174
image, 174
map, 174
nodither, 176
ordered, 176
randomcube, 177
randomgrey, 177
reduce, 178
reduce_fs, 178
rigid, 179
spacefactors, 179
colortable, 186
Font, 184
baseline, 184
create, 185
height, 185
load, 185
set_xspacing, 185
set_yspacing, 185
text_extents, 185
write, 186
GD, 196
_decode, 196
decode, 196
decode_alpha, 196
decode_header, 196
encode, 196
GIF, 197
_decode, 201
_decode, 203
decode, 197
decode_layer, 197
decode_layers, 197
decode_map, 198
_encode, 202
encode, 198
encode_trans, 198
end_block, 199
_gca_block, 203
header_block, 199
netscape_loop_block, 200
render_block, 200
render_block, 203
HRZ, 204
decode, 204
_decode, 204
encode, 204
ILBM, 205
decode, 205
_decode, 206
_decode, 206
encode, 205
Image, 137
min, 139
mirrorx, 153
mirrory, 153
modify_by_intensity, 154
noise, 154
orient, 155
orient4, 155
outline, 155
outline_mask, 155
paste, 156
paste_alpha, 157
paste_alpha_color, 157
paste_mask, 158
phaseh, 158
phaselh, 158
phaselv, 158
phaselvh, 158
polyfill, 159
random, 159
randomgrey, 159
read_lsb_grey, 160
read_lsb_rgb, 160
rgb_to_hsv, 149
rotate, 160
rotate_ccw, 161
rotate_cw, 161
rotate_expand, 160
scale, 162
select_colors, 152
select_from, 162
setcolor, 163
setpixel, 163
skewx, 164
skewx_expand, 164
skewy, 164
skewy_expand, 164
sum, 139
sumf, 139
test, 165
threshold, 165
tuned_box, 166
turbulence, 167
write_lsb_grey, 160
write_lsb_rgb, 160
xsize, 167
ysize, 167
JPEG, 216
decode, 216
decode_header, 216
encode, 217
lay, 136
Layer, 180
alpha, 180
alpha_value, 180
autocrop, 181
available_modes, 181
clonce, 182
create, 182
crop, 182
fill, 183
fill_alpha, 183
find_autocrop, 181
get_misc_value, 183
image, 180
mode, 181
set_alpha_value, 180
set_fill, 183
set_image, 180
set_msc_value, 183
set_mode, 181
set_offset, 183
set_tiled, 183
tiled, 183
xoffset, 183
ysize, 183
_load, 136
load, 136
load_layer, 136
PCX, 207
decode, 207
\_decode, 207
delete, 207
\_encode, 207
evaluate, 207
PNG, 208
\_chunk, 208
\_decode, 209
\_decode, 209
decode, 208
encode, 208
PNM, 210
decode, 211
encode, 211
encode_ascii, 211
encode_binary, 211
encode_P1, 211
encode_P2, 211
encode_P3, 211
encode_P4, 211
encode_P5, 211
encode_P6, 211
Poly, 186
PSD, 212
TGA, 212
\_decode, 213
\_decode, 212
encode, 212
TIFF, 218
decode, 218
\_decode, 219
\_encode, 218
encode, 218
TTF, 219
\', 219
Face, 220
\', 221
flush, 220
\_names, 220
names, 220
properties, 221
FaceInstance, 221
create, 221
X, 192
delete_pseudocolor, 192
delete_truecolor, 192
delete_truecolor_masks, 192
delete_pseudocolor, 192
delete_truecolor, 192
delete_truecolor_masks, 192
XBM, 213
\_decode, 213
delete, 213
encode, 213
XCF, 214
\_decode, 214
\_decode, 215
delete, 214
XFace, 221
delete, 211
decode_header, 222
encode, 222
XWD, 215
delete, 216
\_decode, 216
\_decode, 216
\_decode_header, 216
image
Image.Colortable.image, 174
Image.Layer.image, 180
implode
Simulate.implode, 288
implode_nicely
String.implode_nicely, 123
include, 330
index, 426
\_indices
ADT.Table.table\_indices, 261
Image.Color\_indices, 188
Yp.YpMap\_indices, 259
indices, 354
inflate
APPENDIX H. IMAGE.COLOR COLORS

Gz.inflate, 255
Gz.inflate.inflate, 256
initgroups, 236
insert_id
Mysql.mysql.insert_id, 292
install, 405
INT16, ??
INT32, ??
INT64, ??
INT8, ??
INT_TYPE, ??
intp, 355
introduction, 4
1, 4
2, 4
3, 4
4, 5
5, 5
invert
Image.Image.invert, 150
io, 82
is_absolute_path, 355
is_dir
Stdio.is_dir, 102
is_eq, 383
is_equal, 383
is_file
Stdio.is_file, 102
is_ge, 384
is_gt, 384
is_le, 384
is_link
Stdio.is_link, 102
is_lt, 384
is_partial
MIME.Message.is_partial, 284
IS_ZERO, 383
J
JPEG
Image.JPG, 216
kill, 355
Process.create_process.kill, 241
Process.Spawn.kill, 244
L
l_sizeof
Simulate.l_sizeof, 289
latitude
Geographical.Position.latitude, 313
lay
Image.lay, 136
Layer
Image.Layer, 180
layers, 408
lesser
Calendar.time_unit.lesser, 306
light
Image.Color.Color.light, 189
limit
ADT.Table.table.limit, 262
line, 330
Image.Image.line, 151
list_dbs
Mysql.mysql.list_dbs, 293
list_fields
Mysql.mysql.list_fields, 293
list_keys
Yabu.table.list_keys, 270
list_processes
Mysql.mysql.list_processes, 294
list_tables
Mysql.mysql.list_tables, 294
Yabu.db.list_tables, 268
listen_fd
Stdio.Port.listen_fd, 95
listp
Simulate.listp, 290
load
Image.load, 136
load
Image.Font.load, 185
Image.load, 136
load_layer
Image.load_layer, 136
load_module, 356
Locale
Gettext, 301
bindtextdomain, 301
degettext, 302
dgettext, 302
ggettext, 302
localeconv, 303
setlocale, 304
textdomain, 305
localeconv
Locale.Gettext.localeconv, 303
localtime, 356
_locate_references, 334
lock
Thread.Mutex.lock, 115
log, 357
login
Protocols.LysKOM.Session
.login, 229
logout
Protocols.LysKOM.Session
.logout, 229
longest_ordered_sequence
Array.longest_ordered_sequence, 129
longitude
Geographical.Position.longitude, 313
low_mapping_lookup, 391
low_mapping_string_lookup, 391
lower_case, 357
lvalues, 60
LysKOM
Protocols.LysKOM, 227
M
m_delete, 358
m_ind_types, 390
m_indices
Simulate.m_indices, 288
m_sizeof, 390
Simulate.m_sizeof, 288
m_val_types, 390
m_values
Simulate.m_values, 288
make_ascii
Image.Image.make_ascii, 151
make_shared_binary_string, 385
make_shared_binary_string1, 387
make_shared_binary_string2, 387
make_shared_string, 385
make_shared_string1, 387
make_shared_string2, 387
map, 357
ADT.Table.table.map, 263
Array.map, 130
Image.Colortable.map, 174
Yp.YpDomain.map, 258
Yp.YpMap.map, 259
map_array
Simulate.map_array, 288
map_closest
Image.Image.map_closest, 152
map_delete, 390
map_fast
Image.Image.map_fast, 152
map_fs
Image.Image.map_fs, 152
map_regexp
Simulate.map_regexp, 287
mapping_get_item_ptr, 390
mapping_indices, 391
mapping_insert, 390
MAPPING_LOOP, 390
mapping_replace, 391
mapping_string_insert, 391
mapping_to_array, 391
mapping_values, 391
mappingp, 358
master, 358
pike, 379

match
  Image.Image.match, 152
  Regexp.match, 245
  Yp.YpDomain.match, 257
  Yp.YpMap.match, 258

Math, 308
  Matrix, 308
    *, 309
    +, 308
    -, 310
    **, 309
    " +, 308
    ",, 310
    " x, 309
    ' x, 309
  add, 308
  cast, 308
  create, 308
  cross, 309
  mult, 309
  norm, 309
  norm2, 309
  normv, 309
  sub, 310
  transpose, 310

Matrix
  Math.Matrix, 308

max, 358
  Image.Image.max, 139

member_array
  Simulate.member_array, 286

_memory_usage, 334

Message
  MIME.Message, 279

MIME, 272
  decode, 273
  decode_base64, 274
  decode_qp, 274
  decode_uue, 274
  decode_word, 274
  encode, 275
  encode_base64, 275
  encode_qp, 276
  encode_uue, 276
  encode_word, 276
  generate_boundary, 277
  guess_subtype, 277

Message
  body_parts, 280
  boundary, 280
  cast, 282
  charset, 280
  create, 283
  disp_params, 280
  disposition, 280
  get_filename, 284
  getdata, 283
  getencoded, 283
  headers, 281
  is_partial, 284
  params, 281
  setboundary, 284
  setcharset, 284
  setdata, 285
  setdisp_param, 285
  setencoding, 285
  setparam, 285
  subtype, 281
  transfer_encoding, 282
  type, 282
  parse_headers, 277
  quote, 278
  reconstruct_partial, 278
  tokenize, 278

min, 359
  Image.Image.min, 139

mirrorx
  Image.Image.mirrorx, 153

mirrory
  Image.Image.mirrory, 153

misc, 74
mkdir, 359
mkdirhier
Stdio.mkdirhier, 102
mklist
Simulate.mklist, 290
mkmapping, 359
mkmultiset, 360
mktime, 360
mode
Image.Layer.mode, 181
modifiers, 71
modify_by_intensity
Image.Image.modify_by_intensity, 154
modules, 77
mpz
Gmp.mpz, 246
mult
Math.Matrix.mult, 309
multisetp, 360
Mutex
Thread.Mutex, 114
mv, 361
mysql
Mysql.mysql, 291
Mysql, 290
mysql, 291
affected_rows, 291
big_query, 291
binary_data, 291
create, 291
create_db, 292
drop_db, 292
error, 292
host_info, 292
insert_id, 292
list_dbs, 293
list_fields, 293
list_processes, 294
list_tables, 294
protocol_info, 294
reload, 294
select_db, 295
server_info, 295
shutdown, 295
statistics, 295
N
name
Image.Color.Color.name, 190
_names
Image.TTF.Face_names, 220
names
Image.TTF.Face.names, 220
neon
Image.Color.Color.neon, 189
netscape_loop_block
Image.GIF.netscape_loop_block, 200
new, 361
_next, 334
next
Calendar.time_unit.next, 306
next_object, 361
nextkey
Gdbm.gdbm.nextkey, 249
nondither
Image.Colortable.nodither, 176
noise
Image.Image.noise, 154
norm
Math.Matrix.norm, 309
norm2
Math.Matrix.norm2, 309
normv
Math.Matrix.normv, 309
O
object_program, 362
object_variablep, 362
objectp, 362
oldpike_info, 406
open
Stdio.File.open, 84
open_socket
  Stdio.File.open_socket, 91
openlog, 236
operators, 51
order
  Yp.YpDomain.order, 258
  Yp.YpMap.order, 259
ordered
  Image.Colortable.ordered, 176
orient
  Image.Image.orient, 155
orient4
  Image.Image.orient4, 155
outline
  Image.Image.outline, 155
outline_mask
  Image.Image.outline_mask, 155
PCX
params
  MIME.Message.params, 281
parse
  Calendar.Gregorian.Year.parse, 307
parse_headers
  MIME.parse_headers, 277
Parser, 317
SGML, 317
  create, 318
  feed, 318
  finish, 318
  result, 318
paste
  Image.Image.paste, 156
paste_alpha
  Image.Image.paste_alpha, 157
paste_alpha_color
  Image.Image.paste_alpha_color, 157
paste_mask
  Image.Image.paste_mask, 158
PI
params
  MIME.Message.params, 281
parse
  Calendar.Gregorian.Year.parse, 307
parse_headers
  MIME.parse_headers, 277
Parser, 317
SGML, 317
  create, 318
  feed, 318
  finish, 318
  result, 318
paste
  Image.Image.paste, 156
paste_alpha
  Image.Image.paste_alpha, 157
paste_alpha_color
  Image.Image.paste_alpha_color, 157
paste_mask
  Image.Image.paste_mask, 158
PCX
permute
  Array.permute, 130
permute
  Stdio.perror, 102
phaseh
  Image.Image.phaseh, 158
phaseshv
  Image.Image.phaseshv, 158
phasesv
  Image.Image.phasesv, 158
phasesvh
  Image.Image.phasesvh, 158
PI
  Simulate.PI, 287
pid
  Process.create_process.pid, 241
  Process.Spawn.pid, 244
pike
  master.pike, 379
pike_random
  Crypto.randomness.pike_random, 312
pipe
  Stdio.File.pipe, 92
PNG
  Image.PNG, 208
PNM
  Image.PNM, 210
Poly
  Image.Poly, 186
polyfill
  Image.Image.polyfill, 159
popen
  Process.popen, 241
Port
  Stdio.Port, 94
Position
  Geographical.Position, 313
post_url
  Protocols.HTTP.post_url, 224
post_url_data
  Protocols.HTTP.post_url_data, 224
post_url_nice
  Protocols.HTTP.post_url_nice, 224
pow, 362
powm
  Gmp.mpz.powm, 247
pragma, 331
preface, 2
preprocessor, 329
prev
  Calendar.time_unit.prev, 306
  .prev, 334
previous_object
  Simulate.previous_object, 286
printf
  Stdio.FILE.printf, 93
probably_prime_p
  Gmp.mpz.probably_prime_p, 247
Process, 239
  create_process, 239
  kill, 241
  pid, 241
  set_priority, 240
status, 241
wait, 241
exec, 243
exece, 242
popen, 241
spawn, 242
Spawn, 243
  fd, 243
  kill, 244
  pid, 244
stderr, 243
stdin, 243
stdout, 243
wait, 244
system, 242
program, 362
programs, 43
properties
  Image.TTF.Face.properties, 221
protocol_info
  Mysql.mysql.protocol_info, 294
Protocols, 222
DNS, 232
  client, 232
  create, 232
  get_primary_mx, 232
  gethostbyaddr, 232
  gethostbyname, 232
HTTP, 223
  delete_url, 223
  get_url, 223
  get_url_data, 223
  get_url_nice, 223
  http_encode_query, 319
  http_encode_string, 319
  post_url, 224
  post_url_data, 224
  post_url_nice, 224
  put_url, 224
Query, 225
  ., 227
  async_request, 225
  cast, 225
data, 226
downloaded_bytes, 226
  set_callbacks, 225
  thread_request, 226
total_bytes, 226
unentity, 224
LysKOM, 227
Connection, 230
  async_/call/, 230
  async_cb_/call/, 230
  /call/, 230
create, 230
Request, 230
_request, 231
', 231
async, 231
_async, 231
reply, 231
_reply, 231
_sync, 231
Session, 227
create, 227
create_person, 228
create_text, 228
login, 229
logout, 229
send_message, 229
try_complete_person, 229

PSD
Image.PSD, 212

purge
Yabu.db.purge, 269
Yabu.table.purge, 271
push_array_items, 388

put_url
Protocols.HTTP.put_url, 224
putenv, 362

Q
Query
Protocols.HTTP.Query, 225
query_address
Stdio.File.query_address, 91
Stdio_UDP.query_address, 98
query_block_size, 297
query_blocksize
Crypto.rsa.query_blocksize, 300
query_close_callback
Stdio.File.query_close_callback, 90
query_digest_size, 299
query_host_name, 363
Simulate.query_host_name, 290
query_id
Stdio.File.query_id, 89
Stdio.Port.query_id, 96
query_key_length, 297
query_num_arg, 363
query_read_callback
Stdio.File.query_read_callback, 89
query_write_callback
Stdio.File.query_write_callback, 89

Queue
Thread.Queue, 118
quote
MIME.quote, 278

R
random, 363
Image.Image.random, 159
random_seed, 363
randomcube
Image.Colortable.randomcube, 177
randomgrey
Image.Colortable.randomgrey, 177
Image.Image.randomgrey, 159
randomness
Crypto.randomness, 312
rc4
Crypto.rc4, 298
read
Crypto.randomness.arcfour_random
.read, 313
Crypto.randomness.pike_random
.read, 312
Stdio.File.read, 84
Stdio_UDP.read, 97
Thread.Fifo.read, 118
Thread.Queue.read, 119
read_bytes
Stdio.read_bytes, 103
read_file
Stdio.read_file, 103
read_lsb_grey
Image.Image.read_lsb_grey, 160
read_lsb_rgb
Image.Image.read_lsb_rgb, 160
read_oob
Stdio.File.read_oob, 85
Readline
Stdio.Readline, 100
readline
Stdio.readline, 103
readlink, 237
reconstruct_partial
MIME.reconstruct_partial, 278
recursive_rm
Stdio.recursive_rm, 103
reduce
Array.reduce, 130
Image.Colortable.reduce, 178
reduce_fs
Image.Colortable.reduce_fs, 178
.refs, 334
Regexp, 244
create, 245
match, 245
split, 245
register_program, 394
reload
Mysql.mysql.reload, 294
remove
ADT.Table.table.remove, 264
remove_call_out, 364
remove_include_path, 364
remove_module_path, 364
remove_program_path, 364
rename
ADT.Table.table.rename, 262
.render_block
Image.GIF.render_block, 203
render_block
Image.GIF.render_block, 200
reorganize
Gdbm.gdbm.reorganize, 250
replace, 365
replace_master, 365
reply
Protocols.LysKOM.Request
.Request.reply, 231
_reply
Protocols.LysKOM.Request
.Request_reply, 231
_Request
Protocols.LysKOM.Request
_Request, 231
Request
Protocols.LysKOM.Request, 230
result
Parser.SGML.result, 318
reverse, 365
ADT.Table.table.reverse, 262
rgb
Image.Color.rgb, 189
Image.Color.rgb, 188
rgb_to_hsv
Image.Image.rgb_to_hsv, 149
rigid
Image.Colortable.rigid, 179
rint, 366
rm, 366
rollback
Yabu.transaction.rollback, 272
rotate
Image.Image.rotate, 160
rotate_ccw
Image.Image.rotate_ccw, 161
rotate_cw
Image.Image.rotate_cw, 161
rotate_expand
Image.Image.rotate_expand, 160
round, 366
row
  ADT.Table.table.row, 264
rows, 366
rreduce
  Array.reduce, 131
rsa
  Crypto.rsa, 299
rsort
  ADT.Table.table.sort, 262
rusage, 366
S
  Image.Color.Color.s, 191
scale
  Image.Image.scale, 162
search, 367
search_array
  Array.search_array, 131
seek
  Stdio.File.seek, 86
select
  ADT.Table.table.select, 263
select_colors
  Image.Image.select_colors, 152
select_db
  Mysql.mysql.select_db, 295
select_from
  Image.Image.select_from, 162
send
  Stdio.UDP.send, 97
send_message
  Protocols.LysKOM.Session
  .send_message, 229
sendfile
  Stdio.sendfile, 104
server
  Yp.YpDomain.server, 258
  Yp.YpMap.server, 259
server_info
  Mysql.mysql.server_info, 295
Session
  Protocols.LysKOM.Session,
  227
set
  Thread.thread_local.thread_local
  .set, 119
  Yabu.table.set, 269
set_alpha_value
  Image.Layer.set_alpha_value,
  180
set_blocking
  Stdio.File.set_blocking, 89
set_buffer
  Stdio.File.set_buffer, 87
set_callbacks
  Protocols.HTTP.Query.set_callbacks,
  225
set_close_callback
  Stdio.File.set_close_callback, 88
set_close_on_exec
  Stdio.File.set_close_on_exec, 92
set_decrypt_key, 297
set_encrypt_key, 297
set_fill
  Image.Layer.set_fill, 183
set_id
  Stdio.File.set_id, 89
  Stdio.Port.set_id, 95
set_image
  Image.Layer.set_image, 180
set_misc_value
  Image.Layer.set_misc_value,
  183
set_mode
  Image.Layer.set_mode, 181
set_nonblocking
  Stdio.File.set_nonblocking, 87
  Stdio.UDP.set_nonblocking, 97
set_offset
  Image.Layer.set_offset, 183
set_priority
Process.create_process.set_priority, 240
set_private_key
Crypto.rsa.set_private_key, 299
set_public_key
Crypto.rsa.set_public_key, 299
set_read_callback
Stdio.File.set_read_callback, 88
Stdio.UDP.set_read_callback, 98
set_tiled
Image.Layer.set_tiled, 183
set_write_callback
Stdio.File.set_write_callback, 88
set_xspacing_scale
Image.Font.set_xspacing_scale, 185
set_yspacing_scale
Image.Font.set_yspacing_scale, 185
setboundary
MIME.Message.setboundary, 284
setcharset
MIME.Message.setcharset, 284
setcolor
Image.Image.setcolor, 163
setdata
MIME.Message.setdata, 285
setdisp_param
MIME.Message.setdisp_param, 285
setegid, 237
setencoding
MIME.Message.setencoding, 285
seteuid, 237
setgid, 237
setgroups, 238
setlocale
Locale.Gettext.setlocale, 304
setparam
MIME.Message.setparam, 285
setpixel
Image.Image.setpixel, 163
setuid, 238
SGML
Parser.SGML, 317
sgn, 367
sha_sign
Crypto.rsa.sha_sign, 301
sha_verify
Crypto.rsa.sha_verify, 301
shuffle
Array.shuffle, 131
shutdown
Mysql.mysql.shutdown, 295
sign
Crypto.rsa.sign, 300
signal, 368
Thread.Condition.signal, 117
signame, 368
signum, 368
dillycaps
String.sillycaps, 124
simple_array_index_no_free, 388
simple_mapping_string_lookup, 391
Simulate, 286
add_efun, 289
aggregate_list, 290
all_efuns, 287
explode, 287
filter_array, 287
get_function, 286
implode, 288
l,sizeof, 289
listp, 290
m_indices, 288
m sizeof, 288
m_values, 288
map_array, 288
map_regexp, 287
member_array, 286
mklist, 290
PI, 287
previous_object, 286
query_host_name, 290
strstr, 289
sum, 289
this_function, 286
sin, 368
size
Gmp.mpz.size, 248
Thread.Fifo.size, 118
Thread.Queue.size, 119
sizeof, 369
sizeof
ADT.Table.table._sizeof, 261
Yp.YpMap._sizeof, 259
skewx
Image.Image.skewx, 164
skewx_expand
Image.Image.skewx_expand, 164
skewy
Image.Image.skewy, 164
skewy_expand
Image.Image.skewy_expand, 164
sleep, 369
slice_array, 388
sort, 369
ADT.Table.table.sort, 262
sort_array
Array.sort_array, 132
spacefactors
Image.Colortable.spacefactors, 179
spawn
Process.spawn, 242
Spawn
Process.Spawn, 243
splice
Array.splice, 132
split
Regexp.split, 245
sprintf, 369
sqrt, 373
Gmp.mpz.sqrt, 247
sscanf, 75
Stardate
Calendar.Gregorian.Stardate, 307
stat
Stdio.File.stat, 86
statistics
Mysql.mysql.statistics, 295
status
Process.create_process.status, 241
stderr
Process.Spawn.stderr, 243
Stdio.stderr, 94
stdin
Process.Spawn.stdin, 243
Stdio.stdin, 94
Stdio, 83
append_path, 101
exist, 101
File, 83
assign, 90
close, 84
connect, 91
create, 83
dup, 90
dup2, 90
erro, 87
open, 84
open_socket, 91
pipe, 92
query_address, 91
query_close_callback, 90
query_id, 89
query_read_callback, 89
query_write_callback, 89
read, 84
read_oob, 85
seek, 86
set_blocking, 89
set_buffer, 87
set_close_callback, 88
set_close_on_exec, 92
set_id, 89
set_nonblocking, 87
set_read_callback, 88
set_write_callback, 88
stat, 86
tell, 86
truncate, 86
write, 85
write_oob, 86
FILE, 93
getchar, 93
gets, 93
printf, 93
ungets, 93
file_size, 101
is_dir, 102
is_file, 102
is_link, 102
mkdirhier, 102
perror, 102
Port, 94
accept, 96
bind, 94
create, 95
errno, 96
listen_fd, 95
query_id, 96
set_id, 96
read_bytes, 103
read_file, 103
Readline, 100
create, 100
readline, 103
recursive_rm, 103
sendfile, 104
stdout, 94
stdin, 94
stderr, 94
Terminfo, 98
getFallbackTerm, 98
getTerm, 99
getTermcap, 99
getTerminfo, 99
Termcap, 99
create, 100
tputs, 100
Terminfo, 100
UDP, 96
bind, 96
enable_broadcast, 97
query_address, 98
read, 97
send, 97
set_nonblocking, 97
set_read_callback, 98
werror, 104
write_file, 104
write, 104
Process.Spawn.stdout, 243
Stdio.stdout, 94
store
Gdbm.gdbm.store, 249
STR0, 385
STR1, 385
STR2, 385
strerror, 373
String, 123
capitalize, 124
common_prefix, 124
count, 125
fuzzymatch, 123
implode_nicely, 123
implode, 123
sillycaps, 124
strmult, 125
trim_all_whites, 125
APPENDIX H. IMAGE.COLOR COLORS

trim_whites, 125
width, 125
string_to_unicode, 373
string_to_utf8, 374
stringp, 373
strlen, 374
strmult
   String.strmult, 125
strstr
   Simulate.strstr, 289
sub
   Math.Matrix.sub, 310
subtype
   MIME.Message.subtype, 281
sum
   ADT.Table.table.sum, 263
   Image.Image.sum, 139
   Simulate.sum, 289
sum_arrays
   Array.sum_arrays, 133
sumf
   Image.Image.sumf, 139
symlink, 238
sync
   Gdbm.gdbm.sync, 250
   Protocols.LysKOM.Request._Request.sync, 231
   Yabu.db.sync, 269
   _sync
   Protocols.LysKOM.Request._Request._sync, 231
system
   System, 233
system
   Process.system, 242
T
   table
   Yabu.db.table, 268
Table
   ADT.Table, 260
table-of-contents, 2
tan, 374
tell
   Stdio.File.tell, 86
Termcap
   Stdio.Termino.Termcap, 99
Terminfo
   Stdio.Termino, 98
   Stdio.Termino.Termino, 100
test
   Image.Image.test, 165
text_extents
   Image.Font.text_extents, 185
textdomain
   Locale.Gettext.textdomain, 305
TGA
   Image.TGA, 212
this_function
   Simulate.this_function, 286
this_object, 374
this_thread
   Thread.this_thread, 114
Thread, 112
   all_threads, 114
Condition, 116
   broadcast, 117
   signal, 117
   wait, 117
Fifo, 117
   create, 118
   read, 118
   size, 118
   write, 118
Mutex, 114
   lock, 115
   trylock, 116
Queue, 118
   read, 119
   size, 119
   write, 119
   this_thread, 114
thread_create, 114
thread_local, 119
  thread_local
    get, 119
    set, 119
thread_create
  Thread.thread_create, 114
thread_local
  Thread.thread_local, 119
thread_request
  Protocols.HTTP.Query.thread_request, 226
  threshold
    Image.Image.threshold, 165
throw, 374
TIFF
  Image.TIFF, 218
tiled
    Image.Layer.tiled, 183
time, 375
time_unit
  Calendar.time_unit, 305
TNGDate
  Calendar.Gregorian.Stardate
  TNGDate, 307
tokenize
  MIME.tokenize, 278
total_bytes
  Protocols.HTTP.Query.total_bytes, 226
tputs
  Stdio.Termino.Termcap.tputs, 100
trace, 375
transaction
  Yabu.table.transaction, 271
transfer_encoding
  MIME.Message.transfer_encoding, 282
transpose
  Math.Matrix.transpose, 310
trim_all_whites
  String.trim_all_whites, 125
trim_whites
  String.trim_whites, 125
truncate
  Stdio.File.truncate, 86
try_complete_person
  Protocols.LysKOM.Session
    .try_complete_person, 229
trylock
  Thread.Mutex.trylock, 116
TTF
  Image.TTF, 219
tuned_box
    Image.Image.tuned_box, 166
turbulence
    Image.Image.turbulence, 167
type
  ADT.Table.table.type, 262
  MIME.Message.type, 282
TYPE_FIELD, ??
typeof, 77
types, 31
U
ualarm, 376
UDP
  Stdio.UDP, 96
uLPC, 4
uname, 238
undef, 330
unentity
  Protocols.HTTP.unentity, 224
ungets
  Stdio.FILE.ungets, 93
unicode_to_string, 376
uniq
  Array uniq, 133
update, 298
upper_case, 376
utf8_to_string, 376
V
  values
APPENDIX H.  IMAGE.COLOR COLORS

<table>
<thead>
<tr>
<th>Package</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADT.Table</td>
<td>table_values</td>
<td>261</td>
</tr>
<tr>
<td>Image.Color</td>
<td>values</td>
<td>188</td>
</tr>
<tr>
<td>Yabu.db</td>
<td>values</td>
<td>269</td>
</tr>
<tr>
<td>Yabu.table</td>
<td>values</td>
<td>270</td>
</tr>
<tr>
<td>Yp.YpMap</td>
<td>values</td>
<td>260</td>
</tr>
<tr>
<td>values</td>
<td>verify</td>
<td>377</td>
</tr>
<tr>
<td></td>
<td>Crypto.rsa.verify</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>_verify_internals</td>
<td>335</td>
</tr>
<tr>
<td>version</td>
<td>verify</td>
<td>377</td>
</tr>
<tr>
<td>W</td>
<td>wait</td>
<td>241</td>
</tr>
<tr>
<td></td>
<td>Process.create_process.wait</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process.Spawn.wait</td>
<td>244</td>
</tr>
<tr>
<td></td>
<td>Thread.Condition.wait</td>
<td>117</td>
</tr>
<tr>
<td>error</td>
<td>Stdio.werror</td>
<td>104</td>
</tr>
<tr>
<td>where</td>
<td>ADT.Table.table.where</td>
<td>263</td>
</tr>
<tr>
<td>width</td>
<td>String.width</td>
<td>125</td>
</tr>
<tr>
<td>write</td>
<td>377</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Image.Font.write</td>
<td>186</td>
</tr>
<tr>
<td></td>
<td>Stdio.File.write</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>Thread.Fifo.write</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>Thread.Queue.write</td>
<td>119</td>
</tr>
<tr>
<td>write</td>
<td>file</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>Stdio.write_file</td>
<td>104</td>
</tr>
<tr>
<td>write_rgb</td>
<td>Image.Image.write_rgb</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>Image.Image.write_rgb</td>
<td></td>
</tr>
<tr>
<td>write_oob</td>
<td>Stdio.File.write_oob</td>
<td>86</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>192</td>
</tr>
<tr>
<td>X</td>
<td>Image.X</td>
<td>213</td>
</tr>
<tr>
<td>XBM</td>
<td>Image.XBM</td>
<td>213</td>
</tr>
<tr>
<td>XCF</td>
<td>Image.XCF</td>
<td>214</td>
</tr>
<tr>
<td>XFace</td>
<td>Image.XFace</td>
<td>221</td>
</tr>
<tr>
<td>xoffset</td>
<td>Image.Layer.xoffset</td>
<td>183</td>
</tr>
<tr>
<td>xsizes</td>
<td>Image.Image.xsize</td>
<td>167</td>
</tr>
<tr>
<td></td>
<td>Image.Layer.xsize</td>
<td>183</td>
</tr>
<tr>
<td>XWD</td>
<td>Image.XWD</td>
<td>215</td>
</tr>
<tr>
<td>Y</td>
<td>Yabu, 266</td>
<td></td>
</tr>
<tr>
<td></td>
<td>db</td>
<td></td>
</tr>
<tr>
<td></td>
<td>create</td>
<td>268</td>
</tr>
<tr>
<td></td>
<td>list_tables</td>
<td>268</td>
</tr>
<tr>
<td></td>
<td>purge</td>
<td>269</td>
</tr>
<tr>
<td></td>
<td>sync</td>
<td>269</td>
</tr>
<tr>
<td></td>
<td>table</td>
<td>268</td>
</tr>
<tr>
<td></td>
<td>_values</td>
<td>269</td>
</tr>
<tr>
<td></td>
<td>table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>delete</td>
<td>270</td>
</tr>
<tr>
<td></td>
<td>get</td>
<td>270</td>
</tr>
<tr>
<td></td>
<td>list_keys</td>
<td>270</td>
</tr>
<tr>
<td></td>
<td>purge</td>
<td>271</td>
</tr>
<tr>
<td></td>
<td>set</td>
<td>269</td>
</tr>
<tr>
<td></td>
<td>transaction</td>
<td>271</td>
</tr>
<tr>
<td></td>
<td>_values</td>
<td>270</td>
</tr>
<tr>
<td></td>
<td>transaction</td>
<td></td>
</tr>
<tr>
<td></td>
<td>commit</td>
<td>271</td>
</tr>
<tr>
<td></td>
<td>rollback</td>
<td>272</td>
</tr>
<tr>
<td>Year</td>
<td>Calendar.Gregorian.Year</td>
<td>307</td>
</tr>
<tr>
<td>yoffset</td>
<td>Image.Layer.yoffset</td>
<td>183</td>
</tr>
<tr>
<td>Yp, 256</td>
<td>default_yp_domain</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>YpDomain</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>bind</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>map</td>
<td>258</td>
</tr>
</tbody>
</table>
match, 257
order, 258
server, 258
YpMap, 258
   all, 259
  _indices, 259
   map, 259
match, 258
order, 259
server, 259
  _sizeof, 259
  _values, 260
YpDomain
   Yp.YpDomain, 257
YpMap
   Yp.YpMap, 258
ysize
   Image.Image.ysize, 167
   Image.Layer.ysize, 183
Z
   zero_type, 377