Moscow ML Library Documentation

Version 2.00 of June 2000

Sergei Romanenko, Russian Academy of Sciences, Moscow, Russia
Claudio Russo, Cambridge University, Cambridge, United Kingdom
Peter Sestoft, Royal Veterinary and Agricultural University, Copenhagen, Denmark

This document

This manual describes the Moscow ML library, which includes parts of the SML Basis Library and several extensions. The manual has been generated automatically from the commented signature files.

Alternative formats of this document

Hypertext on the World-Wide Web

The manual is available at `http://www.dina.kvl.dk/~sestoft/mosmllib/` for online browsing.

Hypertext in the Moscow ML distribution

The manual is available for offline browsing at `mosml/doc/mosmllib/index.html` in the distribution.

On-line help in the Moscow ML interactive system

The manual is available also in interactive `mosml` sessions. Type `help "lib";` for an overview of built-in function libraries. Type `help "fromstring";` for help on a particular identifier, such as `fromString`. This will produce a menu of all library structures which contain the identifier `fromstring` (disregarding the lowercase/uppercase distinction):

```
--------------------------------
|   1 | val  Bool.fromString   |
|   2 | val  Char.fromString   |
|   3 | val  Date.fromString   |
|   4 | val  Int.fromString    |
|   5 | val  Path.fromString   |
|   6 | val  Real.fromString   |
|   7 | val  String.fromString |
|   8 | val  Time.fromString   |
|   9 | val  Word.fromString   |
|  10 | val  Word8.fromString  |
--------------------------------
```

Choosing a number from this menu will invoke the help browser on the desired structure, e.g. `Int`.

The Moscow ML home page is `http://www.dina.kvl.dk/~sestoft/mosml.html`

Contents

Module AppleScript

AppleScript -- Apple MacOS scripting

```
type OSAID
type OSAerr = int

exception AppleScriptErr of OSAerr * string

val as_compile    : string -> OSAID
val as_dispose    : OSAID -> unit
val as_run_script : OSAID -> string
val as_run_text   : string -> string
```

These Mac specific functions provide the capability to compile and run AppleScript programs.

The exception AppleScriptErr is raised in the event of an error.

[as_compile str] compiles AppleScript source code text, returning an abstract token of type OSAID. This token may be used to run the script. The token may be used repeatedly until it is returned with as_dispose or until mosml exits.

[as_dispose tok] disposes of the resources associated with the OSAID token so that they may be reused by the AppleScript system. AppleScriptErr is raised upon any attemp to reuse a disposed token.

[as_run_script tok] runs the script associated with the token. This typically involves AppleEvent communication with other programs running on the Mac, or networked Macs. The AppleScript result is returned as a string.

[as_run_text str] compiles and runs the AppleScript source code text, disposing all resources allocated in the process, and returns the AppleScript result as a string.

References:

Inside Macintosh: Interapplication Communication, Chapter 10
AppleScript Language Guide English Edition,
available at http://applescript.apple.com/support.html

Module Array

Array -- SML Basis Library

```
prim_EQtype 'a array

val maxLen   : int

val array    : int * '_a -> '_a array
val tabulate : int * (int -> '_a) -> '_a array
val fromList : '_a list -> '_a array

val length   : 'a array -> int
val sub      : 'a array * int -> 'a
val update   : 'a array * int * 'a  -> unit
val extract  : 'a array * int * int option -> 'a Vector.vector

val copy     : {src: 'a array,  si: int, len: int option,
                dst: 'a array, di: int} -> unit
val copyVec  : {src: 'a vector, si: int, len: int option,
                dst: 'a array, di: int} -> unit

val app      : ('a -> unit) -> 'a array -> unit
val foldl    : ('a * 'b -> 'b) -> 'b -> 'a array -> 'b
val foldr    : ('a * 'b -> 'b) -> 'b -> 'a array -> 'b
val modify   : ('a -> 'a) -> 'a array -> unit

val appi     : (int * 'a -> unit) -> 'a array * int * int option -> unit
val foldli   : (int * 'a * 'b -> 'b) -> 'b -> 'a array * int * int option -> 'b
val foldri   : (int * 'a * 'b -> 'b) -> 'b -> 'a array * int * int option -> 'b
val modifyi  : (int * 'a -> 'a) -> 'a array * int * int option -> unit
```

['ty array] is the type of one-dimensional, mutable, zero-based constant-time-access arrays with elements of type 'ty. Type 'ty array admits equality even if 'ty does not. Arrays a1 and a2 are equal if both were created by the same call to a primitive (array, tabulate, fromList).

Some functions work on a *slice* of an array:

The slice (a, i, SOME n) denotes the subarray a[i..i+n-1]. That is, a[i] is the first element of the slice, and n is the length of the slice. Valid only if 0 <= i <= i+n <= length a.

The slice (a, i, NONE) denotes the subarray a[i..length a-1]. That is, the slice denotes the suffix of the array starting at i. Valid only if 0 <= i <= length a. Equivalent to (a, i, SOME(length a - i)).

slice	meaning	
(a, 0, NONE)	the whole array	a[0..len-1]
(a, 0, SOME n)	a left subarray (prefix)	a[0..n-1]
(a, i, NONE)	a right subarray (suffix)	a[i..len-1]
(a, i, SOME n)	a general slice	a[i..i+n-1]

[maxLen] is the maximal number of elements in an array.

[array(n, x)] returns a new array of length n whose elements are all x. Raises Size if n<0 or n>maxLen.

[tabulate(n, f)] returns a new array of length n whose elements are f 0, f 1, ..., f (n-1), created from left to right. Raises Size if n<0 or n>maxLen.

[fromList xs] returns an array whose elements are those of xs. Raises Size if length xs > maxLen.

[length a] returns the number of elements in a.

[sub(a, i)] returns the i'th element of a, counting from 0. Raises Subscript if i<0 or i>=length a. To make 'sub' infix, use

the declaration
infix 9 sub

[update(a, i, x)] destructively replaces the i'th element of a by x.
Raises Subscript if i<0 or i>=length a.

[extract(a, i, NONE)] returns a vector of the elements a[i..length a-1]
of a. Raises Subscript if i<0 or i>length a.

[extract(a, i, SOME len)] returns a vector of the elements a[i..i+len-1]
of a. Raises Subscript if i<0 or len<0 or i+len>length a or
len>Vector.maxLen.

[copy{src, si, len, dst, di}] destructively copies the slice
(src, si, len) to dst, starting at index di. More precisely:
If len=NONE and n=length src, it copies src[si..n-1] to dst[di..di+n-si].
If len=SOME k, it copies src[si..si+k-1] to dst[di..di+k-1].
Works also if src and dst are the same and the segments overlap.
Raises Subscript if si < 0 or di < 0,
or if len=NONE and di + length src - si > length dst,
or if len=SOME k and k < 0 or si + k > length src or di + k > length dst.

[copyVec{src, si, len, dst, di}] destructively copies the slice
(src, si, len) to dst, starting at index di. More precisely:
If len=NONE and n=length src, it copies src[si..n-1] to dst[di..di+n-si].
If len=SOME k, it copies src[si..si+k-1] to dst[di..di+k-1].
Works also if src and dst are the same and the segments overlap.
Raises Subscript if si < 0 or di < 0,
or if len=NONE and di + length src - si > length dst,
or if len=SOME k and k < 0 or si + k > length src or di + k > length dst.

[foldl f e a] folds function f over a from left to right. That is,
computes f(a[len-1], f(a[len-2], ..., f(a[1], f(a[0], e)) ...)),
where len is the length of a.

[foldr f e a] folds function f over a from right to left. That is,
computes f(a[0], f(a[1], ..., f(a[len-2], f(a[len-1], e)) ...)),
where len is the length of a.

[app f a] applies f to a[j] for j=0,1,...,length a-1.

[modify f a] applies f to a[j] and updates a[j] with the result
f(a[j]) for j=0,1,...,length a-1.

The following iterators generalize the above ones in two ways:

. the index j is also being passed to the function being iterated;
. the iterators work on a slice (subarray) of an array.

[foldli f e (a, i, SOME n)] folds function f over the subarray
a[i..i+n-1] from left to right. That is, computes
f(i+n-1, a[i+n-1], f(..., f(i+1, a[i+1], f(i, a[i], e)) ...)).
Raises Subscript if i<0 or n<0 or i+n > length a.

[foldli f e (a, i, NONE)] folds function f over the subarray
a[i..len-1] from left to right, where len = length a. That is,
computes f(len-1, a[len-1], f(..., f(i+1, a[i+1], f(i, a[i], e)) ...)).
Raises Subscript if i<0 or i > length a.

[foldri f e (a, i, SOME n)] folds function f over the subarray
a[i..i+n-1] from right to left. That is, computes
f(i, a[i], f(i+1, a[i+1], ..., f(i+n-1, a[i+n-1], e) ...)).
Raises Subscript if i<0 or n<0 or i+n > length a.

[foldri f e (a, i, NONE)] folds function f over the subarray
a[i..len-1] from right to left, where len = length a. That is,
computes f(i, a[i], f(i+1, a[i+1], ..., f(len-1, a[len-1], e) ...)).
Raises Subscript if i<0 or i > length a.

[appi f (a, i, SOME n)] applies f to successive pairs (j, a[j]) for
j=i,i+1,...,i+n-1. Raises Subscript if i<0 or n<0 or i+n > length a.

[appi f (a, i, NONE)] applies f to successive pairs (j, a[j]) for
j=i,i+1,...,len-1, where len = length a. Raises Subscript if i<0
or i > length a.

[modifyi f (a, i, SOME n)] applies f to (j, a[j]) and updates a[j]
with the result f(j, a[j]) for j=i,i+1,...,i+n-1. Raises Subscript
if i<0 or n<0 or i+n > length a.

[modifyi f (a, i, NONE)] applies f to (j, a[j]) and updates a[j]
with the result f(j, a[j]) for j=i,i+1,...,len-1. Raises Subscript
if i<0 or i > length a.

Module Array2

Array2 -- SML Basis Library

```
eqtype ’a array

datatype traversal = RowMajor | ColMajor

val array      : int * int * ’_a -> ’_a array
val fromList   : ’_a list list -> ’_a array
val tabulate   : traversal -> int * int * (int * int -> ’_a) -> ’_a array

val dimensions : ’a array -> int * int
val nCols      : ’a array -> int
val nRows      : ’a array -> int

val sub        : ’a array * int * int -> ’a
val update     : ’a array * int * int * ’a -> unit

val row        : ’a array * int -> ’a Vector.vector
val column     : ’a array * int -> ’a Vector.vector

type ’a region = { base : ’a array, row : int, col : int,
                   nrows : int option, ncols : int option}

val copy       : { src : ’a region, dst : ’a array,
                   dst_row : int, dst_col : int } -> unit

val app        : traversal -> (’a -> unit) -> ’a array -> unit
val modify     : traversal -> (’a -> ’a) -> ’a array -> unit
val fold       : traversal -> (’a * ’b -> ’b) -> ’b -> ’a array -> ’b

val appi       : traversal -> (int * int * ’a -> unit) -> ’a region -> unit
val modifyi    : traversal -> (int * int * ’a -> ’a) -> ’a region -> unit
val foldi      : traversal -> (int * int * ’a * ’b -> ’b) -> ’b
                 -> ’a region -> ’b
```

[’ty array] is the type of two-dimensional, mutable, zero-based
constant-time-access arrays with elements of type ’ty.
Type ’ty array admits equality even if ’ty does not. Arrays a1 and a2
are equal if both were created by the same call to one of the
primitives array, fromList, and tabulate.

[traversal] is the type of traversal orders: row major or column major.

[RowMajor] specifies that an operation must be done in row-major
order, that is, one row at a time, from top to bottom, and from
left to right within each row. Row-major traversal visits the
elements of an (m,n)-array with m rows and n columns in this
order:

```
        (0,0), (0,1), (0,2), ..., (0,n-1),
        (1,0), (1,1), (1,2), ..., (1,n-1),
                ...
```

that is, in order of lexicographically increasing (i, j). In
Moscow ML, row-major traversal is usually faster than column-major
traversal.

[ColMajor] specifies that an operation must be done in column-major
order, that is, one column at a time, from left to right, and from
top to bottom within each column. Column-major traversal visits
the elements of an (m,n)-array with m rows and n columns in this
order:

```
        (0,0), (1,0), (2,0), ..., (m-1,0),
        (0,1), (1,1), (2,1), ..., (m-1,1),
                ...
```

that is, in order of lexicographically increasing (j, i).

[array(m, n, x)] returns a new m * n matrix whose elements are all x.
Raises Size if n<0 or m<0.

[fromList xss] returns a new array whose first row has elements

xs1, second row has elements xs2, ..., where xss = [xs1,xs2,...,xsm].
Raises Size if the lists in xss do not all have the same length.

[tabulate RowMajor (m, n, f)] returns a new m-by-n array whose
elements are

```
elements are f(0,0), f(0,1), ..., f(0, n-1),
             f(1,0), f(1,1), ..., f(1, n-1),
                    ...
             f(m-1,0),    ...,    f(m-1, n-1)
```

created in row-major order: f(0,0), f(0,1), ..., f(1,0), f(1,1), ...
Raises Size if n<0 or m<0.

[tabulate ColMajor (m, n, f)] returns a new m-by-n array whose
elements are as above, but created in the column-major order:
f(0,0), f(1,0), ..., f(0, 1), f(1, 1), ... Raises Size if n<0 or m<0.

[dimensions a] returns the dimensions (m, n) of a, where m is the
number of rows and n the number of columns.

[nCols a] returns the number of n of columns of a.

[nRows a] returns the number of m of rows of a.

[sub(a, i, j)] returns the i’th row’s j’th element, counting from 0.
Raises Subscript if i<0 or j<0 or i>=m or j>=n
where (m,n) = dimensions a.

[update(a, i, j, x)] destructively replaces the (i,j)’th element of a
by x. Raises Subscript if i<0 or j<0 or i>=m or j>=n
where (m,n) = dimensions a.

[row (a, i)] returns a vector containing the elements of the ith
row of a. Raises Subscript if i < 0 or i >= height a.

[column (a, j)] returns a vector containing the elements of the jth
column of a. Raises Subscript if j < 0 or j >= width a.

[app RowMajor f a] applies f to the elements a[0,0], a[0,1], ...,
a[0,n-1], a[1,0], ..., a[m-1, n-1] of a, where (m, n) = dimensions a.

[app ColMajor f a] applies f to the elements a[0,0], a[1,0], ...,
a[n-1,0], a[0,1], a[1,1], ..., a[m-1, n-1] of a, where (m, n) =
dimensions a.

[modify RowMajor f a] applies f to the elements a[0,0], a[0,1],
..., a[0,n-1], a[1,0], ..., a[m-1, n-1] of a, updating each element
with the result of the application, where (m, n) = dimensions a.

[modify ColMajor f a] applies f to the elements a[0,0], a[1,0],
..., a[n-1,0], a[0,1], a[1,1], ..., a[m-1, n-1] of a, updating each
element with the result of the application, where (m, n) =
dimensions a.

[fold RowMajor f b a] folds f left-right and top-down over the
elements of a in row-major order. That is, computes

```
     f(a[m-1, n-1], f(a[m-1, n-2], ..., f(a[0,1], f(a[0,0], b)) ...))
```

where (m, n) = dimensions a.

[fold ColMajor f b a] folds f left-right and top-down over the
elements of a in column-major order. That is, computes

```
     f(a[m-1, n-1], f(a[m-2, n-1], ..., f(a[1,0], f(a[0,0], b)) ...))
```

where (m, n) = dimensions a.

The following iterators generalize the above ones in two ways:

* the indexes i and j are also being passed to the function;
* the iterators work on a region (submatrix) of a matrix.

[region] is the type of records { base, row, col, nrows, ncols }
determining the region or submatrix of array base whose upper left
corner has index (row, col).

If nrows = SOME r, then the region has r rows: row, row+1, ..., row+r-1.
If nrows = NONE, then the region extends to the bottom of the matrix.
The field ncols similarly determines the number of columns.

```
A region is valid for an array with dimensions (m, n) if
    (1) either nrows = NONE and 0 <= row <= m
        or nrows = SOME r and 0 <= row <= row + r <= m
and (2) either ncols = NONE and 0 <= col <= n
        or ncols = SOME c and 0 <= col <= col + c <= n.
```

[appi RowMajor f reg] applies f to (i, j, a[i, j]) in order of
lexicographically increasing (i, j) within the region reg. Raises
Subscript if reg is not valid. Note that app tr f a is equivalent
to appi tr (f o #3) {base=a, row=0, col=0, nrows=NONE, ncols=NONE}

[appi ColMajor f reg] applies f to (i, j, a[i, j]) in order of
lexicographically increasing (j, i) within the region reg. Raises
Subscript if reg is not valid.

[modifyi RowMajor f reg)] applies f to (i, j, a[i, j]) in order of
lexicographically increasing (i, j) within the region reg. Raises
Subscript if reg is not valid. Note that modify tr f a is equivalent
to modifyi (f o #3) {base=a, row=0, col=0, nrows=NONE, ncols=NONE}).

[modifyi ColMajor f reg)] applies f to (i, j, a[i, j]) in order of
lexicographically increasing (j, i) within the region reg. Raises
Subscript if reg is not valid.

[foldi RowMajor f b a] folds f over (i, j, a[i, j]) in row-major
order within the region reg, that is, for lexicographically
increasing (i, j) in the region. Raises Subscript if reg is not
valid.

[foldi ColMajor f b a] folds f over (i, j, a[i, j]) in column-major
order within the region reg, that is, for lexicographically
increasing (j, i) in the region. Raises Subscript if reg is not
valid.

[copy { src, dst, dst_row, dst_col }] copies the region determined
by src to array dst such that the upper leftmost corner of src is
copied to dst[dst_row, dst_col]. Works correctly even when src and
dst are the same and the source and destination regions overlap.
Raises Subscript if the src region is invalid, or if src translated
to (dst_row, dst_col) is invalid for dst.

Module Arraysort

Arraysort -- Quicksort for arrays, from SML/NJ library

```
val sort   : ('a * 'a -> order) -> 'a Array.array -> unit
val sorted : ('a * 'a -> order) -> 'a Array.array -> bool
```

[sort ordr arr] sorts array arr in-place, using ordering relation ordr.

[sorted ordr arr] returns true if the elements of array arr is
appear in (weakly) increasing order, according to ordering ordr.

Module BinIO

BinIO -- SML Basis Library

```
type elem   = Word8.word
type vector = Word8Vector.vector
```

Binary input

```
type instream

val openIn       : string -> instream
val closeIn      : instream -> unit
val input        : instream -> vector
val inputAll     : instream -> vector
val inputNoBlock : instream -> vector option
val input1       : instream -> elem option
val inputN       : instream * int -> vector
val endOfStream  : instream -> bool
val lookahead    : instream -> elem option
```

Binary output

```
type outstream

val openOut      : string -> outstream
val openAppend   : string -> outstream
val closeOut     : outstream -> unit
val output       : outstream * vector -> unit
val output1      : outstream * elem -> unit
val flushOut     : outstream -> unit
```

This structure provides input/output functions on byte streams. The functions are state-based: reading from or writing to a stream changes the state of the stream. The streams are buffered: output to a stream may not immediately affect the underlying file or device.

[instream] is the type of state-based byte input streams.

[outstream] is the type of state-based byte output streams.

[elem] is the type Word8.word of bytes.

[vector] is the type of Word8Vector.vector (byte vectors).

BYTE INPUT:

[openIn s] creates a new instream associated with the file named s. Raises Io.Io is file s does not exist or is not accessible.

[closeIn istr] closes stream istr. Has no effect if istr is closed already. Further operations on istr will behave as if istr is at end of stream (that is, will return "" or NONE or true).

[input istr] reads some elements from istr, returning a vector v of those elements. The vector will be empty (size v = 0) if and only if istr is at end of stream or is closed. May block (not return until data are available in the external world).

[inputAll istr] reads and returns the vector v of all bytes remaining in istr up to end of stream.

[inputNoBlock istr] returns SOME(v) if some elements v can be read without blocking; returns SOME("") if it can be determined without blocking that istr is at end of stream; returns NONE otherwise. If istr does not support non-blocking input, raises Io.NonblockingNotSupported.

[input1 istr] returns SOME(e) if at least one element e of istr is

available; returns NONE if istr is at end of stream or is closed; blocks if necessary until one of these conditions holds.

[inputN(istr, n)] returns the next n bytes from istr as a vector, if that many are available; returns all remaining bytes if end of stream is reached before n bytes are available; blocks if necessary until one of these conditions holds.

[endOfStream istr] returns false if any elements are available in istr; returns true if istr is at end of stream or closed; blocks if necessary until one of these conditions holds.

[lookahead istr] returns SOME(e) where e is the next element in the stream; returns NONE if istr is at end of stream or is closed; blocks if necessary until one of these conditions holds. Does not advance the stream.

BYTE OUTPUT:

[openOut s] creates a new outstream associated with the file named s. If file s does not exist, and the directory exists and is writable, then a new file is created. If file s exists, it is truncated (any existing contents are lost).

[openAppend s] creates a new outstream associated with the file named s. If file s does not exist, and the directory exists and is writable, then a new file is created. If file s exists, any existing contents are retained, and output goes at the end of the file.

[closeOut ostr] closes stream ostr; further operations on ostr (except for additional close operations) will raise exception Io.Io.

[output(ostr, v)] writes the byte vector v on outstream ostr.

[output1(ostr, e)] writes the byte e on outstream ostr.

[flushOut ostr] flushes the outstream ostr, so that all data written to ostr becomes available to the underlying file or device.

The functions below are not yet implemented:

[setPosIn(istr, i)] sets istr to the position i. Raises Io.Io if not supported on istr.

[getPosIn istr] returns the current position of istr. Raises Io.Io if not supported on istr.

[endPosIn istr] returns the last position of istr.

[getPosOut ostr] returns the current position in stream ostr. Raises Io.Io if not supported on ostr.

[endPosOut ostr] returns the ending position in stream ostr. Raises Io.Io if not supported on ostr.

[setPosOut(ostr, i)] sets the current position in stream to ostr to i. Raises Io.Io if not supported on ostr.

[mkInstream sistr] creates a state-based instream from the functional instream sistr.

[getInstream istr] returns the functional instream underlying the state-based instream istr.

[setInstream(istr, sistr)] redirects istr, so that subsequent input is taken from the functional instream sistr.

[mkOutstream sostr] creates a state-based outstream from the outstream sostr.

```
[getOutstream ostr] returns the outstream underlying the
state-based outstream ostr.

[setOutstream(ostr, sostr)] redirects the outstream ostr so that
subsequent output goes to sostr.
```

Module Binarymap

Binarymap -- applicative maps as balanced ordered binary trees
From SML/NJ lib 0.2, copyright 1993 by AT&T Bell Laboratories
Original implementation due to Stephen Adams, Southampton, UK

```
type ('key, 'a) dict

exception NotFound

val mkDict    : ('key * 'key -> order) -> ('key, 'a) dict
val insert    : ('key, 'a) dict * 'key * 'a -> ('key, 'a) dict
val find      : ('key, 'a) dict * 'key -> 'a
val peek      : ('key, 'a) dict * 'key -> 'a option
val remove    : ('key, 'a) dict * 'key -> ('key, 'a) dict * 'a
val numItems  : ('key, 'a) dict -> int
val listItems : ('key, 'a) dict -> ('key * 'a) list
val app       : ('key * 'a -> unit) -> ('key,'a) dict -> unit
val revapp    : ('key * 'a -> unit) -> ('key,'a) dict -> unit
val foldr     : ('key * 'a * 'b -> 'b)-> 'b -> ('key,'a) dict -> 'b
val foldl     : ('key * 'a * 'b -> 'b) -> 'b -> ('key,'a) dict -> 'b
val map       : ('key * 'a -> 'b) -> ('key,'a) dict -> ('key, 'b) dict
val transform : ('a -> 'b) -> ('key,'a) dict -> ('key, 'b) dict
```

```
[('key, 'a) dict] is the type of applicative maps from domain type
'key to range type 'a, or equivalently, applicative dictionaries
with keys of type 'key and values of type 'a.  They are implemented
as ordered balanced binary trees.

[mkDict ordr] returns a new, empty map whose keys have ordering
ordr.

[insert(m, i, v)] extends (or modifies) map m to map i to v.

[find (m, k)] returns v if m maps k to v; otherwise raises NotFound.

[peek(m, k)] returns SOME v if m maps k to v; otherwise returns NONE.

[remove(m, k)] removes k from the domain of m and returns the
modified map and the element v corresponding to k.  Raises NotFound
if k is not in the domain of m.

[numItems m] returns the number of entries in m (that is, the size
of the domain of m).

[listItems m] returns a list of the entries (k, v) of keys k and
the corresponding values v in m, in order of increasing key values.

[app f m] applies function f to the entries (k, v) in m, in
increasing order of k (according to the ordering ordr used to
create the map or dictionary).

[revapp f m] applies function f to the entries (k, v) in m, in
decreasing order of k.

[foldl f e m] applies the folding function f to the entries (k, v)
in m, in increasing order of k.

[foldr f e m] applies the folding function f to the entries (k, v)
in m, in decreasing order of k.

[map f m] returns a new map whose entries have form (k, f(k,v)),
where (k, v) is an entry in m.

[transform f m] returns a new map whose entries have form (k, f v),
where (k, v) is an entry in m.
```

Module Binaryset

```
Binaryset -- sets implemented by ordered balanced binary trees
From SML/NJ lib 0.2, copyright 1993 by AT&T Bell Laboratories
Original implementation due to Stephen Adams, Southampton, UK

type 'item set

exception NotFound

val empty        : ('item * 'item -> order) -> 'item set
val singleton    : ('item * 'item -> order) -> 'item -> 'item set
val add          : 'item set * 'item -> 'item set
val addList      : 'item set * 'item list -> 'item set
val retrieve     : 'item set * 'item -> 'item
val peek         : 'item set * 'item -> 'item option
val isEmpty      : 'item set -> bool
val equal        : 'item set * 'item set -> bool
val isSubset     : 'item set * 'item set -> bool
val member       : 'item set * 'item -> bool
val delete       : 'item set * 'item -> 'item set
val numItems     : 'item set ->  int
val union        : 'item set * 'item set -> 'item set
val intersection : 'item set * 'item set -> 'item set
val difference   : 'item set * 'item set -> 'item set
val listItems    : 'item set -> 'item list
val app          : ('item -> unit) -> 'item set -> unit
val revapp       : ('item -> unit) -> 'item set -> unit
val foldr        : ('item * 'b -> 'b) -> 'b -> 'item set -> 'b
val foldl        : ('item * 'b -> 'b) -> 'b -> 'item set -> 'b
val find         : ('item -> bool) -> 'item set -> 'item option
```

['item set] is the type of sets of ordered elements of type 'item. The ordering relation on the elements is used in the representation of the set. The result of combining two sets with different underlying ordering relations is undefined. The implementation uses ordered balanced binary trees.

[empty ordr] creates a new empty set with the given ordering relation.

[singleton ordr i] creates the singleton set containing i, with the given ordering relation.

[add(s, i)] adds item i to set s.

[addList(s, xs)] adds all items from the list xs to the set s.

[retrieve(s, i)] returns i if it is in s; raises NotFound otherwise.

[peek(s, i)] returns SOME i if i is in s; returns NONE otherwise.

[isEmpty s] returns true if and only if the set is empty.

[equal(s1, s2)] returns true if and only if the two sets have the same elements.

[isSubset(s1, s2)] returns true if and only if s1 is a subset of s2.

[member(s, i)] returns true if and only if i is in s.

[delete(s, i)] removes item i from s. Raises NotFound if i is not in s.

[numItems s] returns the number of items in set s.

[union(s1, s2)] returns the union of s1 and s2.

[intersection(s1, s2)] returns the intersectionof s1 and s2.

[difference(s1, s2)] returns the difference between s1 and s2 (that is, the set of elements in s1 but not in s2).

[listItems s] returns a list of the items in set s, in increasing order.

[app f s] applies function f to the elements of s, in increasing order.

[revapp f s] applies function f to the elements of s, in decreasing order.

[foldl f e s] applies the folding function f to the entries of the set in increasing order.

[foldr f e s] applies the folding function f to the entries of the set in decreasing order.

[find p s] returns SOME i, where i is an item in s which satisfies p, if one exists; otherwise returns NONE.

Module Bool

Bool -- SML Basis Library

```
datatype bool = datatype bool

val not       : bool -> bool

val toString   : bool -> string
val fromString : string -> bool option
val scan       : (char, 'a) StringCvt.reader -> (bool, 'a) StringCvt.reader
```

```
   [bool] is the type of Boolean (logical) values: true and false.

   [not b] is the logical negation of b.

   [toString b] returns the string "false" or "true" according as b is
   false or true.

   [fromString s] scans a boolean b from the string s, after possible
   initial whitespace (blanks, tabs, newlines).  Returns (SOME b) if s
   has a prefix which is either "false" or "true"; the value b is the
   corresponding truth value; otherwise NONE is returned.

   [scan getc src] scans a boolean b from the stream src, using the
   stream accessor getc.  In case of success, returns SOME(b, rst)
   where b is the scanned boolean value and rst is the remainder of
   the stream; otherwise returns NONE.
```

Module Byte

Byte -- SML Basis Library

```
val byteToChar      : Word8.word -> Char.char
val charToByte      : Char.char -> Word8.word
val bytesToString   : Word8Vector.vector -> String.string
val stringToBytes   : String.string -> Word8Vector.vector

val unpackStringVec : Word8Vector.vector * int * int option -> string
val unpackString    : Word8Array.array * int * int option -> string
val packString      : Substring.substring * Word8Array.array * int -> unit
```

```
   Conversions between bytes and characters, and between byte vectors
   and strings (character vectors).

   [byteToChar w] is the character corresponding to the byte w.

   [charToByte c] is the byte corresponding to character c.

   [bytesToString v] is the string whose character codes are the bytes
   from vector v.

   [stringToBytes s] is the byte vector of character codes of the string s.

   In Moscow ML, all the above operations take constant time.  That
   is, no copying is done.

   [unpackStringVec (v, i, NONE)] is the string whose character codes are
   the bytes of v[i..length v-1].  Raises Subscript if i<0 or i>length v.
   Equivalent to bytesToString(Word8Vector.extract (v, i, NONE)).

   [unpackStringVec (v, i, SOME n)] is the string whose character codes are
   the bytes of v[i..i+n-1].  Raises Subscript if i<0 or n<0 or i+n>length v.
   Equivalent to bytesToString(Word8Vector.extract (v, i, SOME n)).

   [unpackString (a, i, NONE)] is the string whose character codes are
   the bytes of a[i..length a-1].  Raises Subscript if i<0 or i>length a.
   Equivalent to bytesToString(Word8Array.extract (v, i, NONE)).

   [unpackString (a, i, SOME n)] is the string whose character codes are
   the bytes of a[i..i+n-1].  Raises Subscript if i<0 or n<0 or i+n>length a.
   Equivalent to bytesToString(Word8Array.extract (a, i, SOME n)).

   [packString (ss, a, i)] copies the character codes of substring ss into
   the subarray a[i..i+n-1] where n = Substring.size ss.  Raises Subscript
   if i<0 or i+n > length a.
   Equivalent to Word8Array.copyVec{src=s, si=si, len=SOME n, dst=a, di=i}
   when (s, si, n) = Substring.base ss.
```

Module Callback

Callback -- registering ML values with C, and accessing C values from ML

Registering ML values for access from C code:

```
val register     : string -> 'a -> unit
val unregister   : string -> unit
val isRegistered : string -> bool
```

Accessing C variables and functions from ML:

```
type cptr

val getcptr : string -> cptr
val var     : cptr -> 'b
val app1    : cptr -> 'a1 -> 'b
val app2    : cptr -> 'a1 -> 'a2 -> 'b
val app3    : cptr -> 'a1 -> 'a2 -> 'a3 -> 'b
val app4    : cptr -> 'a1 -> 'a2 -> 'a3 -> 'a4 -> 'b
val app5    : cptr -> 'a1 -> 'a2 -> 'a3 -> 'a4 -> 'a5 -> 'b
```

REGISTERING ML VALUES FOR ACCESS FROM C CODE

This example shows how to register the ML function (fn n => 2*n) so
that it may be called from C code.

```
(0) The ML side registers the function:
       Callback.register "myfun" (fn n => 2*n)

(1) The C side first obtains an ML value pointer:
       valueptr mvp = get_valueptr("myfun");

(2) The C side then uses the ML value pointer to obtain an ML
    value, and uses it:
       callback(get_value(mvp), Val_long(42));
```

Operation (1) involves a callback to ML, and hence may be slow.
Calling get_valueptr may cause the garbage collector to run; hence
other live ML values must be registered as GC roots. The garbage
collector will never move the ML value pointer; hence it need not
be registered as a GC root in the C code.

Operation (2) is very fast. If the garbage collector is invoked
between the call of get_value() and the use of the ML value, then
the value must be registered as a GC root. However, the idiom

```
     callback(get_value(mvp), arg1);
```

is safe provided the evaluation of arg1 does not provoke a garbage
collection (e.g. if arg1 is a variable).

The C function get_valueptr returns NULL if nam is not registered.

The C function get_value returns NULL if nam has been unregistered
(and not reregistered) since mvp was obtained; it raises exception
Fail if mvp itself is NULL. Every access to the ML value from C
code should use the ML valueptr and get_valueptr, otherwise the C
code will not know when the value has been unregistered and
possibly deallocated.

The C functions (in mosml/src/runtime/callback.c)

```
   void registervalue(char* nam, value mlval)
   void unregistervalue(char* nam)
```

can be used just as Callback.register and Callback.unregister.

The C functions

```
   value callbackptr (valueptr mvp, value arg1)
   value callbackptr2(valueptr mvp, value arg1, value arg2)
   value callbackptr3(valueptr mvp, value arg1, value arg2, value arg3)
```

can be used for callback via an ML value pointer; they will raise

exception Fail if the ML function indicated by mvp has been unregistered.

[register nam v] registers the ML value v, so that it can be
retrieved from C code under the name nam. If nam has previously
been registered and then unregistered, it will be reregistered with
the new value. The new value immediately becomes visible to the C
side, both via get_valueptr(nam) and via any ML value pointer
previously obtained for nam. Raises exception Fail if nam has been
registered and not yet unregistered.

[unregister nam] deletes the registration. This prevents C code
from obtaining an ML value pointer for nam and from using an ML
value pointer already obtained (but does not prevent C from
attempting to use a stored ML value previously obtained with the
help of the ML value pointer, which is unsafe anyway). Does
nothing if nam is already unregistered. Raises exception Fail
if nam has never been registered.

[isRegistered nam] returns true if nam has been registered and not
yet unregistered.

ACCESSING REGISTERED C VARIABLES AND FUNCTIONS FROM ML

This example shows how to register the C function

```
   value silly_cfun(value v)
   { return copy_double(42.42 * Double_val(v)); }
```

so that it may be called from ML.

```
(0) The C side registers the function:
       registercptr("mycfun", sillycfun);

(1) The ML side obtains a C pointer and defines an ML function
    via that pointer:
       val sillycfun = app1 (getcptr "mycfun") : real -> real
    The type ascription is needed to ensure any type safety whatsoever.
    Mistakes in the types will lead to crashes, as usual with C.

(2) To the ML side, the new ML function is indistinguishable from
    other ML functions
       val result = sillyfun(3.4)
```

The C function (in mosml/src/runtime/callback.c)

```
     void registercptr(char* nam, void* cptr);
```

is used to register C pointers for access from ML. Only pointers
to static C variables, and C functions, should be registered. There
is no way to unregister a C pointer.

[cptr] is the type of pointers to C variables and C functions.

[getcptr nam] returns a pointer to the C variable or function
registered (by the C side) under the name nam. Raises exception
Fail if the name nam has not been registered.

[var cptr] returns the value of the C variable associated with cptr.

[app1 cptr arg1] applies the C function associated with cptr to arg1.

[app2 cptr arg1 arg2] applies the C function associated with cptr to
(arg1, arg2).

[app3 cptr arg1 arg2 arg3] applies the C function associated with
cptr to (arg1, arg2, arg3).

[app4 cptr arg1 arg2 arg3 arg4] applies the C function associated
with cptr to (arg1, arg2, arg3, arg4).

```
[app5 cptr arg1 arg2 arg3 arg4 arg5] applies the C function
associated with cptr to (arg1, arg2, arg3, arg4, arg5).
```

Module Char

```
Char -- SML Basis Library

type char = char

val minChar : char
val maxChar : char
val maxOrd  : int

val chr     : int  -> char      May raise Chr
val ord     : char -> int
val succ    : char -> char      May raise Chr
val pred    : char -> char      May raise Chr

val isLower     : char -> bool   contains "abcdefghijklmnopqrstuvwxyz"
val isUpper     : char -> bool   contains "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
val isDigit     : char -> bool   contains "0123456789"
val isAlpha     : char -> bool   isUpper orelse isLower
val isHexDigit  : char -> bool   isDigit orelse contains "abcdefABCDEF"
val isAlphaNum  : char -> bool   isAlpha orelse isDigit
val isPrint     : char -> bool   any printable character (incl. #" ")
val isSpace     : char -> bool   contains " \t\r\n\v\f"
val isPunct     : char -> bool   printable, not space or alphanumeric
val isGraph     : char -> bool   (not isSpace) andalso isPrint
val isAscii     : char -> bool   ord c < 128
val isCntrl     : char -> bool   control character

val toLower     : char -> char
val toUpper     : char -> char

val fromString  : string -> char option      ML escape sequences
val toString    : char -> string             ML escape sequences

val fromCString : string -> char option      C escape sequences
val toCString   : char -> string             C escape sequences

val contains    : string -> char -> bool
val notContains : string -> char -> bool

val <       : char * char -> bool
val <=      : char * char -> bool
val >       : char * char -> bool
val >=      : char * char -> bool
val compare : char * char -> order
```

```
[char] is the type of characters.

[minChar] is the least character in the ordering <.

[maxChar] is the greatest character in the ordering <.

[maxOrd] is the greatest character code; equals ord(maxChar).

[chr i] returns the character whose code is i.  Raises Chr if
i<0 or i>maxOrd.

[ord c] returns the code of character c.

[succ c] returns the character immediately following c, or raises
Chr if c = maxChar.

[pred c] returns the character immediately preceding c, or raises
Chr if c = minChar.

[isLower c] returns true if c is a lowercase letter (a to z).

[isUpper c] returns true if c is a uppercase letter (A to Z).

[isDigit c] returns true if c is a decimal digit (0 to 9).
```

[isAlpha c] returns true if c is a letter (lowercase or uppercase).

[isHexDigit c] returns true if c is a hexadecimal digit (0 to 9 or a to f or A to F).

[isAlphaNum c] returns true if c is alphanumeric (a letter or a decimal digit).

[isPrint c] returns true if c is a printable character (space or visible)

[isSpace c] returns true if c is a whitespace character (blank, newline, tab, vertical tab, new page).

[isGraph c] returns true if c is a graphical character, that is, it is printable and not a whitespace character.

[isPunct c] returns true if c is a punctuation character, that is, graphical but not alphanumeric.

[isCntrl c] returns true if c is a control character, that is, if not (isPrint c).

[isAscii c] returns true if 0 <= ord c <= 127.

[toLower c] returns the lowercase letter corresponding to c, if c is a letter (a to z or A to Z); otherwise returns c.

[toUpper c] returns the uppercase letter corresponding to c, if c is a letter (a to z or A to Z); otherwise returns c.

[contains s c] returns true if character c occurs in the string s; false otherwise. The function, when applied to s, builds a table and returns a function which uses table lookup to decide whether a given character is in the string or not. Hence it is relatively expensive to compute val p = contains s but very fast to compute p(c) for any given character.

[notContains s c] returns true if character c does not occur in the string s; false otherwise. Works by construction of a lookup table in the same way as the above function.

[fromString s] attempts to scan a character or ML escape sequence from the string s. Does not skip leading whitespace. For instance, fromString "\\065" equals #"A".

[toString c] returns a string consisting of the character c, if c is printable, else an ML escape sequence corresponding to c. A printable character is mapped to a one-character string; bell, backspace, tab, newline, vertical tab, form feed, and carriage return are mapped to the two-character strings "\\a", "\\b", "\\t", "\\n", "\\v", "\\f", and "\\r"; other characters with code less than 32 are mapped to three-character strings of the form "\\^Z", and characters with codes 127 through 255 are mapped to four-character strings of the form "\\ddd", where ddd are three decimal digits representing the character code. For instance,

```
          toString #"A"      equals "A"
          toString #"\\"     equals "\\\\"
          toString #"\""     equals "\\\""
          toString (chr   0) equals "\\^@"
          toString (chr   1) equals "\\^A"
          toString (chr   6) equals "\\^F"
          toString (chr   7) equals "\\a"
          toString (chr   8) equals "\\b"
          toString (chr   9) equals "\\t"
          toString (chr  10) equals "\\n"
          toString (chr  11) equals "\\v"
          toString (chr  12) equals "\\f"
          toString (chr  13) equals "\\r"
          toString (chr  14) equals "\\^N"
          toString (chr 127) equals "\\127"
          toString (chr 128) equals "\\128"
```

[fromCString s] attempts to scan a character or C escape sequence from the string s. Does not skip leading whitespace. For instance, fromString "\\065" equals #"A".

[toCString c] returns a string consisting of the character c, if c is printable, else an C escape sequence corresponding to c. A printable character is mapped to a one-character string; bell, backspace, tab, newline, vertical tab, form feed, and carriage return are mapped to the two-character strings "\\a", "\\b", "\\t", "\\n", "\\v", "\\f", and "\\r"; other characters are mapped to four-character strings of the form "\\ooo", where ooo are three octal digits representing the character code. For instance,

```
          toString #"A"      equals "A"
          toString #"A"      equals "A"
          toString #"\\"     equals "\\\\"
          toString #"\""     equals "\\\""
          toString (chr   0) equals "\\000"
          toString (chr   1) equals "\\001"
          toString (chr   6) equals "\\006"
          toString (chr   7) equals "\\a"
          toString (chr   8) equals "\\b"
          toString (chr   9) equals "\\t"
          toString (chr  10) equals "\\n"
          toString (chr  11) equals "\\v"
          toString (chr  12) equals "\\f"
          toString (chr  13) equals "\\r"
          toString (chr  14) equals "\\016"
          toString (chr 127) equals "\\177"
          toString (chr 128) equals "\\200"
```

[<]
[<=]
[>]
[>=] compares character codes. For instance, c1 < c2 returns true if ord(c1) < ord(c2), and similarly for <=, >, >=.

[compare(c1, c2)] returns LESS, EQUAL, or GREATER, according as c1 is precedes, equals, or follows c2 in the ordering Char.< .

Module CharArray

CharArray -- SML Basis Library

```
eqtype array
type elem   = Char.char
type vector = CharVector.vector

val maxLen   : int

val array    : int * elem -> array
val tabulate : int * (int -> elem) -> array
val fromList : elem list -> array

val length   : array -> int
val sub      : array * int -> elem
val update   : array * int * elem -> unit
val extract  : array * int * int option -> vector

val copy     : {src: array,  si: int, len: int option,
                dst: array, di: int} -> unit
val copyVec  : {src: vector, si: int, len: int option,
                dst: array, di: int} -> unit

val app      : (elem -> unit) -> array -> unit
val foldl    : (elem * 'b -> 'b) -> 'b -> array -> 'b
val foldr    : (elem * 'b -> 'b) -> 'b -> array -> 'b
val modify   : (elem -> elem) -> array -> unit

val appi     : (int * elem -> unit) -> array * int * int option -> unit
val foldli   : (int * elem * 'b -> 'b) -> 'b -> array * int * int option -> 'b
val foldri   : (int * elem * 'b -> 'b) -> 'b -> array * int * int option -> 'b
val modifyi  : (int * elem -> elem) -> array * int * int option -> unit
```

```
   [array] is the type of one-dimensional, mutable, zero-based
   constant-time-access arrays with elements of type Char.char, that
   is, characters.  Arrays a1 and a2 are equal if both were created by
   the same call to a primitive, or if both are empty.

   All operations are as for Array.array.
```

Module CharVector

CharVector -- SML Basis Library

```
type vector = string
type elem = Char.char

val maxLen   : int

val fromList : elem list -> vector
val tabulate : int * (int -> elem) -> vector

val length   : vector -> int
val sub      : vector * int -> elem
val extract  : vector * int * int option -> vector
val concat   : vector list -> vector

val app      : (elem -> unit) -> vector -> unit
val map      : (elem -> elem) -> vector -> vector
val foldl    : (elem * 'b -> 'b) -> 'b -> vector -> 'b
val foldr    : (elem * 'b -> 'b) -> 'b -> vector -> 'b

val appi     : (int * elem -> unit) -> vector * int * int option -> unit
val mapi     : (int * elem -> elem) -> vector * int * int option -> vector
val foldli   : (int * elem * 'b -> 'b) -> 'b -> vector*int*int option -> 'b
val foldri   : (int * elem * 'b -> 'b) -> 'b -> vector*int*int option -> 'b
```

```
   [vector] is the type of one-dimensional, immutable, zero-based
   constant-time-access vectors with elements of type Char.char, that
   is, characters.  Type vector admits equality, and vectors v1 and v2
   are equal if they have the same length and their elements are
   equal.  The type vector is the same as String.string.

   All operations are as for Vector.vector.
```

Module CommandLine

CommandLine -- SML Basis Library

```
val name      : unit -> string
val arguments : unit -> string list
```

```
   [name ()] returns the name used to start the current process.

   [arguments ()] returns the command line arguments of the current process.
   Hence List.nth(arguments (), 0) is the first argument.
```

Module Date

Date -- SML Basis Library

```
datatype weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun

datatype month = Jan | Feb | Mar | Apr | May | Jun
               | Jul | Aug | Sep | Oct | Nov | Dec

type date

exception Date

val date : {
             year   : int,             e.g. 1999
             month  : month,           Jan, Feb, ...
             day    : int,             1-31
             hour   : int,             0-23
             minute : int,             0-59
             second : int,             0-61, permitting leap seconds
             offset : Time.time option  time zone west of UTC
           } -> date

val year       : date -> int
val month      : date -> month
val day        : date -> int
val hour       : date -> int
val minute     : date -> int
val second     : date -> int
val weekDay    : date -> weekday
val yearDay    : date -> int
val isDst      : date -> bool option
val offset     : date -> Time.time option

val compare    : date * date -> order

val toString   : date -> string
val fmt        : string -> date -> string
val fromString : string -> date option
val scan       : (char, 'a) StringCvt.reader -> (date, 'a) StringCvt.reader

val fromTimeLocal : Time.time -> date
val fromTimeUniv  : Time.time -> date
val toTime        : date -> Time.time
val localOffset   : unit -> Time.time
```

```
   These functions convert times to dates and vice versa, and format
   and scan dates.

   [date] is the type of points in time in a given time zone.  If the
   offset is NONE, then the date is in the local time zone.  If the
   offset is SOME t, then t is the offset of the main timezone
   (ignoring daylight savings time) west of UTC.
        When 0 hours <= t < 12 hours, the represented time is to the
   west of UTC and the local time is UTC-t.
        When 12 hours <= t < 23 hours, the represented time is to the
   East of UTC and the local time is UTC+(24-t).

   [date { year, month, day, hour, minute, second, offset }] returns a
   canonical date value.  Seconds outside the range 0..59 are
   converted to the equivalent minutes and added to the minutes
   argument; leap seconds are ignored.  Similarly, excess minutes are
   converted to hours, hours to days, days to months, and months to
   years.  Then the weekday and day number in the year are computed.
   Leap years are assumed in accordance with the Gregorian calendar,
   for any year after year 0 A.D.

   If the offset is greater than one day (24 hours), then the excess
   days are added to the days, and the offset modulo 24 hours is used.
```

[year dt] returns the year of dt, e.g. 1999.

[month dt] returns the month of dt.

[day dt] returns the day of dt

[hour dt] returns the hour of dt.

[minute dt] returns the minute of dt.

[second dt] returns the second of dt.

[weekDay dt] returns the weekday of dt.

[yearDay dt] returns the number of the day in the year of dt. January 1 is day 0, and December 31 is day 364 (and 365 in leap years).

[isDst dt] returns SOME(true) if daylight savings time is in effect at the date dt; returns SOME(false) if not; and returns NONE if this information is unavailable.

[offset dt] returns NONE if the date dt is in the local time zone; returns SOME t where t is the offset west of UTC otherwise. Thus SOME(Time.zeroTime) is UTC.

[compare(dt1, dt2)] returns LESS, EQUAL, or GREATER, according as date dt1 precedes, equals, or follows dt2 in time. Lexicographically compares the dates. Ignores timezone offset and DST. Does not detect invalid dates.

[toString dt] returns a 24 character string representing the date dt in the following format:

```
                    Wed Mar  8 19:06:45 1995
```

The result may be wrong if the date is not representable as a Time.time value. Raises Date if dt is an invalid date. Corresponds to the ANSI C function ‘asctime’.

[fmt fmtstr dt] formats the date dt according to the format string fmtstr. The format string has the same meaning as with the ANSI C function ‘strftime’. These ANSI C format codes should work on all platforms:

```
   %a  abbreviated weekday name (e.g. "Mon")
   %A  full weekday name (e.g. "Monday")
   %b  abbreviated month name (e.g. "Oct")
   %B  full month name (e.g. "October")
   %c  date and time (e.g. "Dec  2 06:55:15 1979")
   %d  day of month (01..31)
   %H  hour (00..23)
   %I  hour (01..12)
   %j  day of year (001..366)
   %m  month number (01..12)
   %M  minutes (00..59)
   %p  locale's equivalent of a.m./p.m.
   %S  seconds (00..61, allowing for leap seconds)
   %U  week number (00..53), with Sunday as the first day of week 01
   %w  day of week, with 0 representing Sunday (0..6)
   %W  week number (00..53), with Monday as the first day of week 01
   %x  locale's appropriate date representation
   %y  year of century (00..99)
   %Y  year including century (e.g. 1997)
   %Z  time zone name if it exists; otherwise the empty string
   %%  the percent character
```

Example: The current local date in ISO format (e.g. 1998-04-06) can be obtained by using:

```
     fmt "%Y-%m-%d" (fromTimeLocal (Time.now ()))
```

[fromString s] scans a 24-character date from the string s, after possible initial whitespace (blanks, tabs, newlines). The format of the string must be as produced by toString. The fields isDst and offset in the resulting date will be NONE. No check of the

consistency of the date (weekday, date in the month, ...) is performed.

[scan getc src] scans a 24-character date from the stream src, using the stream accessor getc. Otherwise works as fromString. In case of success, returns SOME(date, rst) where date is the scanned date and rst is the remainder of the stream; otherwise returns NONE.

[fromTimeLocal t] returns the local date at (UTC) time t. The resulting date will have offset = NONE. The fields year, month, day, hour, minute, and second are as expected. The resulting isDst may be NONE if the system cannot determine whether daylight savings time is in effect at the given time. Corresponds to the ANSI C function ‘localtime’.

[fromTimeUniv t] is similar to fromTime, but returns the UTC date at (UTC) time t. The resulting date will have offset = SOME Time.zeroTime. Corresponds to the ANSI C function ‘gmtime’.

[toTime dt] returns the (UTC) time corresponding to the date dt. Uses the isDst time field if it is present (SOME _) and cannot be calculated from the given date. May raise Date if the given date is invalid. Raises Time.Time if the Date cannot be represented as a Time.time value. At least the dates in the interval 1970-2030 can be represented as Time.time values. Corresponds to the ANSI C function ‘mktime’.

[localOffset ()] is the local time zone offset west of UTC. It holds that 0 hours <= localOffset () < 24 hours.

Module Dynarray

Dynarray -- polymorphic dynamic arrays a la SML/NJ library

```
type ’a array

val array    : int * ’_a -> ’_a array
val subArray : ’_a array * int * int -> ’_a array
val fromList : ’_a list * ’_a -> ’_a array
val tabulate : int * (int -> ’_a) * ’_a -> ’_a array
val sub      : ’a array * int -> ’a
val update   : ’_a array * int * ’_a  -> unit
val default  : ’a array -> ’a
val bound    : ’a array -> int
```

```
   [’ty array] is the type of one-dimensional, mutable, zero-based
   unbounded arrays with elements of type ’ty.  Type ’ty array does
   not admit equality.

   [array(n, d)] returns a dynamic array, all of whose elements are
   initialized to the default d.  The parameter n is used as a hint of the
   upper bound on non-default elements.  Raises Size if n < 0.

   [subArray(a, m, n)] returns a new array with the same default
   value as a, and whose values in the range [0,n-m] equal the
   values in a in the range [m,n].  Raises the exception Size if n < m.

   [fromList (xs, d)] returns an array whose first elements are
   those of [xs], and the rest are the default d.

   [tabulate(n, f, d)] returns a new array whose first n elements
   are f 0, f 1, ..., f (n-1), created from left to right, and whose
   remaining elements are the default d.  Raises Size if n < 0.

   [sub(a, i)] returns the i’th element of a, counting from 0.
   Raises Subscript if i < 0.

   [update(a, i, x)] destructively replaces the i’th element of a by x.
   Raises Subscript if i < 0.

   [default a] returns the default value of the array a.

   [bound a] returns an upper bound on the indices of non-default values.
```

Module Dynlib

Dynlib -- dynamic linking with foreign functions

```
type dlHandle
type symHandle

exception Closed

datatype flag = RTLD_LAZY | RTLD_NOW
val dlopen  : { lib : string, flag : flag, global : bool } -> dlHandle
val dlsym   : dlHandle -> string -> symHandle
val dlclose : dlHandle -> unit

val var  : symHandle -> ’b
val app1 : symHandle -> ’a1 -> ’b
val app2 : symHandle -> ’a1 -> ’a2 -> ’b
val app3 : symHandle -> ’a1 -> ’a2 -> ’a3 -> ’b
val app4 : symHandle -> ’a1 -> ’a2 -> ’a3 -> ’a4 -> ’b
val app5 : symHandle -> ’a1 -> ’a2 -> ’a3 -> ’a4 -> ’a5 -> ’b
```

```
   Structure Dynlib provides dynamic loading and calling of C
   functions, using the dlfcn interface.  A dynamic library is a
   collection of symbols (C variables and functions).

   An ML value passed to or returned from a symbol has type ‘value’ as
   defined in src/runtime/mlvalues.h.  The C functions should use the
   macroes defined there to access and produce ML values.  When
   writing a C function, remember that the garbage collector may be
   activated whenever you allocate an ML value.  Also, remember that
   the garbage collector may move values from the young heap to the
   old one, so that a C pointer pointing into the ML heap may need to
   be updated. Use the Push_roots and Pop_roots macroes to achieve
   this.

   [dlHandle] is the type of dynamic library handles.  A dynamic
   library handle is created by opening a dynamic library using
   dlopen.  This will load the library into the runtime system.  The
   dynamic library handle is used for accessing symbols in that
   library.  The library may be closed and removed from the runtime
   system using dlclose.

   The same library may be opened more than once, resulting in
   different library handles.  The physical library will be loaded
   only once, though, and will remain in the runtime system until all
   handles to the library have been closed.

   [symHandle] is the type of symbol handles.  A symbol handle is used
   to access a symbol (variable or function) in the dynamic library,
   using the functions var, app1, app2, ..., app5.  Type safety is the
   responsibility of the programmer; the runtime system performs no
   type checking.  Hence you are advised to add explicit types
   whenever you define an ML function in terms of var, app1, ..., app5.

   How to create a dynamically loadable library
   --------------------------------------------
   Assume file "xyz.c" contains your C functions.

   To compile xyz.c into xyz.o and then create a dynamic library
   libxyz.so from xyz.o:

     Under Linux and OSF/1 (Digital Unix):
        gcc -c -o xyz.o xyz.c
        ld -shared -o libxyz.so xyz.o
     Under Solaris (ignore the warnings from ld):
        gcc -c -o xyz.o xyz.c
        ld -G -B symbolic -z nodefs -o libxyz.so xyz.o
     Under HP-UX:
        gcc -fPIC -c -o xyz.o xyz.c
        ld -b -B symbolic -E -o libxyz.so xyz.o
```

If "xyz.o" depends on another library "libabc.a" you may link the
required functions into libxyz.so just by adding -labc or libabc.a
to the above linker command.

If "xyz.o" depends on another dynamic library "libabc.so" you may
specify this by adding -labc to the above linker command. Then
Dynlib.dlopen will automatically load libabc.so before libxyz.so.

[dlopen { lib, flag, global }] will load and open the library in
file 'lib', returning a handle to it. Libraries are usually
specified just by file name, leaving out the directory path.
Linux/Unix-specific information: Libraries are searched for in
those directories mentioned in LD_LIBRARY_PATH, those mentioned in
/etc/ld.so.cache, in /usr/lib and /lib. (Note that
/etc/ld.so.cache is created from /etc/ld.so.conf by running
ldconfig; you must be superuser to do that).
If 'global' is true, then the library's global symbols are
made available for other libraries subsequently loaded.

[flag] is the type of library loading modes: RTLD_LAZY and RTLD_NOW.

[RTLD_LAZY] specifies that only symbol relocations will be
performed when calling dlopen, whereas function relocations will be
performed later when a function is invoked for the first time (if
ever). This is the normal situation.

[RTLD_NOW] specifies that all function relocations must be
performed immediately, also for functions that will never be
called. This checks that all functions are defined, but may waste
some time.

[dlsym dlh nam] returns a symbol handle for the symbol called 'nam'
in the library associated with dlh. Raises Closed if dlh has been
closed.

[dlclose dlh] closes the library handle and deallocates the library
if there are no more open handles to this library.

The following functions raise Closed if the associated handle has
been closed.

[var sym] returns the value of the C variable associated with sym.

[app1 sym arg1] applies the C function associated with sym to arg1.

[app2 sym arg1 arg2] applies the C function associated with sym to
(arg1, arg2).

[app3 sym arg1 arg2 arg3] applies the C function associated with
sym to (arg1, arg2, arg3).

[app4 sym arg1 arg2 arg3 arg4] applies the C function associated
with sym to (arg1, arg2, arg3, arg4).

[app5 sym arg1 arg2 arg3 arg4 arg5] applies the C function
associated with sym to (arg1, arg2, arg3, arg4, arg5).

Module FileSys

OS.FileSys -- SML Basis Library

```
type dirstream

val openDir   : string -> dirstream
val readDir   : dirstream -> string option
val rewindDir : dirstream -> unit
val closeDir  : dirstream -> unit

val chDir     : string -> unit
val getDir    : unit -> string
val mkDir     : string -> unit
val rmDir     : string -> unit
val isDir     : string -> bool

val realPath  : string -> string
val fullPath  : string -> string
val isLink    : string -> bool
val readLink  : string -> string

val modTime   : string -> Time.time
val setTime   : string * Time.time option -> unit
val remove    : string -> unit
val rename    : {old: string, new: string} -> unit

datatype access = A_READ | A_WRITE | A_EXEC
val access    : string * access list -> bool

val fileSize  : string -> int

val tmpName   : unit -> string

eqtype file_id
val fileId    : string -> file_id
val hash      : file_id -> word
val compare   : file_id * file_id -> order
```

These functions operate on the file system. They raise OS.SysErr
in case of errors.

[openDir p] opens directory p and returns a directory stream for
use by readDir, rewindDir, and closeDir. Subsequent calls to
readDir will return the directory entries in some unspecified
order.

[readDir dstr] returns SOME(s), consuming an entry s from the
directory stream if it is non-empty; returns NONE if it is empty
(when all directory entries have been read). Only entries distinct
from the parent arc and the current arc (that is, .. and . in Unix,
DOS, and Windows; see the Path structure) will be returned.

[rewindDir dstr] resets the directory stream as if it had just been
opened.

[closeDir dstr] closes the directory stream. All subsequent
operations on the stream will raise OS.SysErr.

[chDir p] changes the current working directory to p. This affects
calls to the functions use, load, compile in the interactive
system, as well as all functions defined in this library. If p
specifies a volume name, then this command also changes the current
volume (relevant under DOS, Windows, OS/2, etc.).

[getDir ()] returns the name of the current working directory.

[mkDir p] creates directory p on the file system.

[rmDir p] removes directory p from the file system.

[isDir p] tests whether p is a directory.

[fullPath p] returns a canonical form of path p, where all occurrences of the arcs ".", "..", "" have been expanded or removed, and (under Unix) symbolic links have been fully expanded. Raises SysErr if a directory on the path, or the file or directory named, does not exist or is not accessible, or if there is a link loop.

[realPath p] behaves as fullPath(p) if p is absolute. If p is relative and on the same volume as the current working directory, it returns a canonical path relative to the current working directory, where superfluous occurrences of the arcs ".", "..", "" have been removed, and (under Unix) symbolic links have been fully expanded. Raises SysErr if a directory on the path, or the file or directory named, does not exist or is not accessible, or if there is a link loop. Raises Path if p is relative and on a different volume than the current working directory.

[isLink p] returns true if p names a symbolic link. Raises SysErr if the file does not exist or there is an access violation. On operating systems without symbolic links, it returns false, or raises SysErr if the file does not exist or there is an access violation.

[readLink p] returns the contents of the symbolic link p. Raises SysErr if p does not exist or is not a symbolic link, or there is an access violation. On operating systems without symbolic links, it raises SysErr.

[modTime p] returns the modification time of file p.

[setTime (p, tmopt)] sets the modification and access time of file p. If tmopt is SOME t, then the time t is used; otherwise the current time, that is, Time.now(), is used.

[remove p] deletes file p from the file system.

[rename {old, new}] changes the name of file ‘old’ to ‘new’.

[access] is the type of access permissions:

[A_READ] specifies read access.

[A_WRITE] specifies write access.

[A_EXEC] specifies permission to execute the file (or directory).

[access (p, accs)] tests the access permissions of file p, expanding symbolic links as necessary. If the list accs of required access permission is empty, it tests whether p exists. If accs contains A_READ, A_WRITE, or A_EXEC, respectively, it tests whether the user process has read, write, or execute permission for the file.
    Under Unix, the access test is done with the ‘real’ user id and group id (as opposed to the ‘effective’ user id and group id) of the user process. Hence access("file", [A_READ]) may return false, yet the file may be readable by the process, in case the effective user id or group id has been changed by setuid.

[fileSize p] return the size, in bytes, of the file p. Raises SysErr if p does not exist or its directory is not accessible.

[tmpName ()] returns a file name suitable for creating a fresh temporary file. Note that there is no guarantee that the file name will be unique, since a file of that name may be created between the call to tmpName and a subsequent call to openOut which creates the file. The file name will be absolute, usually of the form /tmp/xxxxxxxx provided by POSIX tmpnam (3).

[file_id] is the type of unique identities of file system objects (including device ids and volume ids, but possibly insensitive to

volume changes on removable volumes, such as tapes and diskettes). The set of file ids is equipped with a total linear order.

[fileId p] returns the file_id of the file system object named by path p. It holds that fileId p1 = fileId p2 if and only if p1 and p2 name the same file system object.

[hash fid] returns a hashvalue for fid, suitable for use in a hashtable of file ids (and hence files).
If fid1 = fid2 then hash fid1 = hash fid2.

[compare (fid1, fid2)] returns LESS, EQUAL, or GREATER, according as fid1 precedes, equals, or follows fid2 in the total linear order on file ids. This is suitable for e.g. an ordered binary tree of file ids (and hence files).

Module Gdbm

Gdbm -- GNU gdbm persistent string hashtables -- requires Dynlib

```
type table

datatype openmode =
    READER                          read-only access (nonexclusive)
  | WRITER                          read/write, table must exist
  | WRCREAT                         read/write, create if necessary
  | NEWDB                           read/write, create empty table

type datum = string

exception NotFound
exception AlreadyThere
exception NotWriter
exception Closed
exception GdbmError of string

val withtable  : string * openmode -> (table -> 'a) -> 'a
val withtables : (string * openmode) list -> (table list -> 'a) -> 'a
val add        : table -> datum * datum -> unit
val insert     : table -> datum * datum -> unit
val find       : table -> datum -> datum
val peek       : table -> datum -> datum option
val hasKey     : table -> datum -> bool
val remove     : table -> datum -> unit
val listKeys   : table -> datum list
val numItems   : table -> int
val listItems  : table -> (datum * datum) list
val app        : (datum * datum -> unit) -> table -> unit
val map        : (datum * datum -> 'a) -> table -> 'a list
val fold       : (datum * datum * 'a -> 'a) -> 'a -> table -> 'a
val fastwrite  : bool ref
val reorganize : table -> unit
```

[table] is the type of an opened table. A value of type table can be used only in the argument f to the withtable function. This makes sure that the table is closed after use.

[openmode] is the type of opening modes. Read-only access (READER) is non-exclusive; read/write access (WRITER, WRCREAT, NEWDB) is exclusive.

[withtable (nam, mod) f] first opens the table db in file nam with mode mod, then applies f to db, then closes db. Makes sure to close db even if an exception is raised during the evaluation of f(db). Raises GdbmError with an informative message in case the table cannot be opened. E.g. the table cannot be opened for reading if already opened for writing, and cannot be opened for writing if already opened for reading.

A table is only guaranteed to work properly if created by withtable using open modes WRCREAT or NEWDB. If you create a table by creating and then opening an empty file, then numItems, listKeys, listItems, etc. will raise an exception.

[withtables nammod f], where nammod = [(nam1, mod1), ..., (namn, modn)], is equivalent to

```
     withtable (nam1, mod1) (fn db1 =>
         withtable (nam2, mod2) (fn db2 =>
             ...
                 f [db1, db2, ...]))
```

That is, first opens the databases db1, db2, ... in that order in files nam1, nam2, ... with modes mod1, mod2, ..., then applies f to [db1, db2, ...], and finally closes [db1, db2, ...]. Makes sure to close all databases even if an exception is raised during the opening of db1, db2, ... or during the evaluation of f[db1, db2, ...].

[add db (k,v)] adds the pair (k, v) to db. Raises AlreadyThere if there is a pair (k, _) in db already. Raises NotWriter if db is not opened in write mode.

[insert db (k, v)] adds the pair (k, v) to db, replacing any pair (k, _) at k if present. Raises NotWriter if db is not opened in write mode.

[find db k] returns v if the pair (k, v) is in db; otherwise raises NotFound.

[peek db k] returns SOME v if the pair (k, v) is in db; otherwise returns NONE.

[hasKey db k] returns true if there is a pair (k, _) in db; otherwise returns false.

[remove db k] deletes the pair (k, _) from the table if present; otherwise raises NotFound. Raises NotWriter if db is not opened in write mode.

[listKeys db] returns a list of all keys in db in an unspecified order.

[numItems db] is the number of (key, value) pairs in db. Equivalent to length(listKeys db).

[listItems db] returns a list of all (key, value) pairs in db in some order. Equivalent to

```
     List.map (fn key => (key, find(db,key))) (listKeys db)
```

[app f db] is equivalent to List.app f (listItems db), provided the function f does not change the set of keys in the table. Otherwise the effect is unpredictable.

[map f db] is equivalent to List.map f (listItems db), provided the function f does not change the set of keys in the table. Otherwise the result and effect are unpredictable.

[fold f a db] is equivalent to

```
     List.foldr (fn ((k, v), r) => f(k, v, r)) a (listItems db)
```

provided the function f does not change the set of keys in the table. Otherwise the result and effect are unpredictable.

[fastwrite] can be set to speed up writes to a table. By default, !fastwrite is false and every write to a table will be followed by file system synchronization. This is safe, but slow if you perform thousands of writes. However, if !fastwrite is true when calling withtable, then writes may not be followed by synchronization, which may speed up writes considerably. In any case, the file system is synchronized before withtable returns.

[reorganize db] has no visible effect, but may be called after a lot of deletions to shrink the size of the table file.

Module Gdimage

Gdimage -- creating PNG images -- requires Dynlib

```
type image

type color

datatype style =
    ColorS of color
  | TransparentS

datatype mode =
    Color of color
  | Transparent
  | Brushed of image
  | Styled of style vector
  | StyledBrushed of bool vector * image
  | Tiled of image

datatype font =
    Tiny
  | Small
  | MediumBold
  | Large
  | Giant

type rgb = int * int * int          RGB color components, 0..255
type xy  = int * int                points (x, y) and sizes (w, h)

val image     : xy -> rgb -> image
val fromPng   : string -> image
val toPng     : image -> string -> unit
val stdoutPng : image -> unit
val size      : image -> xy

val color          : image -> rgb -> color
val rgb            : image -> color -> rgb
val htmlcolors     : image -> { aqua : color, black : color, blue : color,
                                fuchsia : color, gray : color,
                                green : color, lime : color, maroon : color,
                                navy : color, olive : color, purple : color,
                                red : color, silver : color, teal : color,
                                white : color, yellow : color }
val getTransparent : image -> color option
val setTransparent : image -> color -> unit
val noTransparent  : image -> unit

val drawPixel   : image -> mode -> xy -> unit
val drawLine    : image -> mode -> xy * xy -> unit
val drawRect    : image -> mode -> xy * xy -> unit
val fillRect    : image -> mode -> xy * xy -> unit
val drawPolygon : image -> mode -> xy vector -> unit
val fillPolygon : image -> mode -> xy vector -> unit
val drawArc     : image -> mode -> { c : xy, wh : xy, from : int, to : int }
                  -> unit
val fill        : image -> mode -> xy -> unit
val fillBorder  : image -> mode -> xy -> color -> unit

val copy        : { src : image, srcxy : xy, srcwh : xy,
                    dst : image, dstxy : xy } -> unit
val copyResize  : { src : image, srcxy : xy, srcwh : xy,
                    dst : image, dstxy : xy, dstwh : xy } -> unit

val char        : image -> color -> font -> xy -> char -> unit
val charUp      : image -> color -> font -> xy -> char -> unit
val string      : image -> color -> font -> xy -> string -> unit
val stringUp    : image -> color -> font -> xy -> string -> unit
val charsize    : font -> xy
```

This is an interface to version 1.7.3 of Thomas Boutell's gd image package for creating PNG images.

[image] is the type of images being drawn. They can be created from scratch, imported from PNG files, and exported to PNG files.

All functions correctly clip to the actual size of the image.

[color] is the type of colors. Currently there can be at most 256 different colors in an image.

[style] is the type of drawing styles. A style is either a color, or transparent.

```
[mode] is the type of drawing modes for line drawing and filling.
It may be one of
     Color c         where c is a color
     Transparent
     Brushed img     for line drawing using the given image as brush
     Styled stys     for line drawing, cyclically using the styles
                     in the given vector to create a dashed line
     StyledBrushed (vis, img)
                     for line drawing, using the given image as a brush,
                     cyclically switching it on and off according to the
                     given bool vector
     Tiled img       for filling, using the given image as a tile
```

[font] is the type of fonts: Tiny, Small, MediumBold, Large, Giant

[rgb] is the type of (r, g, b) triples, where the components indicate color intensity as an integer value in the range 0..255.

[xy] is the type of pairs, used for (x, y) coordinates and to indicate dimensions (width, height). The origin (0, 0) is the upper left-hand corner of the image. The x coordinates increase to the right; the y coordinates increase downwards.

[image (w, h) rgb] creates a new empty image with size (w, h) and the background color rgb. Raises Fail if the image cannot be created.

[fromPng filename] reads an image from the given PNG file. Raises Fail if the file does not exist or does not contain a PNG image.

[size img] returns (w, h) where w is the width and h the height of img.

[toPng img filename] write the image to the given file in PNG format.

[stdoutPng img] writes the image to standard output in PNG format, preceded by the HTTP header "Content-type: image/png\n\n". Useful in CGI scripts.

[color img rgb] returns the color code corresponding to rgb in the color table of img. Reuses the color code if it has already been allocated; otherwise allocates the color if possible; otherwise returns an approximation to the color rgb.

[htmlcolors im] returns a record containing the 16 standard HTML colors: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow. This call will allocate all these colors in the color table of the image, even if you do not use all of them.

[rgb img color] returns (r, g, b) where r, g, b are the component intensities of the given color in the color table of img.

[getTransparent img] returns SOME c where c is the 'transparent' color of the image, if any; otherwise returns NONE.

[setTransparent img col] makes the given color transparent in the image.

[noTransparent img] makes all colors non-transparent in the image. This is useful for images that are to be used as tiles for filling. Such images are not allowed to have a transparent color.

[drawPixel img mode xy] draws the pixel in img at xy using the given mode.

[drawLine img mode (xy1, xy2)] draws a line in img from xy1 to xy2 using the given mode.

[drawRect img mode (xy1, xy2)] draws a rectangle in img with opposing corners xy1 and xy2 using the given mode.

[fillRect img mode (xy1, xy2)] draws a filled rectangle in img with opposing corners xy1 and xy2 using the given mode.

[drawPolygon img mode xys] draws a polygon in img with corners as given by the vector xys of coordinates using the given mode.

[fillPolygon img mode xys] draws a filled polygon in img with corners as given by the vector xys of coordinates using the given mode.

[drawArc img mode { c, wh, from, to }] draw part of an ellipsis arc in img, with center c, width and height wh, using the given ‘from’ and ‘to’ angles, given in degrees (0..360).

[fill img mode xy] fills the region in img around xy which has the same color as the point at img, using the given mode.

[fillBorder img mode xy col] fills the region in img around xy which is delimited by the color col, using the given mode.

[copy { src, srcxy, srcwh, dst, dstxy }] copies part of the image src into the image dst, without rescaling. More precisely, copies the subimage of src whose upper left-hand corner is srcxy and whose size is srcwh, into the subimage of dst whose upper left-hand corner is dstxy. The images src and dst may be the same, but if the subimages overlap, then the result is unpredictable.

[copyResize { src, srcxy, srcwh, dst, dstxy, dstwh }] copies part of the image src into the image dst, rescaling to the given size dstwh of the destination subimage. Otherwise works as copy.

[char img col font xy ch] draws the character ch left-right (to be read from south) in img at xy using the given color.

[charUp img col font xy ch] draws the character ch bottom-up (to be read from east) in img at xy using the given color.

[string img col font xy s] draws the string s left-right (to be read from south) in img at xy using the given color.

[stringUp img col font xy s] draws the string s bottom-up (to be read from east) in img at xy using the given color.

[charsize font] returns (w, h) where w is the width and h the height, in pixels, of each character in the given font.

Module General

SML Basis Library and Moscow ML top-level declarations

SML Basis Library types

```
type     exn
eqtype   unit
datatype order = LESS | EQUAL | GREATER
```

Additional Moscow ML top-level types

```
datatype bool = false | true
eqtype char
eqtype int
datatype 'a option = NONE | SOME of 'a
type ppstream
eqtype real
eqtype string
type substring
type syserror
type 'a vector
eqtype word
eqtype word8
datatype 'a list = nil | op :: of 'a * 'a list
datatype 'a ref  = ref of 'a
datatype 'a frag = QUOTE of string | ANTIQUOTE of 'a
```

SML Basis Library exceptions

```
exception Bind
exception Chr
exception Div
exception Domain
exception Fail of string
exception Match
exception Overflow
exception Subscript
exception Size
```

Additional Moscow ML top-level exceptions

```
exception Graphic of string
exception Interrupt
exception Invalid_argument of string
exception Io of {function : string, name : string, cause : exn }
exception Out_of_memory
exception SysErr of string * syserror option
```

SML Basis Library values

```
val !          : 'a ref -> 'a
val :=         : 'a ref * 'a -> unit

val o          : ('b -> 'c) * ('a -> 'b) -> ('a -> 'c)
val ignore     : 'a -> unit
val before     : 'a * 'b -> 'a

val exnName    : exn -> string
val exnMessage : exn -> string
```

Additional Moscow ML top-level values

```
val not    : bool -> bool
val ^      : string * string -> string

val =      : ''a * ''a -> bool
val <>     : ''a * ''a -> bool

val ceil   : real -> int                round towards plus infinity
val floor  : real -> int                round towards minus infinity
val real   : int -> real                equals Real.fromInt
val round  : real -> int               round to nearest even
val trunc  : real -> int               round towards zero

val vector : 'a list -> 'a vector
```

Below, numtxt is int, Word.word, Word8.word, real, char, string:

```
val <   : numtxt * numtxt -> bool
val <=  : numtxt * numtxt -> bool
val >   : numtxt * numtxt -> bool
val >=  : numtxt * numtxt -> bool

val makestring : numtxt -> string
```

Below, realint is int or real:

```
val ~   : realint -> realint           raises Overflow
val abs : realint -> realint           raises Overflow
```

Below, num is int, Word.word, Word8.word, or real:

```
val +   : num * num -> num             raises Overflow
val -   : num * num -> num             raises Overflow
val *   : num * num -> num             raises Overflow
val /   : real * real -> real          raises Div, Overflow
```

Below, wordint is int, Word.word or Word8.word:

```
val div : wordint * wordint -> wordint  raises Div, Overflow
val mod : wordint * wordint -> wordint  raises Div
```

[exn] is the type of exceptions.

[unit] is the type containing the empty tuple () which equals the empty record { }.

[order] is used as the return type of comparison functions.

[bool] is the type of booleans: false and true. Equals Bool.bool.

[char] is the type of characters such as #"A". Equals Char.char.

[int] is the type of integers. Equals Int.int.

[option] is the type of optional values. Equals Option.option.

[ppstream] is the type of pretty-printing streams, see structure PP. Pretty-printers may be installed in the top-level by function Meta.installPP; see the Moscow ML Owner's Manual.

[real] is the type of floating-point numbers. Equals Real.real.

[string] is the type of character strings. Equals String.string.

[substring] is the type of substrings. Equals Substring.substring.

[syserror] is the abstract type of system error codes. Equals OS.syserror.

[vector] is the type of immutable vectors. Equals Vector.vector.

[word] is the type of unsigned words. Equals Word.word.

[word8] is the type of unsigned bytes. Equals Word8.word.

['a list] is the type of lists of elements of type 'a. Equals List.list.

['a ref] is the type of mutable references to values of type 'a.

['a frag] is the type of quotation fragments, resulting from the

parsing of quotations ` ... ` and antiquotations. See the Moscow ML Owner's Manual.

[Bind] is the exception raised when the right-hand side value in a valbind does not match the left-hand side pattern.

[Chr] signals an attempt to produce an unrepresentable character.

[Div] signals an attempt to divide by zero.

[Domain] signals an attempt to apply a function outside its domain of definition; such as computing Math.sqrt(~1).

[Fail] signals the failure of some function, usually in the Moscow ML specific library structures.

[Match] signals the failure to match a value against the patterns in a case, handle, or function application.

[Overflow] signals the attempt to compute an unrepresentable number.

[Subscript] signals the attempt to use an illegal index in an array, dynarray, list, string, substring, vector or weak array.

[Size] signals the attempt to create an array, string or vector that is too large for the implementation.

[Graphic] signals the failure of Graphics primitives (DOS only).

[Interrupt] signals user interrupt of the computation.

[Invalid_argument] signals the failure of a function in the runtime system.

[Io { function, name, cause }] signals the failure of an input/output operation (function) when operating on a file (name). The third field (cause) may give a reason for the failure.

[Out_of_memory] signals an attempt to create a data structure too large for the implementation, or the failure to extend the heap or stack.

[SysErr (msg, err)] signals a system error, described by msg. A system error code may be given by err. If so, it will usually hold that msg = OS.errorMsg err.

SML Basis Library values

[! rf] returns the value pointed to by reference rf.

[:=(rf, e)] evaluates rf and e, then makes the reference rf point to the value of e. Since := has infix status, this is usually written
```
     rf := e
```

[o(f, g)] computes the functional composition of f and g, that is, fn x => f(g x). Since o has infix status, this is usually written
```
     f o g
```

[ignore e] evaluates e, discards its value, and returns () : unit.

[before(e1, e2)] evaluates e1, then evaluates e2, then returns the value of e1. Since before has infix status, this is usually written
```
     e1 before e2
```

[exnName exn] returns a name for the exception constructor in exn. Never raises an exception itself. The name returned may be that of any exception constructor aliasing with exn. For instance,
```
     let exception E1; exception E2 = E1 in exnName E2 end
```
may evaluate to "E1" or "E2".

[exnMessage exn] formats and returns a message corresponding to

```
exception exn.  For the exceptions defined in the SML Basis Library,
the message will include the argument carried by the exception.


Additional Moscow ML top-level values

[not b] returns the logical negation of b.

[^] is the string concatenation operator.

[=] is the polymorphic equality predicate.

[<>] is the polymorphic inequality predicate.

[ceil r] is the smallest integer >= r (rounds towards plus infinity).
May raise Overflow.

[floor r] is the largest integer <= r (rounds towards minus infinity).
May raise Overflow.

[real i] is the floating-point number representing integer i.
Equivalent to Real.fromInt.

[round r] is the integer nearest to r, using the default rounding
mode.  May raise Overflow.

[trunc r] is the numerically largest integer between r and zero
(rounds towards zero).  May raise Overflow.

[vector [x1, ..., xn]] returns the vector #[x1, ..., xn].

[< (x1, x2)]
[<=(x1, x2)]
[> (x1, x2)]
[>=(x1, x2)]

These are the standard comparison operators for arguments of type
int, Word.word, Word8.word, real, char or string.

[makestring v] returns a representation of value v as a string, for
v of type int, Word.word, Word8.word, real, char or string.

[~ x] is the numeric negation of x (which can be real or int).  May
raise Overflow.

[abs x] is the absolute value of x (which can be real or int).  May
raise Overflow.

[+ (e1, e2)]
[- (e1, e2)]
[* (e1, e2)]

These are the standard arithmetic operations for arguments of type
int, Word.word, Word8.word, and real.  They are unsigned in the
case of Word.word and Word8.word.  May raise Overflow.

[/ (e1, e2)] is the floating-point result of dividing e1 by e2.
May raise Div and Overflow.

[div(e1, e2)] is the integral quotient of dividing e1 by e2 for
arguments of type int, Word.word, and Word8.word.  See Int.div and
Word.div for more details.  May raise Div, Overflow.

[mod(e1, e2)] is the remainder when dividing e1 by e2, for
arguments of type int, Word.word, and Word8.word.  See Int.mod and
Word.mod for more details.  May raise Div.
```

Module Help

Help -- on-line help functions

```
val help           : string -> unit

val displayLines   : int ref
val helpdirs       : string list ref
val indexfiles     : string list ref
val specialfiles   : {term : string, file : string, title : string} list ref
val welcome        : string vector ref
val browser        : (string -> unit) ref
val defaultBrowser : string -> unit
```

```
   [help s] provides on-line help on the topic indicated by string s.

      help "lib";   gives an overview of the Moscow ML library.
      help "id";    provides help on identifier id (case-insensitive).

   If exactly one identifier in the library matches id (case-insensitive),
   then the browser opens the signature defining that identifier,
   positioning the first occurrence of id at the center of the screen.

   If more than one identifier matches id (case-insensitive), then a
   small menu lists the signatures containing the identifier.  To
   invoke the browser, just type in the number of the desired
   signature.

   The browser accepts the following commands, which must be followed
   by a newline:

      d      move down by half a screen
      u      move up by half a screen
      t      move to top of file
      b      move to bottom of file
      /str   cyclically search for string str in help file (case-insensitive)
      n      search for next occurrence of str
      q      quit the browser

   A newline by itself moves down one screen (24 lines).

   [helpdirs] is a reference to a list of additional directories to be
   searched for help files.  The directories are searched in order,
   after the -stdlib directory.

   [indexfiles] is a reference to a list of full paths of help term
   index files.  Setting ‘indexfiles’ affects subsequent invocations
   of ‘help’.  (Every invocation of ‘help’ reads the index files anew).

   [specialfiles] is a reference to a list of {term, file, title}
   records, each of which maps a search term to the specified file
   with the specified title (in the browser).  The string in the
   ‘term’ field should be all lowercase, since the argument passed to
   ‘help’ will be converted to lowercase.

   [welcome] is a reference to the text shown in response to the query
   help "".  This is a vector of lines of text.

   [browser] is a reference to the function that gets invoked on the
   text of the help file.  Initially set to defaultBrowser.

   [defaultBrowser] is the default (built-in) help browser.

   [displayLines] is a reference to the size of the display (window)
   assumed by the defaultBrowser; initially 24 lines.  Set it to the
   actual size of your window for best results.
```

Module Int

Int -- SML Basis Library

```
type int = int

val precision : int option
val minInt    : int option
val maxInt    : int option

val ~         : int -> int              Overflow
val *         : int * int -> int        Overflow
val div       : int * int -> int        Div, Overflow
val mod       : int * int -> int        Div
val quot      : int * int -> int        Div, Overflow
val rem       : int * int -> int        Div
val +         : int * int -> int        Overflow
val -         : int * int -> int        Overflow
val >         : int * int -> bool
val >=        : int * int -> bool
val <         : int * int -> bool
val <=        : int * int -> bool
val abs       : int -> int              Overflow
val min       : int * int -> int
val max       : int * int -> int

val sign      : int -> int
val sameSign  : int * int -> bool
val compare   : int * int -> order

val toInt     : int -> int
val fromInt   : int -> int
val toLarge   : int -> int
val fromLarge : int -> int

val scan      : StringCvt.radix
                -> (char, 'a) StringCvt.reader -> (int, 'a) StringCvt.reader
val fmt       : StringCvt.radix -> int -> string

val toString  : int -> string
val fromString : string -> int option   Overflow
```

```
[precision] is SOME n, where n is the number of significant bits in an
integer.  In Moscow ML n is 31 in 32-bit architectures and 63 in 64-bit
architectures.

[minInt] is SOME n, where n is the most negative integer.

[maxInt] is SOME n, where n is the most positive integer.

[~]
[*]
[+]
[-] are the usual operations on integers.  They raise Overflow if
the result is not representable as an integer.

[abs] returns the absolute value of its argument.  Raises Overflow
if applied to the most negative integer.

[div] is integer division, rounding towards minus infinity.
Evaluating i div 0 raises Div.  Evaluating i div ~1 raises
Overflow if i is the most negative integer.

[mod] is the remainder for div.  If q = i div d and r = i mod d then
it holds that qd + r = i, where either 0 <= r < d or d < r <= 0.
Evaluating i mod 0 raises Div, whereas i mod ~1 = 0, for all i.

[quot] is integer division, rounding towards zero.  Evaluating
quot(i, 0) raises Div.  Evaluating quot(i, ~1) raises Overflow if i
is the most negative integer.
```

```
[rem(i, d)] is the remainder for quot.  That is, if q = quot(i, d)
and r = rem(i, d) then d * q + r = i, where r is zero or has the
same sign as i.  If made infix, the recommended fixity for quot and
rem is
     infix 7 quot rem

[min(x, y)] is the smaller of x and y.

[max(x, y)] is the larger of x and y.

[sign x] is ~1, 0, or 1, according as x is negative, zero, or positive.

[<]
[<=]
[>]
[>=] are the usual comparisons on integers.

[compare(x, y)] returns LESS, EQUAL, or GREATER, according
as x is less than, equal to, or greater than y.

[sameSign(x, y)] is true iff sign x = sign y.

[toInt x] is x (because this is the default int type in Moscow ML).

[fromInt x] is x (because this is the default int type in Moscow ML).

[toLarge x] is x (because this is the largest int type in Moscow ML).

[fromLarge x] is x (because this is the largest int type in Moscow ML).

[fmt radix i] returns a string representing i, in the radix (base)
specified by radix.

  radix    description                    output format
  ------------------------------------------------------
   BIN     signed binary      (base  2)    ~?[01]+
   OCT     signed octal       (base  8)    ~?[0-7]+
   DEC     signed decimal     (base 10)    ~?[0-9]+
   HEX     signed hexadecimal (base 16)    ~?[0-9A-F]+

[toString i] returns a string representing i in signed decimal format.
Equivalent to (fmt DEC i).

[fromString s] returns SOME(i) if a decimal integer numeral can be
scanned from a prefix of string s, ignoring any initial whitespace;
returns NONE otherwise.  A decimal integer numeral must have form,
after possible initial whitespace:
     [+~-]?[0-9]+

[scan radix getc charsrc] attempts to scan an integer numeral
from the character source charsrc, using the accessor getc, and
ignoring any initial whitespace.  The radix argument specifies the base
of the numeral (BIN, OCT, DEC, HEX).  If successful, it returns
SOME(i, rest) where i is the value of the number scanned, and rest
is the unused part of the character source.  A numeral must have
form, after possible initial whitespace:

  radix    input format
  ---------------------------
   BIN     [+~-]?[0-1]+
   OCT     [+~-]?[0-7]+
   DEC     [+~-]?[0-9]+
   HEX     [+~-]?[0-9a-fA-F]+
```

Module Intmap

Intmap -- Applicative maps with integer keys
From SML/NJ lib 0.2, copyright 1993 by AT&T Bell Laboratories
Original implementation due to Stephen Adams, Southampton, UK

```
type 'a intmap

exception NotFound

val empty     : unit -> 'a intmap
val insert    : 'a intmap * int * 'a -> 'a intmap
val retrieve  : 'a intmap * int -> 'a
val peek      : 'a intmap * int -> 'a option
val remove    : 'a intmap * int -> 'a intmap * 'a
val numItems  : 'a intmap ->  int
val listItems : 'a intmap -> (int * 'a) list
val app       : (int * 'a -> unit) -> 'a intmap -> unit
val revapp    : (int * 'a -> unit) -> 'a intmap -> unit
val foldr     : (int * 'a * 'b -> 'b) -> 'b -> 'a intmap -> 'b
val foldl     : (int * 'a * 'b -> 'b) -> 'b -> 'a intmap -> 'b
val map       : (int * 'a -> 'b) -> 'a intmap -> 'b intmap
val transform : ('a -> 'b) -> 'a intmap -> 'b intmap
```

['a intmap] is the type of applicative maps from int to 'a.

[empty] creates a new empty map.

[insert(m, i, v)] extends (or modifies) map m to map i to v.

[retrieve(m, i)] returns v if m maps i to v; otherwise raises NotFound.

[peek(m, i)] returns SOME v if m maps i to v; otherwise NONE.

[remove(m, i)] removes i from the domain of m and returns the modified map and the element v corresponding to i. Raises NotFound if i is not in the domain of m.

[numItems m] returns the number of entries in m (that is, the size of the domain of m).

[listItems m] returns a list of the entries (i, v) of integers i and the corresponding values v in m, in increasing order of i.

[app f m] applies function f to the entries (i, v) in m, in increasing order of i.

[revapp f m] applies function f to the entries (i, v) in m, in decreasing order of i.

[foldl f e m] applies the folding function f to the entries (i, v) in m, in increasing order of i.

[foldr f e m] applies the folding function f to the entries (i, v) in m, in decreasing order of i.

[map f m] returns a new map whose entries have form (i, f(i,v)), where (i, v) is an entry in m.

[transform f m] returns a new map whose entries have form (i, f(i,v)), where (i, v) is an entry in m.

Module Intset

Intset -- applicative sets of integers
From SML/NJ lib 0.2, copyright 1993 by AT&T Bell Laboratories
Original implementation due to Stephen Adams, Southampton, UK

```
type intset

exception NotFound

val empty        : intset
val singleton    : int -> intset
val add          : intset * int -> intset
val addList      : intset * int list -> intset
val isEmpty      : intset -> bool
val equal        : intset * intset -> bool
val isSubset     : intset * intset -> bool
val member       : intset * int -> bool
val delete       : intset * int -> intset
val numItems     : intset ->  int
val union        : intset * intset -> intset
val intersection : intset * intset -> intset
val difference   : intset * intset -> intset
val listItems    : intset -> int list
val app          : (int -> unit) -> intset -> unit
val revapp       : (int -> unit) -> intset -> unit
val foldr        : (int * 'b -> 'b) -> 'b -> intset -> 'b
val foldl        : (int * 'b -> 'b) -> 'b -> intset -> 'b
val find         : (int -> bool) -> intset -> int option
```

[intset] is the type of sets of integers.

[empty] is the empty set of integers.

[singleton i] is the singleton set containing i.

[add(s, i)] adds item i to set s.

[addList(s, xs)] adds all items from the list xs to the set s.

[isEmpty s] returns true if and only if the set is empty.

[equal(s1, s2)] returns true if and only if the two sets have the same elements.

[isSubset(s1, s2)] returns true if and only if s1 is a subset of s2.

[member(s, i)] returns true if and only if i is in s.

[delete(s, i)] removes item i from s. Raises NotFound if i is not in s.

[numItems s] returns the number of items in set s.

[union(s1, s2)] returns the union of s1 and s2.

[intersection(s1, s2)] returns the intersectionof s1 and s2.

[difference(s1, s2)] returns the difference between s1 and s2 (that is, the set of elements in s1 but not in s2).

[listItems s] returns a list of the items in set s, in increasing order.

[app f s] applies function f to the elements of s, in increasing order.

[revapp f s] applies function f to the elements of s, in decreasing order.

[foldl f e s] applies the folding function f to the entries of the set in increasing order.

```
[foldr f e s] applies the folding function f to the entries of the
set in decreasing order.

[find p s] returns SOME i, where i is an item in s which satisfies
p, if one exists; otherwise returns NONE.
```

Module Lexing

Lexing -- run-time library for lexers generated by mosmllex
Closely based on the library for camllex. Copyright 1993 INRIA, France

```
local open Obj in

type lexbuf

val createLexerString : string -> lexbuf
val createLexer       : (CharArray.array -> int -> int) -> lexbuf

val getLexeme         : lexbuf -> string
val getLexemeChar     : lexbuf -> int -> char
val getLexemeStart    : lexbuf -> int
val getLexemeEnd      : lexbuf -> int
```

For internal use in generated lexers:

```
val dummyAction       : lexbuf -> obj
val backtrack         : lexbuf -> 'a
prim_val getNextChar  : lexbuf -> char = 1 "get_next_char"

prim_val getLexBuffer     : lexbuf -> string          = 1 "field1"
prim_val getLexAbsPos     : lexbuf -> int             = 1 "field2"
prim_val getLexStartPos   : lexbuf -> int             = 1 "field3"
prim_val getLexCurrPos    : lexbuf -> int             = 1 "field4"
prim_val getLexLastPos    : lexbuf -> int             = 1 "field5"
prim_val getLexLastAction : lexbuf -> (lexbuf -> obj)  = 1 "field6"

prim_val setLexAbsPos     : lexbuf -> int -> unit            = 2 "setfield2"
prim_val setLexStartPos   : lexbuf -> int -> unit            = 2 "setfield3"
prim_val setLexCurrPos    : lexbuf -> int -> unit            = 2 "setfield4"
prim_val setLexLastPos    : lexbuf -> int -> unit            = 2 "setfield5"
prim_val setLexLastAction : lexbuf -> (lexbuf -> obj) -> unit = 2 "setfield6"
end
```

```
These functions are for use in mosmllex-generated lexers.  For
further information, see the Moscow ML Owner's Manual.  For
examples, see mosml/examples/lexyacc and mosml/examples/calc.

[lexbuf] is the type of lexer buffers.  A lexer buffer is the
argument passed to the scanning functions defined by the
mosmllex-generated scanners.  The lexer buffer holds the current
state of the scanner, plus a function to refill the buffer from the
input.

[createLexerString s] returns a lexer buffer which reads from the
given string s. Reading starts from the first character in the
string.  An end-of-input condition is generated when the end of the
string is reached.

[createLexer f] returns a lexer buffer that will use the given
function f for reading additional input.  When the lexer needs more
characters, it will call the given function as (f carr n), where
carr is a character array, and n is an integer.  The function
should put at most characters or in carr, starting at character
number 0, and return the number of characters actually stored.  A
return value of 0 means end of input.

A lexer definition (input to mosmllex) consists of fragments of
this form

   parse
       lhs1   { rhs1 }
     | lhs2   { rhs2 }
     | lhs3   { rhs3 }
     | ...

where the lhs are regular expressions matching some string of
```

characters, and the rhs are corresponding semantic actions, written in ML. The following functions can be used in the semantic actions:

[getLexeme lexbuf] returns the string matched by the left-hand side regular expression.

[getLexemeChar lexbuf i] returns character number i in the matched string.

[getLexemeStart lexbuf] returns the start position of the matched string (in the input stream). The first character in the stream has position 0.

[getLexemeEnd lexbuf] returns the end position, plus one, of the matched string (in the input stream). The first character in the stream has position 0.

Module List

List -- SML Basis Library

```
datatype list = datatype list

exception Empty

val null      : 'a list -> bool
val hd        : 'a list -> 'a                         Empty
val tl        : 'a list -> 'a list                    Empty
val last      : 'a list -> 'a                         Empty

val nth       : 'a list * int -> 'a                   Subscript
val take      : 'a list * int -> 'a list              Subscript
val drop      : 'a list * int -> 'a list              Subscript

val length    : 'a list -> int

val rev       : 'a list -> 'a list

val @         : 'a list * 'a list -> 'a list
val concat    : 'a list list -> 'a list
val revAppend : 'a list * 'a list -> 'a list

val app       : ('a -> unit) -> 'a list -> unit
val map       : ('a -> 'b) -> 'a list -> 'b list
val mapPartial : ('a -> 'b option) -> 'a list -> 'b list

val find      : ('a -> bool) -> 'a list -> 'a option
val filter    : ('a -> bool) -> 'a list -> 'a list
val partition : ('a -> bool ) -> 'a list -> ('a list * 'a list)

val foldr     : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
val foldl     : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b

val exists    : ('a -> bool) -> 'a list -> bool
val all       : ('a -> bool) -> 'a list -> bool

val tabulate  : int * (int -> 'a) -> 'a list           Size

val getItem   : 'a list -> ('a * 'a list) option
```

['a list] is the type of lists of elements of type 'a.

[null xs] is true iff xs is nil.

[hd xs] returns the first element of xs. Raises Empty if xs is nil.

[tl xs] returns all but the first element of xs.
Raises Empty if xs is nil.

[last xs] returns the last element of xs. Raises Empty if xs is nil.

[nth(xs, i)] returns the i'th element of xs, counting from 0.
Raises Subscript if i<0 or i>=length xs.

[take(xs, i)] returns the first i elements of xs. Raises Subscript if i<0 or i>length xs.

[drop(xs, i)] returns what is left after dropping the first i elements of xs. Raises Subscript if i<0 or i>length xs.
It holds that take(xs, i) @ drop(xs, i) = xs when 0 <= i <= length xs.

[length xs] returns the number of elements in xs.

[rev xs] returns the list of xs's elements, reversed.

[xs @ ys] returns the list which is the concatenation of xs and ys.

[concat xss] returns the list which is the concatenation of all the

```
lists in xss.

[revAppend(xs, ys)] is equivalent to rev xs @ ys, but more efficient.

[app f xs] applies f to the elements of xs, from left to right.

[map f xs] applies f to each element x of xs, from left to
right, and returns the list of f's results.

[mapPartial f xs] applies f to each element x of xs, from left
to right, and returns the list of those y's for which f(x)
evaluated to SOME y.

[find p xs] applies f to each element x of xs, from left to
right until p(x) evaluates to true; returns SOME x if such an x
exists otherwise NONE.

[filter p xs] applies p to each element x of xs, from left to
right, and returns the sublist of those x for which p(x) evaluated
to true.

[partition p xs] applies p to each element x of xs, from left
to right, and returns a pair (pos, neg) where pos is the sublist
of those x for which p(x) evaluated to true, and neg is the sublist of
those for which p(x) evaluated to false.

[foldr op% e xs] evaluates x1 % (x2 % ( ... % (x(n-1) % (xn % e)) ... ))
where xs = [x1, x2, ..., x(n-1), xn], and % is taken to be infixed.

[foldl op% e xs] evaluates xn % (x(n-1) % ( ... % (x2 % (x1 % e))))
where xs = [x1, x2, ..., x(n-1), xn], and % is taken to be infixed.

[exists p xs] applies p to each element x of xs, from left to
right until p(x) evaluates to true; returns true if such an x
exists, otherwise false.

[all p xs] applies p to each element x of xs, from left to
right until p(x) evaluates to false; returns false if such an x
exists, otherwise true.

[tabulate(n, f)] returns a list of length n whose elements are
f(0), f(1), ..., f(n-1), created from left to right.  Raises Size
if n<0.

[getItem xs] attempts to extract an element from the list xs.  It
returns NONE if xs is empty, and returns SOME (x, xr) if xs=x::xr.
This can be used for scanning booleans, integers, reals, and so on
from a list of characters.  For instance, to scan a decimal integer
from a list cs of characters, compute
     Int.scan StringCvt.DEC List.getItem cs
```

Module ListPair

ListPair -- SML Basis Library

```
val zip    : 'a list * 'b list -> ('a * 'b) list
val unzip  : ('a * 'b) list -> 'a list * 'b list
val map    : ('a * 'b -> 'c)   -> 'a list * 'b list -> 'c list
val app    : ('a * 'b -> unit) -> 'a list * 'b list -> unit
val all    : ('a * 'b -> bool) -> 'a list * 'b list -> bool
val exists : ('a * 'b -> bool) -> 'a list * 'b list -> bool
val foldr  : ('a * 'b * 'c -> 'c) -> 'c -> 'a list * 'b list -> 'c
val foldl  : ('a * 'b * 'c -> 'c) -> 'c -> 'a list * 'b list -> 'c
```

```
   These functions process pairs of lists.  No exception is raised
   when the lists are found to be of unequal length.  Instead the
   excess elements from the longer list are disregarded.

   [zip (xs, ys)] returns the list of pairs of corresponding elements
   from xs and ys.

   [unzip xys] returns a pair (xs, ys), where xs is the list of first
   components of xys, and ys is the list of second components from
   xys.  Hence zip (unzip xys) has the same result and effect as xys.

   [map f (xs, ys)] applies function f to the pairs of corresponding
   elements of xs and ys and returns the list of results.  Hence
   map f (xs, ys) has the same result and effect as List.map f (zip (xs, ys)).

   [app f (xs, ys)] applies function f to the pairs of corresponding
   elements of xs and ys and returns ().  Hence app f (xs, ys) has the
   same result and effect as List.app f (zip (xs, ys)).

   [all p (xs, ys)] applies predicate p to the pairs of corresponding
   elements of xs and ys until p evaluates to false or one or both
   lists is exhausted; returns true if p is true of all such pairs;
   otherwise false.  Hence all p (xs, ys) has the same result and
   effect as Lisp.all p (zip (xs, ys)).

   [exists p (xs, ys)] applies predicate p to the pairs of corresponding
   elements of xs and ys until p evaluates to true or one or both
   lists is exhausted; returns true if p is true of any such pair;
   otherwise false.  Hence exists p (xs, ys) has the same result and
   effect as Lisp.exists p (zip (xs, ys)).

   [foldr f e (xs, ys)] evaluates f(x1, y1, f(x2, y2, f(..., f(xn, yn, e))))
   where xs = [x1, x2, ..., x(n-1), xn, ...],
         ys = [y1, y2, ..., y(n-1), yn, ...],
   and    n = min(length xs, length ys).
   Equivalent to List.foldr (fn ((x, y), r) => f(x, y, r)) e (zip(xs, ys)).

   [foldl f e (xs, ys)] evaluates f(xn, yn, f( ..., f(x2, y2, f(x1, y1, e))))
   where xs = [x1, x2, ..., x(n-1), xn, ...],
         ys = [y1, y2, ..., y(n-1), yn, ...],
   and    n = min(length xs, length ys).
   Equivalent to List.foldl (fn ((x, y), r) => f(x, y, r)) e (zip(xs, ys)).
```

Module Listsort

Listsort

```
val sort   : ('a * 'a -> order) -> 'a list -> 'a list
val sorted : ('a * 'a -> order) -> 'a list -> bool
```

[sort ordr xs] sorts the list xs in nondecreasing order, using the given ordering. Uses Richard O'Keefe's smooth applicative merge sort.

[sorted ordr xs] checks that the list xs is sorted in nondecreasing order, in the given ordering.

Module Location

Location -- error reporting for mosmllex and mosmlyac
Based on src/compiler/location from the Caml Light 0.6 distribution

```
datatype Location =  Source file positions
    Loc of int       Position of the first character
         * int       Position of the character following the last one

val errLocation : string * BasicIO.instream * Lexing.lexbuf -> Location
                  -> unit
val errMsg      : string * BasicIO.instream * Lexing.lexbuf -> Location
                  -> string -> 'a
val errPrompt   : string -> unit;
val nilLocation : Location
val getCurrentLocation : unit -> Location
val mkLoc : 'a -> Location * 'a
val xLR   : Location * 'a -> Location
val xL    : Location * 'a -> int
val xR    : Location * 'a -> int
val xxLR  : Location * 'a -> Location * 'b -> Location
val xxRL  : Location * 'a -> Location * 'b -> Location
```

These functions support error reporting in lexers and parsers generated with mosmllex and mosmlyac. The directory mosml/examples/lexyacc/ contains an example of their use.

[errLocation (file, stream, lexbuf) loc] prints the part of the lexer input which is indicated by location loc.

If file <> "" then it is assumed to be the name of the file from which the lexer reads, the stream is assumed to be an open input stream associated with this file, and lexbuf is the lexer buffer used to read from the stream. Under MS DOS (and presumably Windows, OS/2, and MacOS), the stream must have been opened in binary mode (with Nonstdio.open_in_bin), or else the positioning in the file will be wrong (due to the translation of CRLF into newline in text files).

If file = "" then the lexer is assumed to read from some source other than a stream, and the lexbuf (rather than the instream) is used to obtain the location indicated, if possible. In this case the stream is immaterial; it will not be used.

[errMsg (file, stream, lexbuf) loc msg] calls errLocation to print the indicated part of the lexer input, then prints the error message msg and raises exception Fail.

[errPrompt msg] prints "! ", the string msg, and a newline on standard output.

[nilLocation] is the undefined location.

[getCurrentLocation ()] can be called within the semantic action part of a grammar rule (only) and returns the location of the string matching the left-hand side of the rule.

[mkLoc a] can be called within the semantic action part of a grammar rule (only), and returns a pair (loc, a) of the current location and the value a. This is typically used to decorate abstract syntax tree nodes with location information, for use in subsequent error reports.

[xLR loc_a] returns the location of the decorated value loc_a.

[xL loc_a] returns the left end position of loc_a.

[xR loc_a] returns the right end position of loc_a.

[xxLR loc_a loc_b] returns the location extending from the left end of loc_a to the right end of loc_b.

```
[xxRL loc_a loc_b] returns the location extending from the right end
of loc_a to the left end of loc_b.
```

Module Math

Math -- SML Basis Library

```
type real = real

val pi    : real
val e     : real

val sqrt  : real -> real
val sin   : real -> real
val cos   : real -> real
val tan   : real -> real
val atan  : real -> real
val asin  : real -> real
val acos  : real -> real
val atan2 : real * real -> real
val exp   : real -> real
val pow   : real * real -> real
val ln    : real -> real
val log10 : real -> real
val sinh  : real -> real
val cosh  : real -> real
val tanh  : real -> real
```

```
[pi] is the circumference of the circle with diameter 1, that is,
3.14159265358979323846.

[e] is the base of the natural logarithm: 2.7182818284590452354.

[sqrt x] is the square root of x.  Raises Domain if x < 0.0.

[sin r] is the sine of r, where r is in radians.

[cos r] is the cosine of r, where r is in radians.

[tan r] is the tangent of r, where r is in radians.  Raises Domain if
r is a multiple of pi/2.

[atan t] is the arc tangent of t, in the open interval ] ~pi/2, pi/2 [.

[asin t] is the arc sine of t, in the closed interval [ ~pi/2, pi/2 ].
Raises Domain if abs x > 1.

[acos t] is the arc cosine of t, in the closed interval [ 0, pi ].
Raises Domain if abs x > 1.

[atan2(y, x)] is the arc tangent of y/x, in the interval ] ~pi, pi ],
except that atan2(y, 0) = sign y * pi/2.  The quadrant of the result
is the same as the quadrant of the point (x, y).
Hence sign(cos(atan2(y, x))) = sign x
and   sign(sin(atan2(y, x))) = sign y.

[exp x] is e to the x’th power.

[pow (x, y)] is x it the y’th power, defined when
   y >= 0 and (y integral or x >= 0)
or y < 0 and ((y integral and x <> 0.0) or x > 0).

We define pow(0, 0) = 1.

[ln x] is the natural logarithm of x (that is, with base e).
Raises Domain if x <= 0.0.

[log10 x] is the base-10 logarithm of x.  Raises Domain if x <= 0.0.

[sinh x] returns the hyperbolic sine of x, mathematically defined as
(exp x - exp (~x)) / 2.  Raises Overflow if x is too large.

[cosh x] returns the hyperbolic cosine of x, mathematically defined as
(exp x + exp (~x)) / 2.  Raises Overflow if x is too large.
```

```
[tanh x] returns the hyperbolic tangent of x, mathematically defined
as (sinh x) / (cosh x).  Raises Domain if x is too large.
```

Module Meta

Meta -- functions available only in interactive Moscow ML sessions

```
val printVal         : 'a -> 'a
val printDepth       : int ref
val printLength      : int ref
val installPP        : (ppstream -> 'a -> unit) -> unit

val liberal          : unit -> unit
val conservative     : unit -> unit
val orthodox         : unit -> unit

val use              : string -> unit
val compile          : string -> unit
val compileToplevel  : string list -> string -> unit
val compileStructure : string list -> string -> unit

val load             : string -> unit
val loadOne          : string -> unit
val loaded           : unit -> string list
val loadPath         : string list ref

val quietdec         : bool ref
val verbose          : bool ref

val quotation        : bool ref
val valuepoly        : bool ref

val quit             : unit -> 'a
```

```
These values and functions are available in the Moscow ML
interactive system only.

[printVal e] prints the value of expression e to standard output
exactly as it would be printed at top-level, and returns the value
of e.  Output is flushed immediately.  This function is provided as
a simple debugging aid.  The effect of printVal is similar to that
of 'print' in Edinburgh ML or Umeaa ML.  For string arguments, the
effect of SML/NJ print can be achieved by the function
TextIO.print : string -> unit.

[printDepth] determines the depth (in terms of nested constructors,
records, tuples, lists, and vectors) to which values are printed by
the top-level value printer and the function printVal.  The components
of the value whose depth is greater than printDepth are printed as
'#'.  The initial value of printDepth is 20. This value can be
changed at any moment, by evaluating, for example,
     printDepth := 17;

[printLength] determines the way in which list values are printed
by the top-level value printer and the function printVal.  If the
length of a list is greater than printLength, then only the first
printLength elements are printed, and the remaining elements are
printed as '...'.  The initial value of printLength is 200.  This
value can be changed at any moment, by evaluating, for example,
     printLength := 500;

[quit ()] quits Moscow ML immediately.

[installPP pp] installs the prettyprinter pp : ppstream -> ty -> unit
at type ty.  The type ty must be a nullary (parameter-less) type
constructor representing a datatype, either built-in (such as bool)
or user-defined.  Whenever a value of type ty is about to be
printed by the interactive system, or function printVal is invoked
on an argument of type ty, the pretty-printer pp will be invoked to
print it.  See library unit PP for more information.

[use "f"] causes ML declarations to be read from file f as if they
were entered from the console.  A file loaded by use may, in turn,
evaluate calls to use.  For best results, use 'use' only at top
```

level, or at top level within a use'd file.

[liberal ()] sets liberal mode for the compilation functions: accept (without warnings) all extensions to the SML Modules language. The extensions are: higher-order modules (functors defined within structures and functors); first-order modules (structures can be packed as values, and values can be unpacked as structures); and recursively defined modules (signatures and structures). The liberal, conservative, and orthodox modes affect the functions compile, compileStructure, and compileToplevel. The liberal mode may be set also by the mosml option -liberal.

[conservative ()] sets conservative mode for the compilation functions: accept all extensions to the SML Modules language, but issue a warning for each use. The conservative mode may be set also by the mosml option -conservative. This is the default.

[orthodox ()] sets orthodox mode for the compilation functions: reject all uses of the extensions to the SML Modules language. That is, accept only SML Modules syntax. The orthodox mode may be set also by the mosml option -orthodox.

[compile "U.sig"] will compile and elaborate the specifications in file U.sig in structure mode, producing a compiled signature U in file U.ui. This function is backwards compatible with Moscow ML 1.44 and earlier. Equivalent to compileStructure [] "U.sig".

[compile "U.sml"] will elaborate and compile the declarations in file U.sml in structure mode, producing a compiled structure U in bytecode file U.uo. If there is an explicit signature file U.sig, then file U.ui must exist, and the unit body must match the signature. If there is no U.sig, then an inferred signature file U.ui will be produced also. No evaluation takes place. This function is backwards compatible with Moscow ML 1.44 and earlier. Equivalent to compileStructure [] "U.sml".

The declared identifiers will be reported if verbose is true (see below); otherwise compilation will be silent. In any case, compilation warnings are reported, and compilation errors abort the compilation and raise the exception Fail with a string argument.

[compileStructure opnunits "U.sig"] compiles the specifications in file U.sig as if they form a signature declaration
```
     signature U = sig ... contents of U.sig ... end
```
The contents of opnunits is added to the compilation context in which the specifications in U.sig are compiled. The result is a compiled signature file U.ui. This
corresponds to invoking the batch compiler as follows:
```
     mosmlc -c U1.ui ... Un.ui -structure U.sig
```
where opnunits equals ["U1", ..., "Un"].

[compileStructure opnunits "U.sml"] compiles the declarations in file U.sml as if they formed a structure declaration
```
     structure U = struct ... contents of U.sml ... end
```
The contents of opnunits is added to the compilation context in which the declarations in U.sml are compiled. If U.ui exists already and represents a signature called U, then the compiled declarations are matched against it. The result is a bytecode file U.uo. If no file U.ui existed, then also a file U.ui is created, containing the inferred signature of structure U. This corresponds to invoking the batch compiler as follows:
```
     mosmlc -c U1.ui ... Un.ui -structure U.sml
```
where opnunits equals ["U1", ..., "Un"].

[compileToplevel opnunits "U.sig"] compiles the specifications in file U.sig, in a context in which all declarations from opnunits are visible, creating a compiled signature file U.ui. This corresponds to invoking the batch compiler as follows:
```
     mosmlc -c U1.ui ... Un.ui -toplevel U.sig
```
where opnunits equals ["U1", ..., "Un"].

[compileToplevel opnunits "U.sml"] compiles the declarations in

file U.sml, in a context in which all declarations from opnunits are visible, creating a bytecode file U.uo. If U.ui exists already, then the compiled declarations are matched against it; otherwise the file U.ui is created. This corresponds to invoking the batch compiler as follows
```
     mosmlc -c U1.ui ... Un.ui -toplevel U.sml
```
where opnunits equals ["U1", ..., "Un"].

[load "U"] will load and evaluate the compiled unit body from file U.uo. The resulting values are not reported, but exceptions are reported, and cause evaluation and loading to stop. If U is already loaded, then load "U" has no effect. If any other unit is mentioned by U but not yet loaded, then it will be loaded automatically before U.

After loading a unit, it can be opened with 'open U'. Opening it at top-level will list the identifiers declared in the unit.

When loading U, it is checked that the signatures of units mentioned by U agree with the signatures used when compiling U, and it is checked that the signature of U has not been modified since U was compiled; these checks are necessary for type safety. The exception Fail is raised if these signature checks fail, or if the file containing U or a unit mentioned by U does not exist.

[loadOne "U"] is similar to 'load "U"', but raises exception Fail if U is already loaded or if some unit mentioned by U is not yet loaded. That is, it does not automatically load any units mentioned by U. It performs the same signature checks as 'load'.

[loaded ()] returns a list of the names of all compiled units that have been loaded so far. The names appear in some random order.

[loadPath] determines the load path: which directories will be searched for interface files (.ui files), bytecode files (.uo files), and source files (.sml files). This variable affects the load, loadOne, and use functions. The current directory is always searched first, followed by the directories in loadPath, in order. By default, only the standard library directory is in the list, but if additional directories are specified using option -I, then these directories are prepended to loadPath.

[quietdec] when true, turns off the interactive system's prompt and responses, except warnings and error messages. Useful for writing scripts in SML. The default value is false; can be set to true with the -quietdec command line option.

[verbose] determines whether the signature inferred by a call to compile will be printed. The printed signature follows the syntax of Moscow ML signatures, so the output of compile "U.sml" can be edited to subsequently create file U.sig. The default value is ref false.

[quotation] determines whether quotations and antiquotations are permitted in declarations entered at top-level and in files compiled with compile. A quotation is a piece of text surrounded by backquote characters 'a b c' and is used to embed object language phrases in ML programs; see the Moscow ML Owner's Manual for a brief explanation of quotations. When quotation is false, the backquote character is an ordinary symbol which can be used in ML symbolic identifiers. When quotation is true, the backquote character is illegal in symbolic identifiers, and a quotation 'a b c' will be recognized by the parser and evaluated to an object of type 'a General.frag list. False by default.

[valuepoly] determines whether value polymorphism is used or not in the type checker. With value polymorphism (the default), there is no distinction between imperative ('_a) and applicative ('a) type variables, and type variables are generalized only in bindings to non-expansive expressions. Non-generalized type variables are left free, to be instantiated when the bound identifier is used. An expression is non-expansive if it is a variable, a special

```
constant, a function, a tuple or record of non-expansive
expressions, a parenthesized or typed non-expansive expression, or
the application of an exception or value constructor (other than
ref) to a non-expansive expression.  If valuepoly is false, then
the type checker will distinguish imperative and applicative type
variables, generalize all applicative type variables, and
generalize imperative type variables only in non-expansive
expressions.  True by default.
```

Module Mosml

Mosml -- some Moscow ML specific functions

```
val argv      : unit -> string list
val time      : ('a -> 'b) -> ('a -> 'b)
val listDir   : string -> string list
val doubleVec : real -> Word8Vector.vector
val vecDouble : Word8Vector.vector -> real
val floatVec  : real -> Word8Vector.vector
val vecFloat  : Word8Vector.vector -> real
val md5sum    : string -> string

datatype runresult =
    Success of string
  | Failure of string

val run : string -> string list -> string -> runresult
```

```
   [argv ()] returns the command line strings of the current process.
   Hence List.nth(argv (), 0) is the command used to invoke the SML
   process, List.nth(argv (), 1) is its first argument, and so on.
   We recommend using the SML Basis Library CommandLine structure instead.

   [time f arg] applies f to arg and returns the result; as a side
   effect, it prints the time (cpu, system, and real time) consumed by
   the evaluation.

   [listDir path] returns the list of all files and subdirectories of
   the directory indicated by path.  Raises OS.SysErr in case of failure.

   [doubleVec r] returns an eight-element vector of Word8.word, which
   contains the real number in the IEEE 754 floating-point 'double
   format' bit layout stored in big-endian (high byte first) order.

   [vecDouble v] accepts an eight-element vector v of Word8.word, and
   returns the real number obtained by taking v to be an IEEE 754
   floating-point 'double format' number stored in big-endian (high
   byte first) order.  Raises Fail if v is not en eight-element
   vector.

   [floatVec r] returns a four-element vector of Word8.word, which
   contains the real number in the IEEE 754 floating-point 'float
   format' bit layout stored in big-endian (high byte first) order.
   Raises Fail if r is not representable as a 32-bit float.

   [vecFloat v] accepts a four-element vector v of Word8.word, and
   returns the real obtained by taking v to be an IEEE 754
   floating-point 'float format' number stored in big-endian (high
   byte first) order.  Raises Fail if v is not a four-element vector.

   [md5sum s] computes the 128-bit MD5 checksum of string s and
   returns it as a 22 character base64 string.

   [run cmd args inp] executes the program cmd with command-line
   arguments args and standard input inp.  Returns Success s where s
   is the program's (standard and error) output as a string, if it
   executed successfully; otherwise returns Failure s where s is its
   (standard and error) output as a string.
```

Module Mosmlcgi

Mosmlcgi -- support for writing CGI scripts in Moscow ML

1. Accessing the fields or parameters of a CGI call

```
val cgi_fieldnames     : string list
val cgi_field_strings  : string -> string list;
val cgi_field_string   : string -> string option;
val cgi_field_integer  : string * int -> int;
```

2. Accessing parts in multipart/form-data; form-based file upload

```
val cgi_partnames      : string list

type part
val cgi_part           : string -> part option
val cgi_parts          : string -> part list

val part_fieldnames    : part -> string list
val part_type          : part -> string option
val part_data          : part -> string
val part_field_strings : part -> string -> string list
val part_field_string  : part -> string -> string option
val part_field_integer : part -> string * int -> int
```

3. Administrative information

```
val cgi_server_software       : string option
val cgi_server_name           : string option
val cgi_gateway_interface     : string option
val cgi_server_protocol       : string option
val cgi_server_port           : string option
val cgi_request_method        : string option
val cgi_http_accept           : string option
val cgi_http_user_agent       : string option
val cgi_http_referer          : string option
val cgi_path_info             : string option
val cgi_path_translated       : string option
val cgi_script_name           : string option
val cgi_query_string          : string option
val cgi_remote_host           : string option
val cgi_remote_addr           : string option
val cgi_remote_user           : string option
val cgi_remote_ident          : string option
val cgi_auth_type             : string option
val cgi_content_type          : string option
val cgi_content_length        : string option
val cgi_annotation_server     : string option

val cgi_http_cookie           : string option
val cgi_http_forwarded        : string option
val cgi_http_host             : string option
val cgi_http_proxy_connection : string option
val cgi_script_filename       : string option
val cgi_document_root         : string option
val cgi_server_admin          : string option
val cgi_api_version           : string option
val cgi_the_request           : string option
val cgi_request_uri           : string option
val cgi_request_filename      : string option
val cgi_is_subreq             : string option
```

```
   The Mosmlcgi library is for writing CGI programs in Moscow ML.  A
   CGI program may be installed on a WWW server and is invoked in
   response to HTTP requests sent to the server from a web browser,
   typically from an HTML FORM element.


   1. Obtaining field values sent from an ordinary HTML form
   ---------------------------------------------------------
```

```
[cgi_fieldnames] is a list of the names of fields present in the
CGI call message.  If field name fnm is in cgi_fieldnames, then
cgi_field_string fnm <> NONE.

[cgi_field_strings fnm] is a (possibly empty) list of the strings
bound to field fnm.

[cgi_field_string fnm] returns SOME(s) where s is a string bound to
field name fnm, if any; otherwise NONE.  Equivalent to
     case cgi_field_strings fnm of
          []     => NONE
        | s :: _ => SOME s

[cgi_field_integer (fnm, deflt)] attempts to parse an integer from
field fnm.  Returns i if cgi_field_string(fnm) = SOME(s) and an
integer i can be parsed from a prefix of s; otherwise returns deflt.


2. Obtaining field values sent with ENCTYPE="multipart/form-data"
-----------------------------------------------------------------

[cgi_partnames] is a list of the names of the parts of the
multipart/form-data message.

The type part is the abstract type of parts of a message.  Each part
may have several fields.  In this implementation, the field of a
part cannot be a another part itself.

[cgi_parts pnm] is a (possibly empty) list of the parts called pnm.

[cgi_part pnm] is SOME(prt) where prt is a part called pnm, if any;
otherwise NONE.  Equivalent to
     case cgi_parts pnm of
          []       => NONE
        | prt :: _ => SOME prt

[part_fieldnames prt] is the list of field names in part pnm.

[part_type prt] is SOME(typ) if the part prt contains a specification
'Context-Type: typ'; otherwise NONE.

[part_data prt] is the data contain in part prt; for instance, the
contents of a file uploaded via form-based file upload.

[part_field_strings prt fnm] is a (possibly empty) list of the
strings bound to field fnm in part prt.

[part_field_string prt fnm] returns SOME(s) where s is a string
bound to field name fnm in part prt, if any; otherwise NONE.
Equivalent to
     case part_field_strings prt fnm of
          []     => NONE
        | s :: _ => SOME s

[part_field_integer prt (fnm, deflt)] attempts to parse an integer
from field fnm of part prt.  Returns i if part_field_string prt fnm
= SOME(s) and an integer i can be parsed from a prefix of s;
otherwise returns deflt.


3. Administrative and server information
----------------------------------------

Each of the following variables has the value SOME(s) if the
corresponding CGI environment variable is bound to string s;
otherwise NONE:

[cgi_server_software] is the value of SERVER_SOFTWARE

[cgi_server_name] is the value of SERVER_NAME
```

```
[cgi_gateway_interface] is the value of GATEWAY_INTERFACE

[cgi_server_protocol] is the value of SERVER_PROTOCOL

[cgi_server_port] is the value of SERVER_PORT

[cgi_request_method] is the value of REQUEST_METHOD

[cgi_http_accept] is the value of HTTP_ACCEPT

[cgi_http_user_agent] is the value of HTTP_USER_AGENT

[cgi_http_referer] is the value of HTTP_REFERER

[cgi_path_info] is the value of PATH_INFO

[cgi_path_translated] is the value of PATH_TRANSLATED

[cgi_script_name] is the value of SCRIPT_NAME

[cgi_query_string] is the value of QUERY_STRING

[cgi_remote_host] is the value of REMOTE_HOST

[cgi_remote_addr] is the value of REMOTE_ADDR

[cgi_remote_user] is the value of REMOTE_USER

[cgi_remote_ident] is the value of REMOTE_IDENT

[cgi_auth_type] is the value of AUTH_TYPE

[cgi_content_type] is the value of CONTENT_TYPE

[cgi_content_length] is the value of CONTENT_LENGTH, that is, the
length of the data transmitted in the CGI call.

[cgi_annotation_server] is the value of ANNOTATION_SERVER

[cgi_http_cookie] is the value of HTTP_COOKIE

[cgi_http_forwarded] is the value of HTTP_FORWARDED

[cgi_http_host] is the value of HTTP_HOST

[cgi_http_proxy_connection] is the value of HTTP_PROXY_CONNECTION

[cgi_script_filename] is the value of SCRIPT_FILENAME

[cgi_document_root] is the value of DOCUMENT_ROOT

[cgi_server_admin] is the value of SERVER_ADMIN

[cgi_api_version] is the value of API_VERSION

[cgi_the_request] is the value of THE_REQUEST

[cgi_request_uri] is the value of REQUEST_URI

[cgi_request_filename] is the value of REQUEST_FILENAME

[cgi_is_subreq] is the value of IS_SUBREQ
```

Module Mosmlcookie

Mosmlcookie -- getting and setting cookies in CGI scripts

```
exception CookieError of string

val allCookies     : string list
val getCookieValue : string -> string option
val getCookie      : string -> string option

type cookiedata =
    { name   : string,
      value  : string,
      expiry : Date.date option,
      domain : string option,
      path   : string option,
      secure : bool }

val setCookie    : cookiedata -> string
val setCookies   : cookiedata list -> string

val deleteCookie : { name : string, path : string option } -> string
```

```
   These functions may be used in CGI scripts to get and set cookies.
   (c) Hans Molin, Computing Science Dept., Uppsala University, 1999.

   [getCookieValue ck] returns SOME(v) where v is the value associated
   with the cookie ck, if any; otherwise returns NONE.

   [getCookie ck] returns SOME(nv) where nv is the ck=value string
   for the cookie ck, if any; otherwise returns NONE.

   [allCookies] is a list [nv1, nv2, ..., nvm] of all the ck=value
   pairs of defined cookies.

   [setCookie { name, value, expiry, domain, path, secure }] returns a
   string which (when transmitted to a browser as part of the HTTP
   response header) sets a cookie with the given name, value, expiry
   date, domain, path, and security.

   [setCookies ckds] returns a string which (when transmitted to a
   browser as part of the HTTP response header) sets the specified cookies.

   [deleteCookie { name, path }] returns a string which (when
   transmitted to a browser as part of the HTTP response header)
   deletes the specified cookie by setting its expiry to some time in
   the past.
```

Module Msp

Msp -- utilities for CGI scripts and ML Server Pages

Efficiently concatenable word sequences

```
datatype wseq =
    Empty                           The empty sequence
  | Nl                              Newline
  | $ of string                     A string
  | $$ of string list               A sequence of strings
  | && of wseq * wseq;              Concatenation of sequences
```

Manipulating wseqs

```
val prmap    : ('a -> wseq) -> 'a list -> wseq
val prsep    : wseq -> ('a -> wseq) -> 'a list -> wseq
val flatten  : wseq -> string
val printseq : wseq -> unit
val vec2list : 'a vector -> 'a list
```

Shorthands for accessing CGI parameters

```
exception ParamMissing of string
exception NotInt of string * string

val %        : string -> string
val %?       : string -> bool
val %#       : string -> int
val %%       : string * string -> string
val %%#      : string * int -> int
```

HTML generic marks

```
val mark0    : string -> wseq
val mark0a   : string -> string -> wseq
val mark1    : string -> wseq -> wseq
val mark1a   : string -> string -> wseq -> wseq
val comment  : wseq -> wseq
```

HTML documents and headers

```
val html     : wseq -> wseq
val head     : wseq -> wseq
val title    : wseq -> wseq
val body     : wseq -> wseq
val bodya    : string -> wseq -> wseq
val htmldoc  : wseq -> wseq -> wseq
```

HTML headings and vertical format

```
val h1       : wseq -> wseq
val h2       : wseq -> wseq
val h3       : wseq -> wseq
val h4       : wseq -> wseq
val h5       : wseq -> wseq
val h6       : wseq -> wseq
val p        : wseq -> wseq
val pa       : string -> wseq -> wseq
val br       : wseq
val bra      : string -> wseq
val hr       : wseq
val hra      : string -> wseq

val divi        : wseq -> wseq
val divia       : string -> wseq -> wseq
val blockquote  : wseq -> wseq
val blockquotea : string -> wseq -> wseq
val center      : wseq -> wseq
```

```
val address     : wseq -> wseq
val pre         : wseq -> wseq
```

HTML anchors and hyperlinks

```
val ahref    : string -> wseq -> wseq
val ahrefa   : string -> string -> wseq -> wseq
val aname    : string -> wseq -> wseq
```

HTML text formats and style

```
val em       : wseq -> wseq
val strong   : wseq -> wseq
val tt       : wseq -> wseq
val sub      : wseq -> wseq
val sup      : wseq -> wseq
val fonta    : string -> wseq -> wseq
```

HTML lists

```
val ul       : wseq -> wseq
val ula      : string -> wseq -> wseq
val ol       : wseq -> wseq
val ola      : string -> wseq -> wseq
val li       : wseq -> wseq
val dl       : wseq -> wseq
val dla      : string -> wseq -> wseq
val dt       : wseq -> wseq
val dd       : wseq -> wseq
```

HTML tables

```
val table    : wseq -> wseq
val tablea   : string -> wseq -> wseq
val tr       : wseq -> wseq
val tra      : string -> wseq -> wseq
val td       : wseq -> wseq
val tda      : string -> wseq -> wseq
val th       : wseq -> wseq
val tha      : string -> wseq -> wseq
val caption  : wseq -> wseq
val captiona : string -> wseq -> wseq
```

HTML images and image maps

```
val img      : string -> wseq
val imga     : string -> string -> wseq
val map      : string -> wseq -> wseq
val mapa     : string -> string -> wseq -> wseq
val area     : { alt : string option, coords : string,
                 href : string option, shape : string} -> wseq
```

HTML forms etc

```
val form       : string -> wseq -> wseq
val forma      : string -> string -> wseq -> wseq
val input      : string -> wseq
val inputa     : string -> string -> wseq
val intext     : string -> string -> wseq
val inpassword : string -> string -> wseq
val incheckbox : {name : string, value : string} -> string -> wseq
val inradio    : {name : string, value : string} -> string -> wseq
val inreset    : string -> string -> wseq
val insubmit   : string -> string -> wseq
val inhidden   : {name : string, value : string} -> wseq
val textarea   : string -> wseq -> wseq
val textareaa  : string -> string -> wseq -> wseq
val select     : string -> string -> wseq -> wseq
val option     : string -> wseq
```

HTML frames and framesets

```
val frameset   : string -> wseq -> wseq
val frame      : { src : string, name : string } -> wseq
val framea     : { src : string, name : string } -> string -> wseq
```

HTML encoding

```
val urlencode  : string -> string
val htmlencode : string -> string
```

```
   This module provides support functions for writing CGI scripts and
   ML Server Page scripts.

   [wseq] is the type of efficiently concatenable word sequences.
   Building an HTML page (functionally) as a wseq is more efficient
   than building it (functionally) as a string, and more convenient
   and modular than building it (imperatively) by calling print.

   [Empty] represents the empty string "".

   [Nl] represents the string "\n" consisting of a single newline character.

   [$ s] represents the string s.

   [$$ ss] represents the string String.concat(ss).

   [&&(ws1, ws2)] represents the concatenation of the strings
   represented by ws1 and ws2.  The function && should be declared
        infix &&

   [prmap f xs] is f x1 && ... && f xn evaluated from left to right,
   when xs is [x1, ..., xn].

   [prsep sep f xs] is f x1 && sep && ... && sep && f xn, evaluated
   from left to right, when xs is [x1, ..., xn].

   [flatten ws] is the string represented by ws.

   [printseq ws] is equivalent to print(flatten ws), but avoids
   building any new strings.

   [vec2list vec] is a list of the elements of vector vec.  Use it to
   convert e.g. the results of a database query into a list, for
   processing with prmap or prsep.


   Shorthands for accessing CGI parameters:

   [%? fnm] returns true if there is a string associated with CGI
   parameter fnm; otherwise returns false.

   [% fnm] returns a string associated with CGI parameter fnm if there
   is any; raises ParamMissing(fnm) if no strings are associated with
   fnm.  Equivalent to
       case Mosmlcgi.cgi_field_string fnm of
           NONE   => raise ParamMissing "fnm"
         | SOME v => v
   In general, multiple strings may be associated with a CGI parameter;
   use Mosmlcgi.cgi_field_strings if you need to access all of them.

   [%# fnm] returns the integer i if there is a string associated with
   CGI parameter fnm, and that string is parsable as ML integer i.
   Raises ParamMissing(fnm) if no string is associated with fnm.
   Raises NotInt(fnm, s) if there is a string but it is not parsable
   as an ML int.

   [%%(fnm, dflt)] returns a string associated with CGI parameter fnm
   if there is any; otherwise returns the string dflt.

   [%%#(fnm, dflt)] returns the integer i if there is a string
   associated with CGI parameter fnm, and that string is parsable as
```

```
   an ML int; otherwise returns the string dflt.


   HTML generic marks:

   [mark0 t] generates the HTML tag <t> as a wseq.

   [mark0a attr t] generates the attributed HTML tag <t attr> as a wseq.

   [mark1 t ws] generates  <t>ws</t>  as a wseq.

   [mark1a attr t ws] generates  <t attr>ws</t> as a wseq.

   [comment ws] generates  <!--ws-->  as a wseq.


   HTML documents and headers:

   [html ws] generates <HTML>ws</HTML>.

   [head ws] generates <HEAD>ws</HEAD>.

   [title ws] generates <TITLE>ws</TITLE>.

   [body ws] generates <BODY>ws</BODY>.

   [bodya attr ws] generates <BODY attr>ws</BODY>.

   [htmldoc titl ws] generates
   <HTML><HEAD><TITLE>titl</TITLE></HEAD><BODY>ws</BODY></HTML>.


   HTML headings and vertical format:

   [h1 ws] generates <H1>ws</H1>.

   [p ws] generates <P>ws</P>.

   [pa attr ws] generates <P attr>ws</P>.

   [br] generates <BR>.

   [bra attr] generates <BR attr>.

   [hr] generates <HR>.

   [hra attr] generates <HR attr>.

   [divi ws] generates <DIV>ws</DIV>.

   [divia attr ws] generates <DIV attr>ws</DIV>.

   [blockquote ws] generates <BLOCKQUOTE>ws</BLOCKQUOTE>.

   [blockquotea attr ws] generates <BLOCKQUOTE attr>ws</BLOCKQUOTE>

   [center ws] generates <CENTER>ws</CENTER>.

   [address ws] generates <ADDRESS>ws</ADDRESS>.

   [pre ws] generates <PRE>ws</PRE>.


   HTML anchors and hyperlinks:

   [ahref link ws] generates <A HREF="link">ws</A>.

   [ahrefa link attr ws] generates <A HREF="link" attr>ws</A>.

   [aname nam ws] generates <A NAME="name">ws</A>.
```

HTML text formats and style:

```
[em ws] generates <EM>ws</EM>.

[strong ws] generates <STRONG>ws</STRONG>.

[tt ws] generates <TT>ws</TT>.

[sub ws] generates <SUB>ws</SUB>.

[sup ws] generates <SUP>ws</SUP>.

[fonta attr ws] generates <FONT attr>ws</FONT>.
```

HTML lists:

```
[ul ws] generates <UL>ws</UL>.

[ula attr ws] generates <UL attr>ws</UL>.

[ol ws] generates <OL>ws</OL>.

[ola attr ws] generates <OL attr>ws</OL>.

[li ws] generates <LI>ws</LI>.

[dl ws] generates <DL>ws</DL>.

[dla attr ws] generates <DL attr>ws</DL>.

[dt ws] generates <DT>ws</DT>.

[dd ws] generates <DD>ws</DD>.
```

HTML tables:

```
[table ws] generates <TABLE>ws</TABLE>.

[tablea attr ws] generates <TABLE attr>ws</TABLE>.

[tr ws] generates <TR>ws</TR>.

[tra attr ws] generates <TR attr>ws</TR>.

[td ws] generates <TD>ws</TD>.

[tda attr ws] generates <TD attr>ws</TD>.

[th ws] generates <TH>ws</TH>.

[tha attr ws] generates <TH attr>ws</TH>.

[caption ws] generates <CAPTION>ws</CAPTION>.

[captiona attr ws] generates <CAPTION attr>ws</CAPTION>.
```

HTML images and image maps:

```
[img s] generates <IMG SRC="s">.

[imga s attr] generates <IMG SRC="s" attr>.

[map nam ws] generates <MAP NAME="name">ws</MAP>.

[mapa nam attr ws] generates <MAP NAME="name" attr>ws</MAP>.

[area { alt, coords, href, shape}] generates
    <AREA SHAPE="shape" COORDS="coords" HREF="link" ALT="desc">
when href is SOME link (where HREF is replaced by NOHREF otherwise)
```

```
and  alt  is SOME desc (where ALT is omitted otherwise).
```

HTML forms etc:

```
[form act ws] generates <FORM ACTION="act">ws</FORM>.

[forma act attr ws] generates <FORM ACTION="act" attr>ws</FORM>.

[input typ] generates <INPUT TYPE=typ>.

[inputa typ attr] generates <INPUT TYPE=typ attr>.

[intext name attr] generates <INPUT TYPE=TEXT NAME="name" attr>.

[inpassword name attr] generates <INPUT TYPE=PASSWORD NAME="name" attr>.

[incheckbox {name, value} attr] generates
<INPUT TYPE=CHECKBOX NAME="name" VALUE="value" attr>.

[inradio {name, value} attr] generates
<INPUT TYPE=RADIO NAME="name" VALUE="value" attr>.

[inreset value attr] generates <INPUT TYPE=RESET VALUE="value" attr>.

[insubmit value attr] generates <INPUT TYPE=SUBMIT VALUE="value" attr>.

[inhidden {name, value}] generates
<INPUT TYPE=HIDDEN NAME="name" VALUE="value">.

[textarea name ws] generates <TEXTAREA NAME="name">ws</TEXTAREA>.

[textareaa name attr ws] generates
<TEXTAREA NAME="name" attr>ws</TEXTAREA>.

[select name attr ws] generates <SELECT NAME="name" attr>ws</SELECT>.

[option value] generates <OPTION VALUE="value">.
```

HTML frames and framesets:

```
[frameset attr ws] generates <FRAMESET attr>ws</FRAMESET>.

[frame { src, name }] generates <FRAME SRC="src" NAME="name">.

[framea { src, name } attr] generates <FRAME SRC="src" NAME="name" attr>.
```

HTML encoding functions:

```
[urlencode s] returns the url-encoding of s.  That is, space (ASCII 32)
is replaced by '+' and every non-alphanumeric character c except
the characters - _ . is replaced by %hh, where hh is the hexadecimal
representation of the ASCII code of c.

[htmlencode s] returns the html-encoding of s.  That is, < and >
are replaced by &lt; and &gt; respectively, and & is replaced by
&amp;
```

Module Mysql

Mysql -- interface to the MySQL database server -- requires Dynlib

```
type dbconn                                 Connection to server
type dbresult                               Result of a query

exception Closed                            Connection is closed
exception Null                              Field value is NULL
```

Opening, closing, and maintaining database connections

```
val openbase : { dbhost    : string option,   database server host
                 dbname    : string option,   database name
                 dboptions : string option,   (not used by MySQL)
                 dbport    : string option,   database server port
                 dbpwd     : string option,   user passwd
                 dbtty     : string option,   (not used by MySQL)
                 dbuser    : string option    database user
               } -> dbconn

val closebase    : dbconn -> unit
val db           : dbconn -> string
val host         : dbconn -> string option
val options      : dbconn -> string
val port         : dbconn -> string
val tty          : dbconn -> string

val status       : dbconn -> bool
val reset        : dbconn -> unit
val errormessage : dbconn -> string option
```

Query execution and result set information

```
datatype dbresultstatus =
    Bad_response           (not used by mysql)
  | Command_ok             The query was a command
  | Copy_in                (not used by mysql)
  | Copy_out               (not used by mysql)
  | Empty_query
  | Fatal_error            (not used by mysql)
  | Nonfatal_error
  | Tuples_ok              The query successfully returned tuples

val execute      : dbconn -> string -> dbresult
val resultstatus : dbresult -> dbresultstatus
val ntuples      : dbresult -> int
val cmdtuples    : dbresult -> int
val nfields      : dbresult -> int
val fname        : dbresult -> int -> string
val fnames       : dbresult -> string vector
val fnumber      : dbresult -> string -> int option
```

Accessing the fields of a resultset

```
val getint       : dbresult -> int -> int -> int
val getreal      : dbresult -> int -> int -> real
val getstring    : dbresult -> int -> int -> string
val getdate      : dbresult -> int -> int -> int * int * int   Y M D
val gettime      : dbresult -> int -> int -> int * int * int   H M S
val getdatetime  : dbresult -> int -> int -> Date.date
val getbool      : dbresult -> int -> int -> bool
val isnull       : dbresult -> int -> int -> bool

datatype dynval =
    Int of int                      MySQL int4
  | Real of real                    MySQL float8 (float4)
  | String of string                MySQL text (varchar)
  | Date of int * int * int         MySQL date yyyy-mm-dd
  | Time of int * int * int         MySQL time hh:mm:ss
```

```
  | DateTime of Date.date               MySQL datetime
  | NullVal                             MySQL NULL value

val getdynfield  : dbresult -> int -> int -> dynval
val getdyntup    : dbresult -> int -> dynval vector
val getdyntups   : dbresult -> dynval vector vector
val dynval2s     : dynval -> string
```

Bulk copying to or from a table

```
val copytableto   : dbconn * string * (string -> unit) -> unit
val copytablefrom : dbconn * string * ((string -> unit) -> unit) -> unit
```

Some standard ML and MySQL types:

```
datatype dyntype =
    IntTy             ML int                MySQL int4
  | RealTy            ML real               MySQL float8, float4
  | StringTy          ML string             MySQL text, varchar
  | DateTy            ML (yyyy, mth, day)   MySQL date
  | TimeTy            ML (hh, mm, ss)       MySQL time
  | DateTimeTy        ML Date.date          MySQL datetime, abstime
  | UnknownTy

val fromtag : dyntype -> string
val ftype   : dbresult -> int -> dyntype
val ftypes  : dbresult -> dyntype Vector.vector

val applyto : 'a -> ('a -> 'b) -> 'b
```

Formatting the result of a database query as an HTML table

```
val formattable : dbresult -> Msp.wseq
val showquery   : dbconn -> string -> Msp.wseq
```

[dbconn] is the type of connections to a MySQL database.

[dbresult] is the type of result sets from MySQL queries.

[openbase { dbhost, dbport, dboptions, dbtty, dbname, dbuser, dbpwd }] opens a connection to a MySQL database server on the given host (default the local one) on the given port (default ?), to the given database (defaults to the user's login name), for the given user name (defaults to the current user's login name), and the given password (default none). The result is a connection which may be used in subsequent queries. In MySQL, unlike PostgreSQL, the dboptions and dbtty fields are not used.

[closebase dbconn] closes the database connection. No further queries can be executed.

[db dbconn] returns the name of the database.

[host dbconn] returns SOME h, where h is the database server host name, if the connection uses the Internet; returns NONE if the connection is to a socket on the local server.

[options dbconn] returns the options given when opening the database.

[port dbconn] returns the port number of the connection.

[tty dbconn] returns the name of the tty used for logging.

[status dbconn] returns true if the connection is usable, false otherwise.

[reset dbconn] attempts to close and then reopen the connection to the database server.

[errormessage dbconn] returns NONE if no error occurred, and SOME msg if an error occurred, where msg describes the error.

[execute dbconn query] sends an SQL query to the database server for execution, and returns a resultset dbres.

[resultstatus dbres] returns the status of the result set dbres. After a select query that succeeded, it will be Tuples_ok.

[ntuples dbres] returns the number of tuples in the result set after a query.

[cmdtuples dbres] returns the number of tuples affected by an insert, update, or delete SQL command.

[nfields dbres] returns the number of fields in each tuple after a query.

[fname dbres fno] returns the name of field number fno (in the result set after a query). The fields are numbered 0, 1,...

[fnames dbres] returns a vector of the field names (in the result set after a query).

[fnumber dbres fname] returns SOME i where i is the number (0, 1, ...) of the field called fname (in the result set after a query), if the result set contains such a field name; returns NONE otherwise.

[ftype dbres fno] returns the dyntype of field number fno (in the result set after a query).

[ftypes dbres] returns a vector of the dyntypes (in the result set after a query).

[fromtag dt] returns the name of the preferred MySQL type used to represent values of the dyntype dt. This may be used when building ‘create table’ statements.

[getint dbres fno tupno] returns the integer value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[getreal dbres fno tupno] returns the floating-point value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[getstring dbres fno tupno] returns the string value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[getdate dbres fno tupno] returns the date (yyyy, mth, day) value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL. Raises Fail if the field cannot be scanned as a date.

[gettime dbres fno tupno] returns the time-of-day (hh, mm, ss) value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL. Raises Fail if the field cannot be scanned as a time.

[getdatetime dbres fno tupno] returns the Date.date value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL. Raises Fail if the field cannot be scanned as a date.

[getbool dbres fno tupno] returns the boolean value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[isnull dbres fno tupno] returns true if the value of field number fno in tuple tupno of result set dbres is NULL; false otherwise.

[getdynfield dbres fno tupno] returns the value of field number fno

in tuple tupno of result set dbres as a dynval (a wrapped value). A NULL value is returned as NullVal. Note that the partial application (getdynfield dbres fno) precomputes the type of the field fno. Hence it is far more efficient to compute

```
     let val getfno = getdynfield dbres fno
     in tabulate(ntuples dbres, getfno) end
```

than to compute

```
     let fun getfno tupno = getdynfield dbres fno tupno
     in tabulate(ntuples dbres, getfno) end
```

because the latter repeatedly computes the type of the field.

[getdyntup dbres tupno] returns the fields of tuple tupno in result set dbres as a vector of dynvals.

[getdyntups dbres] returns all tuples of result set dbres as a vector of vectors of dynvals.

[dynval2s dv] returns a string representing the dynval dv.

[applyto x f] computes f(x). This is convenient for applying several functions (given in a list or vector) to the same value:

```
   map (applyto 5) (tabulate(3, getdynfield dbres))
```

equals

```
   [getdynfield dbres 0 5, getdynfield dbres 1 5, getdynfield dbres 2 5]
```

[copytableto(dbconn, tablename, put)] simulates a PostgreSQL "COPY TABLE TO" statement, applies the function put to every tuple of the table, represented as a line of text (not terminated by newline \n), and cleans up at the end. For instance, to copy the contents of a table t to a text stream s (one tuple on each line), define

```
   fun put line =
       (TextIO.output(s, line); TextIO.output(s, "\n"))
```

and execute

```
   copytableto(dbconn, "t", put).
```

[copytablefrom(dbconn, tablename, useput)] simulates a PostgreSQL "COPY TABLE FROM" statement, creates a put function for copying lines to the table, passes the put function to useput, and cleans up at the end. The put function may be called multiple times for each line (tuple); the end of each line is indicated by the newline character "\n" as usual. For instance, to copy the contents of a text stream s to a table t, define

```
   fun useput put =
       while not (TextIO.endOfStream s) do put(TextIO.inputLine s);
```

and execute

```
   copytablefrom(dbconn, "t", useput).
```

Note that TextIO.inputLine preserves the newline at the end of each line.

[formattable dbresult] returns a wseq representing an HTML table. The HTML table has a column for every field in the dbresult. The first row is a table header giving the names of the fields in the dbresult. The remaining rows correspond to the tuples in the dbresult, in the order they are provided by the database server. Null fields are shown as NULL.

[showquery dbconn query] sends the SQL query to the database server, then uses formattable to format the result of the query.

Module NJ93

NJ93 -- compatibility SML/NJ 0.93 top-level environment

```
val print     : string -> unit
```

NJ93 Integer

```
val max       : int * int -> int
val min       : int * int -> int
```

NJ93 List

```
exception Hd and Tl and Nth and NthTail
```

```
val hd        : 'a list -> 'a            Hd
val tl        : 'a list -> 'a list       Tl
val nth       : 'a list * int -> 'a      Nth
val nthtail   : 'a list * int -> 'a list  NthTail
val app       : ('a -> 'b) -> 'a list -> unit
val revapp    : ('a -> 'b) -> 'a list -> unit
val fold      : ('a * 'b -> 'b) -> 'a list -> 'b -> 'b
val revfold   : ('a * 'b -> 'b) -> 'a list -> 'b -> 'b
```

NJ93 Real

```
val ceiling   : real -> int
val truncate  : real -> int
```

NJ93 Ref

```
val inc       : int ref -> unit
val dec       : int ref -> unit
```

NJ93 String

```
exception Substring
```

```
val ordof     : string * int -> int
val ord       : string -> int                  Ord
val chr       : int -> string                  Chr
val substring : string * int * int -> string   Substring
val explode   : string -> string list
val implode   : string list -> string
```

NJ93 top-level math functions

```
val sqrt      : real -> real
val sin       : real -> real
val cos       : real -> real
val arctan    : real -> real
val exp       : real -> real
val ln        : real -> real
```

NJ93 top-level input/output, standard

```
type instream and outstream
```

```
val std_in        : instream
val open_in       : string -> instream
val input         : instream * int -> string
val lookahead     : instream -> string
val close_in      : instream -> unit
val end_of_stream : instream -> bool

val std_out       : outstream
val open_out      : string -> outstream
val output        : outstream * string -> unit
val close_out     : outstream -> unit
```

NJ93 top-level input/output, non-standard

```
val open_in_bin   : string -> instream
val open_out_bin  : string -> outstream
val inputc        : instream -> int -> string
val std_err       : outstream
val outputc       : outstream -> string -> unit
val flush_out     : outstream -> unit
val input_line    : instream -> string
val can_input     : instream * int -> bool
val open_append   : string -> outstream
```

Module Nonstdio

Nonstdio -- non-standard I/O -- use BinIO and TextIO instead

```
local open BasicIO in

val open_in_bin       : string -> instream
val buff_input        : instream -> CharArray.array -> int -> int -> int
val input_char        : instream -> char Raises Size
val input_binary_int  : instream -> int
val input_value       : instream -> 'a
val seek_in           : instream -> int -> unit
val pos_in            : instream -> int
val in_stream_length  : instream -> int
val fast_really_input : instream -> string -> int ->  int -> unit

val open_out_bin      : string -> outstream
val open_out_exe      : string -> outstream
val output_char       : outstream -> Char.char -> unit
val output_byte       : outstream -> int -> unit
val buff_output       : outstream -> CharArray.array -> int -> int -> unit
val output_binary_int : outstream -> int -> unit
val output_value      : outstream -> 'a -> unit
val seek_out          : outstream -> int -> unit
val pos_out           : outstream -> int

val file_exists       : string -> bool

end
```

Module OS

OS -- SML Basis Library

```
signature OS = sig

type syserror = syserror

exception SysErr of string * syserror option

val errorMsg      : syserror -> string

structure FileSys : FileSys
structure Path    : Path
structure Process : Process

end
```

```
   [errorMsg err] returns a string explaining the error message system
   error code err, as found in a SysErr exception.  The precise form
   of the string depends on the operating system.
```

Module Option

Option -- SML Basis Library

```
exception Option

datatype option = datatype option

val getOpt         : 'a option * 'a -> 'a
val isSome         : 'a option -> bool
val valOf          : 'a option -> 'a
val filter         : ('a -> bool) -> 'a -> 'a option
val map            : ('a -> 'b) -> 'a option -> 'b option
val app            : ('a -> unit) -> 'a option -> unit
val join           : 'a option option -> 'a option
val compose        : ('a -> 'b) * ('c -> 'a option) -> ('c -> 'b option)
val mapPartial     : ('a -> 'b option) -> ('a option -> 'b option)
val composePartial : ('a -> 'b option) * ('c -> 'a option) -> ('c -> 'b option)
```

[getOpt (xopt, d)] returns x if xopt is SOME x; returns d otherwise.

[isSome vopt] returns true if xopt is SOME x; returns false otherwise.

[valOf vopt] returns x if xopt is SOME x; raises Option otherwise.

[filter p x] returns SOME x if p x is true; returns NONE otherwise.

[map f xopt] returns SOME (f x) if xopt is SOME x; returns NONE otherwise.

[app f xopt] applies f to x if xopt is SOME x; does nothing otherwise.

[join xopt] returns x if xopt is SOME x; returns NONE otherwise.

[compose (f, g) x] returns SOME (f y) if g x is SOME y; returns NONE otherwise. It holds that compose (f, g) = map f o g.

[mapPartial f xopt] returns f x if xopt is SOME x; returns NONE otherwise. It holds that mapPartial f = join o map f.

[composePartial (f, g) x] returns f y if g x is SOME y; returns NONE otherwise. It holds that composePartial (f, g) = mapPartial f o g.

The operators (map, join, SOME) form a monad.

Module PP

PP -- pretty-printing -- from the SML/NJ library

```
type ppconsumer = { consumer  : string -> unit,
                    linewidth : int,
                    flush     : unit -> unit }

datatype break_style =
    CONSISTENT
  | INCONSISTENT

val mk_ppstream    : ppconsumer -> ppstream
val dest_ppstream  : ppstream -> ppconsumer
val add_break      : ppstream -> int * int -> unit
val add_newline    : ppstream -> unit
val add_string     : ppstream -> string -> unit
val begin_block    : ppstream -> break_style -> int -> unit
val end_block      : ppstream -> unit
val clear_ppstream : ppstream -> unit
val flush_ppstream : ppstream -> unit
val with_pp        : ppconsumer -> (ppstream -> unit) -> unit
val pp_to_string   : int -> (ppstream -> 'a -> unit) -> 'a -> string
```

This structure provides tools for creating customized Oppen-style pretty-printers, based on the type ppstream. A ppstream is an output stream that contains prettyprinting commands. The commands are placed in the stream by various function calls listed below.

There following primitives add commands to the stream: begin_block, end_block, add_string, add_break, and add_newline. All calls to add_string, add_break, and add_newline must happen between a pair of calls to begin_block and end_block must be properly nested dynamically. All calls to begin_block and end_block must be properly nested (dynamically).

[ppconsumer] is the type of sinks for pretty-printing. A value of type ppconsumer is a record

```
              { consumer  : string -> unit,
                linewidth : int,
                flush     : unit -> unit }
```

of a string consumer, a specified linewidth, and a flush function which is called whenever flush_ppstream is called.

A prettyprinter can be called outright to print a value. In addition, a prettyprinter for a base type or nullary datatype ty can be installed in the top-level system. Then the installed prettyprinter will be invoked automatically whenever a value of type ty is to be printed.

[break_style] is the type of line break styles for blocks:

[CONSISTENT] specifies that if any line break occurs inside the block, then all indicated line breaks occur.

[INCONSISTENT] specifies that breaks will be inserted to only to avoid overfull lines.

[mk_ppstream {consumer, linewidth, flush}] creates a new ppstream which invokes the consumer to output text, putting at most linewidth characters on each line.

[dest_ppstream ppstrm] extracts the linewidth, flush function, and consumer from a ppstream.

[add_break ppstrm (size, offset)] notifies the pretty-printer that a line break is possible at this point.

```
* When the current block style is CONSISTENT:
   ** if the entire block fits on the remainder of the line, then
      output size spaces; else
   ** increase the current indentation by the block offset;
```

```
      further indent every item of the block by offset, and add
      one newline at every add_break in the block.
* When the current block style is INCONSISTENT:
   ** if the next component of the block fits on the remainder of
      the line, then output size spaces; else
   ** issue a newline and indent to the current indentation level
      plus the block offset plus the offset.
```

[add_newline ppstrm] issues a newline.

[add_string ppstrm str] outputs the string str to the ppstream.

[begin_block ppstrm style blockoffset] begins a new block and level of indentation, with the given style and block offset.

[end_block ppstrm] closes the current block.

[clear_ppstream ppstrm] restarts the stream, without affecting the underlying consumer.

[flush_ppstream ppstrm] executes any remaining commands in the ppstream (that is, flushes currently accumulated output to the consumer associated with ppstrm); executes the flush function associated with the consumer; and calls clear_ppstream.

[with_pp consumer f] makes a new ppstream from the consumer and applies f (which can be thought of as a producer) to that ppstream, then flushed the ppstream and returns the value of f.

[pp_to_string linewidth printit x] constructs a new ppstream ppstrm whose consumer accumulates the output in a string s. Then evaluates (printit ppstrm x) and finally returns the string s.

Example 1: A simple prettyprinter for Booleans:

```
    load "PP";
    fun ppbool pps d =
        let open PP
        in
            begin_block pps INCONSISTENT 6;
            add_string pps (if d then "right" else "wrong");
            end_block pps
        end;
```

Now one may define a ppstream to print to, and exercise it:

```
    val ppstrm = PP.mk_ppstream {consumer  =
                                 fn s => TextIO.output(TextIO.stdOut, s),
                                 linewidth = 72,
                                 flush     =
                                  fn () => TextIO.flushOut TextIO.stdOut};

    fun ppb b = (ppbool ppstrm b; PP.flush_ppstream ppstrm);

    - ppb false;
    wrong> val it = () : unit
```

The prettyprinter may also be installed in the toplevel system; then it will be used to print all expressions of type bool subsequently computed:

```
    - installPP ppbool;
    > val it = () : unit
    - 1=0;
    > val it = wrong : bool
    - 1=1;
    > val it = right : bool
```

See library Meta for a description of installPP.

Example 2: Prettyprinting simple expressions (examples/pretty/ppexpr.sml):

```
    datatype expr =
        Cst of int
      | Neg of expr
      | Plus of expr * expr

    fun ppexpr pps e0 =
        let open PP
            fun ppe (Cst i)        = add_string pps (Int.toString i)
              | ppe (Neg e)        = (add_string pps "~"; ppe e)
              | ppe (Plus(e1, e2)) = (begin_block pps CONSISTENT 0;
                                      add_string pps "(";
                                      ppe e1;
                                      add_string pps " + ";
                                      add_break pps (0, 1);
                                      ppe e2;
                                      add_string pps ")";
                                      end_block pps)
        in
            begin_block pps INCONSISTENT 0;
            ppe e0;
            end_block pps
        end

    val _ = installPP ppexpr;
```

Some example values:

```
    val e1 = Cst 1;
    val e2 = Cst 2;
    val e3 = Plus(e1, Neg e2);
    val e4 = Plus(Neg e3, e3);
    val e5 = Plus(Neg e4, e4);
    val e6 = Plus(e5, e5);
    val e7 = Plus(e6, e6);
    val e8 =
        Plus(e3, Plus(e3, Plus(e3, Plus(e3, Plus(e3, Plus(e3, e7))))));
```

Module Parsing

Parsing -- runtime library for parsers generated by mosmlyac
Based on the runtime library for camlyacc; copyright 1993 INRIA, France

```
local open Vector Obj Lexing in

val symbolStart : unit -> int
val symbolEnd   : unit -> int
val itemStart   : int -> int
val itemEnd     : int -> int
val clearParser : unit -> unit
```

For internal use in generated parsers:

```
type parseTables =
    actions     (unit -> obj) vector  *
    transl      int vector *
    lhs         string *
    len         string *
    defred      string *
    dgoto       string *
    sindex      string *
    rindex      string *
    gindex      string *
    tablesize   int *
    table       string *
    check       string

exception yyexit of obj
exception ParseError of (obj -> bool)

val yyparse : parseTables -> int -> (lexbuf -> 'a) -> lexbuf -> 'b
val peekVal : int -> 'a

end
```

```
   These functions are for use in mosmlyac-generated parsers.  For
   further information, see the Moscow ML Owner's Manual.  For
   examples, see mosml/examples/lexyacc and mosml/examples/calc.

   A grammar definition (input to mosmlyac) consists of fragments of
   this form

       nonterm :
          grsyms1   { action1 }
        | grsyms2   { action2 }
        | grsyms3   { action3 }
        | ...

   where the grsyms are sequences of grammar symbols, matching some
   string of characters, and the actions are corresponding semantic
   actions, written in ML.  The following functions can be used in the
   semantic actions:

   [symbolStart ()] returns the start position of the string that
   matches the sequence of grammar symbols.  The first character in
   the input stream has position 0.  May be called in a semantic
   action only.

   [symbolEnd ()] returns the end position, plus one, of the string
   that matches the sequence of grammar symbols.  The first character
   in the input stream has position 0.  May be called in a semantic
   action only.

   [itemStart i] returns the start position of the string that matches
   the i'th grammar symbol in the sequence.  The first grammar symbol
   has number 1.  The first character in the input stream has position
   0.  May be called in a semantic action only.
```

```
   [itemEnd i] returns the end position, plus one, of the string that
   matches the i'th grammar symbol in the sequence.  The first grammar
   symbols has number 1.  The first character in the input stream has
   position 0.  May be called in a semantic action only.

   [clearParser ()] clears the parser stack.  It may be called after a
   parsing function has returned, to remove all pointers from the
   parser stack to structures that were built by semantic actions
   during parsing.  This is not strict necessary, but reduces the
   memory requirements of the program.
```

Module Path

OS.Path -- SML Basis Library

```
exception Path

val parentArc    : string
val currentArc   : string

val fromString   : string -> {isAbs : bool, vol : string, arcs : string list}
val toString     : {isAbs : bool, vol : string, arcs : string list} -> string

val getVolume    : string -> string
val validVolume  : {isAbs : bool, vol : string} -> bool
val getParent    : string -> string

val isAbsolute   : string -> bool
val isRelative   : string -> bool
val mkAbsolute   : string * string -> string
val mkRelative   : string * string -> string

val concat       : string * string -> string

val mkCanonical  : string -> string
val isCanonical  : string -> bool

val splitDirFile : string -> {dir : string, file : string}
val joinDirFile  : {dir : string, file : string} -> string
val dir          : string -> string
val file         : string -> string

val splitBaseExt : string -> {base : string, ext : string option}
val joinBaseExt  : {base : string, ext: string option} -> string
val base         : string -> string
val ext          : string -> string option
```

This module provides OS-independent functions for manipulating strings that represent file names and paths in a directory structure. None of these functions accesses the actual filesystem.

Definitions:

* An arc denotes a directory or file. Under Unix or DOS, an arc may have form "..", ".", "", or "abc", or similar.

* An absolute path has a root: Unix examples include "/", "/a/b"; DOS examples include "\", "\a\b", "A:\a\b".

* A relative path is one without a root: Unix examples include "..", "a/b"; DOS examples include "..", "a\b", "A:a\b".

* A path has an associated volume. Under Unix, there is only one volume, whose name is "". Under DOS, the volume names are "", "A:", "C:", and similar.

* A canonical path contains no occurrences of the empty arc "" or the current arc ".", and contains or the parent arc ".." only at the beginning and only if the path is relative.

* All functions (except concat) preserve canonical paths. That is, if all arguments are canonical, then so will the result be.

* All functions are defined so that they work sensibly on canonical paths.

* There are three groups of functions, corresponding to three ways to look at paths, exemplified by the following paths:

```
     Unix:    d/e/f/a.b.c       and     /d/e/f/a.b.c
     DOS:     A:d\e\f\a.b.c     and     A:d\e\f\a.b.c
```

(1) A path consists of a sequence of arcs, possibly preceded by a volume and a root:

```
                  vol  [--- arcs ---]       vol  root  [--- arcs ---]
 --------------------------------------------------------------------
 Unix examples:        d  e  f  a.b.c               /   d  e  f  a.b.c
 DOS examples:    A:   d  e  f  a.b         A:     \   d  e  f  a.b
```

(2) A path consists of a directory part and a (last) file name part:

```
                  directory   file           directory  file
 ----------------------------------------------------------------
 Unix examples:   d/e/f       a.b.c          /d/e/f     a.b.c
 DOS examples:    A:d\e\f     a.b            A:\d\e\f   a.b
```

(3) A path consists of a base and an extension:

```
                  base       extension       base       extension
 ----------------------------------------------------------------
 Unix examples:   d/e/f/a.b      c           /d/e/f/a.b      c
 DOS examples:    A:d\e\f\a      b           A:\d\e\f\a      b
```

GROUP 0: General functions on paths:

[parentArc] is the arc denoting a parent directory: ".." under DOS and Unix.

[currentArc] is the arc denoting the current directory: "." under DOS and Unix.

[isRelative p] returns true if p is a relative path.

[isAbsolute p] returns true if p is an absolute path. Equals not (isRelative p).

[validVolume {isAbs, vol}] returns true if vol is a valid volume name for an absolute path (if isAbs=true) resp. for a relative path (if isAbs=false). Under Unix, the only valid volume name is ""; under MS DOS and MS Windows the valid volume names are "", "a:", "b:", ..., and "A:", "B:", ...

[getParent p] returns a string denoting the parent directory of p. It holds that getParent p = p if and only if p is a root.

[concat (p1, p2)] returns the path consisting of p1 followed by p2. Does not preserve canonical paths: concat("a/b", "../c") equals "a/b/../c". This is because "a/b/../c" and "a/c" may not be equivalent in the presence of symbolic links. Raises Path if p2 is not a relative path.

[mkAbsolute(p1, p2)] returns the absolute path made by taking path p2, then p1. That is, returns p1 if p1 is absolute; otherwise returns the canonicalized concatenation of p2 and p1. Raises Path if p2 is not absolute (even if p1 is absolute).

[mkRelative(p1, p2)] returns p1 relative to p2. That is, returns p1 if p1 is already relative; otherwise returns the relative path leading from p2 to p1. Raises Path if p2 is not absolute (and even if p1 is relative), or if p1 and p2 are both absolute but have different roots.

[mkCanonical p] returns a canonical path which is equivalent to p. Redundant occurrences of the parent arc, the current arc, and the empty arc are removed. The canonical path will never be the empty string; the empty path is converted to the current directory path ("." under Unix and DOS).

[isCanonical p] is equal to (p = mkCanonical p).

GROUP 1: Manipulating volumes and arcs:

[fromString p] returns {isAbs=false, vol, arcs} if the path p is relative, and {isAbs=true, vol, arcs} if the path p is absolute. In both cases vol is the volume name and arcs is the list of (possibly empty) arcs of the path. Under Unix, the volume name is always the empty string ""; under DOS it will have form "A:", "C:", or similar.

[toString path] reconstitutes a path from its root (if any) and arcs. Raises Path if applied to a relative path whose first arc is empty. It holds that toString(fromString p) = p, except that in MS DOS, slashes "/" in p will be replaced by backslashes "\". It holds that fromString (toString p) = p when no exception is raised. It holds that isRelative(toString {isAbs=false, vol, arcs}) = true when no exception is raised.

[getVolume p] returns the volume name of the path p, if given. Under Unix and MacOS, this is always the empty string "", and under MS DOS and MS Windows, it may have form "A:", "B:", ...

GROUP 2: Manipulating directory paths and file names:

[splitDirFile p] returns {dir, file} where file is the last arc in p, and dir is the path preceding that arc. A typical use is to split a path into the directory part (dir) and the filename (file).

[joinDirFile {dir, file}] returns the path p obtained by extending the path dir with the arc file.

[dir p] equals #dir (splitDirFile p).

[file p] equals #file (splitDirFile p).

GROUP 3: Manipulating file names and extensions:

[splitBaseExt s] returns {base, ext} where ext = NONE if s has no extension, and ext = SOME e if s has extension e; base is the part of s preceding the extension. A path s is considered having no extension if its last arc contains no extension separator (typically ".") or contains an extension separator only as its leftmost character, or contains an extension separator as its right-most character. Hence none of "a.b/cd", "a/.login", "a.", "..", "." and "." has an extension.

[joinBaseExt {base, ext}] returns an arc composed of the base name and the extension (if different from NONE). It is a left inverse of splitBaseExt, so joinBaseExt (splitBaseExt s) = s, but the opposite does not hold (since the extension may be empty, or may contain extension separators).

[ext s] equals #ext (splitBaseExt s).

[base s] equals #base (splitBaseExt s).

Module Polygdbm

Polygdbm -- GNU gdbm persistent polymorphic hashtables -- requires Dynlib

```
type ('key, 'data) table

exception NotFound
exception AlreadyThere
exception NotWriter
exception Closed
exception GdbmError of string

val withtable  : string * Gdbm.openmode -> (('key, 'data) table -> 'a) -> 'a
val add        : ('key, 'data) table -> 'key * 'data -> unit
val insert     : ('key, 'data) table -> 'key * 'data -> unit
val find       : ('key, 'data) table -> 'key -> 'data
val peek       : ('key, 'data) table -> 'key -> 'data option
val hasKey     : ('key, 'data) table -> 'key -> bool
val remove     : ('key, 'data) table -> 'key -> unit
val listKeys   : ('key, 'data) table -> 'key list
val numItems   : ('key, 'data) table -> int
val listItems  : ('key, 'data) table -> ('key * 'data) list
val app        : ('key * 'data -> unit) -> ('key, 'data) table -> unit
val map        : ('key * 'data -> 'a) -> ('key, 'data) table -> 'a list
val fold       : ('key * 'data * 'a -> 'a) -> 'a -> ('key, 'data) table -> 'a
val fastwrite  : bool ref
val reorganize : ('key, 'data) table -> unit
```

[('key, 'data) table] is the type of an opened table with keys of type 'key and associated values of type 'data. The actual values of type 'key and 'data cannot contain function closures or abstract values. Values involving references (even circular values) can be stored, but the identity of references is preserved only with every single key or value stored, not across several different values.

The Polygdbm table files of are not portable across platforms, because word size and endianness affects the lay-out of values.

A value of type table can be used only in the argument f to the withtable function. This makes sure that the table is closed after use.

[withtable (nam, mod) f] first opens the table db in file nam with mode mod, then applies f to db, then closes db. Makes sure to close db even if an exception is raised during the evaluation of f(db). Raises GdbmError with an informative message in case the table cannot be opened. E.g. the table cannot be opened for reading if already opened for writing, and cannot be opened for writing if already opened for reading.

[add db (k,v)] adds the pair (k, v) to db. Raises AlreadyThere if there is a pair (k, _) in db already. Raises NotWriter if db is not opened in write mode.

[insert db (k, v)] adds the pair (k, v) to db, replacing any pair (k, _) at k if present. Raises NotWriter if db is not opened in write mode.

[find(db, k)] returns v if the pair (k, v) is in db; otherwise raises NotFound.

[peek db k] returns SOME v if the pair (k, v) is in db; otherwise returns NONE.

[hasKey(db, k)] returns true if there is a pair (k, _) in db; otherwise returns false.

[remove db k] deletes the pair (k, _) from the table if present; otherwise raises NotFound. Raises NotWriter if db is not opened in write mode.

```
[listKeys db] returns a list of all keys in db in an unspecified
order.

[numItems db] is the number of (key, value) pairs in db.
Equivalent to length(listKeys db).

[listItems db] returns a list of all (key, value) pairs in db in some
order.  Equivalent to
     List.map (fn key => (key, find(db,key))) (listKeys db)

[app f db] is equivalent to List.app f (listItems db), provided the
function f does not change the set of keys in the table.
Otherwise the effect is unpredictable.

[map f db] is equivalent to List.map f (listItems db), provided the
function f does not change the set of keys in the table.
Otherwise the result and effect are unpredictable.

[fold f a db] is equivalent to
     List.foldr (fn ((k, v), r) => f(k, v, r)) a (listItems db)
provided the function f does not change the set of keys in the
table. Otherwise the result and effect are unpredictable.

[fastwrite] can be set to speed up writes to a table.  By default,
!fastwrite is false and every write to a table will be followed by
file system synchronization.  This is safe, but slow if you perform
thousands of writes.  However, if !fastwrite is true when calling
withtable, then writes may not be followed by synchronization,
which may speed up writes considerably.  In any case, the file
system is synchronized before withtable returns.

[reorganize db] has no visible effect, but may be called after a
lot of deletions to shrink the size of the table file.
```

Module Polyhash

Polyhash -- polymorphic hashtables as in the SML/NJ Library

```
type ('key, 'data) hash_table

val mkTable     : ('_key -> int) * ('_key * '_key -> bool) -> int * exn
                  -> ('_key, '_data) hash_table
val numItems    : ('key, 'data) hash_table -> int
val insert      : ('_key, '_data) hash_table -> '_key * '_data -> unit
val peekInsert  : ('_key, '_data) hash_table -> '_key * '_data
                  -> '_data option
val find        : ('key, 'data) hash_table -> 'key -> 'data
val peek        : ('key, 'data) hash_table -> 'key -> 'data option
val remove      : ('key, 'data) hash_table -> 'key -> 'data
val listItems   : ('key, 'data) hash_table -> ('key * 'data) list
val apply       : ('key * 'data -> unit) -> ('key, 'data) hash_table -> unit
val map         : ('_key * 'data -> '_res) -> ('_key, 'data) hash_table
                  -> ('_key, '_res) hash_table
val filter      : ('key * 'data -> bool) -> ('key, 'data) hash_table -> unit
val transform   : ('data -> '_res) -> ('_key, 'data) hash_table
                  -> ('_key, '_res) hash_table
val copy        : ('_key, '_data) hash_table -> ('_key, '_data) hash_table
val bucketSizes : ('key, 'data) hash_table -> int list
```

Polymorphic hash primitives from Caml Light

```
val hash        : 'key -> int
val hash_param  : int -> int -> 'key -> int
val mkPolyTable : int * exn -> ("_key, '_data) hash_table
```

```
[('key, 'data) hash_table] is the type of hashtables with keys of type
'key and data values of type 'data.

[mkTable (hashVal, sameKey) (sz, exc)] returns a new hashtable,
using hash function hashVal and equality predicate sameKey.  The sz
is a size hint, and exc is the exception raised by function find.
It must be the case that sameKey(k1, k2) implies hashVal(k1) =
hashVal(k2) for all k1,k2.

[numItems htbl] is the number of items in the hash table.

[insert htbl (k, d)] inserts data d for key k.  If k already had an
item associated with it, then the old item is overwritten.

[find htbl k] returns d, where d is the data item associated with key k,
or raises the exception (given at creation of htbl) if there is no such d.

[peek htbl k] returns SOME d, where d is the data item associated with
key k, or NONE if there is no such d.

[peekInsert htbl (k, d)] inserts data d for key k, if k is not
already in the table, returning NONE.  If k is already in the
table, and the associated data value is d', then returns SOME d'
and leaves the table unmodified.

[remove htbl k] returns d, where d is the data item associated with key k,
removing d from the table; or raises the exception if there is no such d.

[listItems htbl] returns a list of the (key, data) pairs in the hashtable.

[apply f htbl] applies function f to all (key, data) pairs in the
hashtable, in some order.

[map f htbl] returns a new hashtable, whose data items have been
obtained by applying f to the (key, data) pairs in htbl.  The new
tables have the same keys, hash function, equality predicate, and
exception, as htbl.

[filter p htbl] deletes from htbl all data items which do not
satisfy predicate p.
```

[transform f htbl] as map, but only the (old) data values are used when computing the new data values.

[copy htbl] returns a complete copy of htbl.

[bucketSizes htbl] returns a list of the sizes of the buckets. This is to allow users to gauge the quality of their hashing function.

[hash k] returns the hash value of k, as a positive integer. If k1=k2 then hash(k1) = hash(k2), so this function can be used when creating hashtables. The application hash(k) always terminates, even on cyclic structures. (From the Caml Light implementation).

[hash_param n m k] computes a hash value for k with the same properties as for hash. The parameters n and m give more precise control over hashing. Hashing performs a depth-first, right-to-left traversal of the structure k, stopping after n meaningful nodes were encountered, or m nodes, meaningful or not, were encountered. Meaningful nodes are: integers, floating-point numbers, strings, characters, booleans, references, and constant constructors.

[mkPolyTable (sz, exc)] creates a new hashtable using the polymorphic hash function (hash) and ML equality (op =); the integer sz is a size hint and the exception exc is to be raised by find.

Module Postgres

Postgres -- interface to PostgreSQL database server -- requires Dynlib

```
type dbconn                                  Connection to server
type dbresult                                Result of a query
type oid                                     Internal object id

exception Closed                             Connection is closed
exception Null                               Field value is NULL
```

Opening, closing, and maintaining database connections

```
val openbase : { dbhost    : string option,   database server host
                 dbname    : string option,   database name
                 dboptions : string option,   options
                 dbport    : string option,   database server port
                 dbpwd     : string option,   user passwd
                 dbtty     : string option,   tty for error log
                 dbuser    : string option    database user
               } -> dbconn

val closebase    : dbconn -> unit
val db           : dbconn -> string
val host         : dbconn -> string option
val options      : dbconn -> string
val port         : dbconn -> string
val tty          : dbconn -> string

val status       : dbconn -> bool
val reset        : dbconn -> unit
val errormessage : dbconn -> string option
```

Query execution and result set information

```
datatype dbresultstatus =
    Bad_response          An unexpected response was received
  | Command_ok            The query was a command
  | Copy_in               The query was "copy <table> from ..."
  | Copy_out              The query was "copy <table> to ..."
  | Empty_query
  | Fatal_error
  | Nonfatal_error
  | Tuples_ok             The query successfully returned tuples

val execute      : dbconn -> string -> dbresult
val resultstatus : dbresult -> dbresultstatus
val ntuples      : dbresult -> int
val cmdtuples    : dbresult -> int
val nfields      : dbresult -> int
val fname        : dbresult -> int -> string
val fnames       : dbresult -> string vector
val fnumber      : dbresult -> string -> int option
```

Accessing the fields of a resultset

```
val getint       : dbresult -> int -> int -> int
val getreal      : dbresult -> int -> int -> real
val getstring    : dbresult -> int -> int -> string
val getdate      : dbresult -> int -> int -> int * int * int Y M D
val gettime      : dbresult -> int -> int -> int * int * int H M S
val getdatetime  : dbresult -> int -> int -> Date.date
val getbool      : dbresult -> int -> int -> bool
val isnull       : dbresult -> int -> int -> bool

datatype dynval =
    Bool of bool                  psql bool
  | Int of int                    psql int4
  | Real of real                  psql float8, float4
  | String of string              psql text, varchar
  | Date of int * int * int           psql date yyyy-mm-dd
  | Time of int * int * int           psql time hh:mm:ss
  | DateTime of Date.date             psql datetime
  | Oid of oid                        psql oid
  | Bytea of Word8Array.array         psql bytea
  | NullVal                           psql NULL

val getdynfield : dbresult -> int -> int -> dynval
val getdyntup   : dbresult -> int -> dynval vector
val getdyntups  : dbresult -> dynval vector vector
val dynval2s    : dynval -> string
```

Bulk copying to or from a table

```
val copytableto   : dbconn * string * (string -> unit) -> unit
val copytablefrom : dbconn * string * ((string -> unit) -> unit) -> unit
```

Some standard ML and Postgres types:

```
datatype dyntype =
    BoolTy             ML bool              psql bool
  | IntTy              ML int               psql int4
  | RealTy             ML real              psql float8, float4
  | StringTy           ML string            psql text, varchar
  | DateTy             ML (yyyy, mth, day)  psql date
  | TimeTy             ML (hh, mm, ss)      psql time
  | DateTimeTy         ML Date.date         psql datetime, abstime
  | OidTy              ML oid               psql oid
  | ByteArrTy          ML Word8Array.array  psql bytea
  | UnknownTy of oid

val fromtag : dyntype -> string
val ftype   : dbresult -> int -> dyntype
val ftypes  : dbresult -> dyntype Vector.vector

val applyto : 'a -> ('a -> 'b) -> 'b
```

Formatting the result of a database query as an HTML table

```
val formattable : dbresult -> Msp.wseq
val showquery   : dbconn -> string -> Msp.wseq
```

[dbconn] is the type of connections to a PostgreSQL database.

[dbresult] is the type of result sets from SQL queries.

[oid] is the type of PostgreSQL internal object identifiers.

[openbase { dbhost, dbport, dboptions, dbtty, dbname, dbuser, dbpwd }] opens a connection to a PostgreSQL database server on the given host (default the local one) on the given port (default 5432), with the given options (default the empty string), with error logging on the given tty (default?), to the given database (defaults to the user's login name), for the given user name (defaults to the current user's login name), and the given password (default none). The result is a connection which may be used in subsequent queries.

[closebase dbconn] closes the database connection. No further queries can be executed.

[db dbconn] returns the name of the database.

[host dbconn] returns SOME h, where h is the database server host name, if the connection uses the Internet; returns NONE if the connection is to a socket on the local server.

[options dbconn] returns the options given when opening the database.

[port dbconn] returns the port number of the connection.

[tty dbconn] returns the name of the tty used for logging.

[status dbconn] returns true if the connection is usable, false otherwise.

[reset dbconn] attempts to close and then reopen the connection to the database server.

[errormessage dbconn] returns NONE if no error occurred, and SOME msg if an error occurred, where msg describes the error.

[execute dbconn query] sends an SQL query to the database server for execution, and returns a resultset dbres.

[resultstatus dbres] returns the status of the result set dbres. After a select query that succeeded, it will be Tuples_ok.

[ntuples dbres] returns the number of tuples in the result set after a query.

[cmdtuples dbres] returns the number of tuples affected by an insert, update, or delete SQL command.

[nfields dbres] returns the number of fields in each tuple after a query.

[fname dbres fno] returns the name of field number fno (in the result set after a query). The fields are numbered 0, 1,...

[fnames dbres] returns a vector of the field names (in the result set after a query).

[fnumber dbres fname] returns SOME i where i is the number (0, 1, ...) of the field called fname (in the result set after a query), if the result set contains such a field name; returns NONE otherwise.

[ftype dbres fno] returns the dyntype of field number fno (in the result set after a query).

[ftypes dbres] returns a vector of the dyntypes (in the result set after a query).

[fromtag dt] returns the name of the preferred PostgreSQL type used to represent values of the dyntype dt. This may be used when building 'create table' statements.

[getint dbres fno tupno] returns the integer value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[getreal dbres fno tupno] returns the floating-point value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[getstring dbres fno tupno] returns the string value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[getdate dbres fno tupno] returns the date (yyyy, mth, day) value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL. Raises Fail if the field cannot be scanned as a date.

[gettime dbres fno tupno] returns the time-of-day (hh, mm, ss) value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL. Raises Fail if the field cannot be scanned as a time.

[getdatetime dbres fno tupno] returns the Date.date value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL. Raises Fail if the field cannot be scanned as a date.

[getbool dbres fno tupno] returns the boolean value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[isnull dbres fno tupno] returns true if the value of field number fno in tuple tupno of result set dbres is NULL; false otherwise.

[getdynfield dbres fno tupno] returns the value of field number fno in tuple tupno of result set dbres as a dynval (a wrapped value). A NULL value is returned as NullVal. Note that the partial application (getdynfield dbres fno) precomputes the type of the field fno. Hence it is far more efficient to compute

```
     let val getfno = getdynfield dbres fno
     in tabulate(ntuples dbres, getfno) end
```

than to compute

```
     let fun getfno tupno = getdynfield dbres fno tupno
     in tabulate(ntuples dbres, getfno) end
```

because the latter repeatedly computes the type of the field.

[getdyntup dbres tupno] returns the fields of tuple tupno in result set dbres as a vector of dynvals.

[getdyntups dbres] returns all tuples of result set dbres as a vector of vectors of dynvals.

[dynval2s dv] returns a string representing the dynval dv.

[applyto x f] computes f(x). This is convenient for applying several functions (given in a list or vector) to the same value:

```
   map (applyto 5) (tabulate(3, getdynfield dbres))
```

equals

```
   [getdynfield dbres 0 5, getdynfield dbres 1 5, getdynfield dbres 2 5]
```

[copytableto(dbconn, tablename, put)] executes a "COPY TABLE TO" statement, applies the function put to every tuple of the table, represented as a line of text (not terminated by newline \n), and cleans up at the end. For instance, to copy the contents of a table t to a text stream s (one tuple on each line), define

```
   fun put line =
       (TextIO.output(s, line); TextIO.output(s, "\n"))
```

and execute

```
   copytableto(dbconn, "t", put).
```

[copytablefrom(dbconn, tablename, useput)] executes a "COPY TABLE FROM" statement, creates a put function for copying lines to the table, passes the put function to useput, and cleans up at the end. The put function may be called multiple times for each line (tuple); the end of each line is indicated with the newline character "\n" as usual. For instance, to copy the contents of a text stream s to a table t, define

```
   fun useput put =
       while not (TextIO.endOfStream s) do put(TextIO.inputLine s);
```

and execute

```
   copytablefrom(dbconn, "t", useput).
```

Note that TextIO.inputLine preserves the newline at the end of each line.

[formattable dbresult] returns a wseq representing an HTML table. The HTML table has a column for every field in the dbresult. The first row is a table header giving the names of the fields in the dbresult. The remaining rows correspond to the tuples in the dbresult, in the order they are provided by the database server. Null fields are shown as NULL.

[showquery dbconn query] sends the SQL query to the database server, then uses formattable to format the result of the query.

Module Process

OS.Process -- SML Basis Library

```
eqtype status

val success   : status
val failure   : status

val system    : string -> status

val atExit    : (unit -> unit) -> unit
val exit      : status -> 'a
val terminate : status -> 'a

val getEnv    : string -> string option
```

Portable functions for manipulating processes.

[success] is the unique status value that signifies successful termination of a process. Note: MS DOS (sometimes) believes that all processes are successful.

[failure] is a status value that signifies an error during execution of a process. Note that in contrast to the success value, there may be several distinct failure values.

[system cmd] asks the operating system to execute command cmd, and returns a status value.

[atExit act] registers the action act to be executed when the current SML program calls Process.exit. Actions will be executed in reverse order of registration.

[exit i] executes all registered actions, then terminates the SML process with completion code i.

[terminate i] terminates the SML process with completion code i (but without executing the registered actions).

[getEnv evar] returns SOME s if the environment variable evar is defined and is associated with the string s; otherwise NONE.

Module Random

Random -- random number generator

```
type generator

val newgenseed : real -> generator
val newgen     : unit -> generator
val random     : generator -> real
val randomlist : int * generator -> real list
val range      : int * int -> generator -> int
val rangelist  : int * int -> int * generator -> int list
```

```
   [generator] is the type of random number generators, here the
   linear congruential generators from Paulson 1991, 1996.

   [newgenseed seed] returns a random number generator with the given seed.

   [newgen ()] returns a random number generator, taking the seed from
   the system clock.

   [random gen] returns a random number in the interval [0..1).

   [randomlist (n, gen)] returns a list of n random numbers in the
   interval [0,1).

   [range (min, max) gen] returns an integral random number in the
   range [min, max).  Raises Fail if min > max.

   [rangelist (min, max) (n, gen)] returns a list of n integral random
   numbers in the range [min, max).  Raises Fail if min > max.
```

Module Real

Real -- SML Basis Library

```
type real = real

exception Div
and Overflow

val ~           : real -> real
val +           : real * real -> real
val -           : real * real -> real
val *           : real * real -> real
val /           : real * real -> real
val abs         : real -> real
val min         : real * real -> real
val max         : real * real -> real
val sign        : real -> int
val compare     : real * real -> order

val sameSign    : real * real -> bool
val toDefault   : real -> real
val fromDefault : real -> real
val fromInt     : int -> real

val floor       : real -> int
val ceil        : real -> int
val trunc       : real -> int
val round       : real -> int

val >           : real * real -> bool
val >=          : real * real -> bool
val <           : real * real -> bool
val <=          : real * real -> bool
val ==          : real * real -> bool
val !=          : real * real -> bool
val ?=          : real * real -> bool

val toString    : real -> string
val fromString  : string -> real option
val scan        : (char, 'a) StringCvt.reader -> (real, 'a) StringCvt.reader
val fmt         : StringCvt.realfmt -> real -> string
```

```
   [~]
   [*]
   [/]
   [+]
   [-]
   [>]
   [>=]
   [<]
   [<=] are the usual operations on defined reals (excluding NaN and Inf).

   [abs x] is x if x >= 0, and ~x if x < 0, that is, the absolute value of x.

   [min(x, y)] is the smaller of x and y.

   [max(x, y)] is the larger of x and y.

   [sign x] is ~1, 0, or 1, according as x is negative, zero, or positive.

   [compare(x, y)] returns LESS, EQUAL, or GREATER, according
   as x is less than, equal to, or greater than y.

   [sameSign(x, y)] is true iff sign x = sign y.

   [toDefault x] is x.

   [fromDefault x] is x.

   [fromInt i] is the floating-point number representing integer i.
```

[floor r] is the largest integer <= r (rounds towards minus infinity).
May raise Overflow.

[ceil r] is the smallest integer >= r (rounds towards plus infinity).
May raise Overflow.

[trunc r] is the numerically largest integer between r and zero
(rounds towards zero). May raise Overflow.

[round r] is the integer nearest to r, using the default rounding
mode. May raise Overflow.

[==(x, y)] is equivalent to x=y in Moscow ML (because of the
absence of NaNs and Infs).

[!=(x, y)] is equivalent to x<>y in Moscow ML (because of the
absence of NaNs and Infs).

[?=(x, y)] is false in Moscow ML (because of the absence of NaNs
and Infs).

[fmt spec r] returns a string representing r, in the format
specified by spec (see below). The requested number of digits must
be >= 0 in the SCI and FIX formats and > 0 in the GEN format;
otherwise Size is raised, even in a partial application fmt(spec).

spec	description	C printf
SCI NONE	scientific, 6 digits after point	%e
SCI (SOME n)	scientific, n digits after point	%.ne
FIX NONE	fixed-point, 6 digits after point	%f
FIX (SOME n)	fixed-point, n digits after point	%.nf
GEN NONE	auto choice, 12 significant digits	%.12g
GEN (SOME n)	auto choice, n significant digits	%.ng

[toString r] returns a string representing r, with automatic choice
of format according to the magnitude of r.
Equivalent to (fmt (GEN NONE) r).

[fromString s] returns SOME(r) if a floating-point numeral can be
scanned from a prefix of string s, ignoring any initial whitespace;
returns NONE otherwise. The valid forms of floating-point numerals
are described by:

```
     [+~-]?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))([eE][+~-]?[0-9]+)?
```

[scan getc charsrc] attempts to scan a floating-point number from
the character source charsrc, using the accessor getc, and ignoring
any initial whitespace. If successful, it returns SOME(r, rest)
where r is the number scanned, and rest is the unused part of the
character source. The valid forms of floating-point numerals
are described by:

```
     [+~-]?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))([eE][+~-]?[0-9]+)?
```

Module Regex

Regex -- regular expressions a la POSIX 1003.2 -- requires Dynlib

```
exception Regex of string

type regex                     A compiled regular expression

datatype cflag =
    Extended                   Compile POSIX extended REs
  | Icase                      Compile case-insensitive match
  | Newline                    Treat \n in target string as new line

datatype eflag =
    Notbol                     Do not match ^ at beginning of string
  | Noteol                     Do not match $ at end of string

val regcomp      : string -> cflag list -> regex

val regexec      : regex -> eflag list -> string -> substring vector option
val regexecBool  : regex -> eflag list -> string -> bool

val regnexec     : regex -> eflag list -> substring
                   -> substring vector option
val regnexecBool : regex -> eflag list -> substring -> bool

val regmatch     : { pat : string, tgt : string } -> cflag list
                     -> eflag list -> substring vector option
val regmatchBool : { pat : string, tgt : string } -> cflag list
                     -> eflag list -> bool

datatype replacer =
    Str of string                      A literal string
  | Sus of int                         The i'th parenthesized group
  | Tr  of (string -> string) * int    Transformation of i'th group
  | Trs of substring vector -> string  Transformation of all groups

val replace1     : regex -> replacer list -> string -> string
val replace      : regex -> replacer list -> string -> string

val substitute1  : regex -> (string -> string) -> string -> string
val substitute   : regex -> (string -> string) -> string -> string

val tokens       : regex -> string -> substring list
val fields       : regex -> string -> substring list

val map          : regex -> (substring vector -> 'a) -> string -> 'a list
val app          : regex -> (substring vector -> unit) -> string -> unit
val fold         : regex
                   -> (substring * 'a -> 'a) * (substring vector * 'a -> 'a)
                   -> 'a -> string -> 'a
```

This structure provides pattern matching with POSIX 1003.2 regular
expressions.

The form and meaning of Extended and Basic regular expressions are
described below. Here R and S denote regular expressions; m and n
denote natural numbers; L denotes a character list; and d denotes a
decimal digit:

Extended	Basic	Meaning
c	c	Match the character c
.	.	Match any character
R*	R*	Match R zero or more times
R+	R\+	Match R one or more times
R|S	R\\\|S	Match R or S
R?	R\?	Match R or the empty string
R{m}	R\{m\}	Match R exactly m times
R{m,}	R\{m,\}	Match R at least m times

```
     R{m,n}      R\{m,n\}    Match R at least m and at most n times
     [L]         [L]         Match any character in L
     [^L]        [^L]        Match any character not in L
     ^           ^           Match at string's beginning
     $           $           Match at string's end
     (R)         \(R\)       Match R as a group; save the match
     \d          \d          Match the same as previous group d
     \\          \\          Match \ --- similarly for *.[]^$
     \+          +           Match + --- similarly for |?{}()
```

Some example character lists L:

```
     [aeiou]         Match vowel: a or e or i or o or u
     [0-9]           Match digit: 0 or 1 or 2 or ... or 9
     [^0-9]          Match non-digit
     [-+*/^]         Match - or + or * or / or ^
     [-a-z]          Match lowercase letter or hyphen (-)
     [0-9a-fA-F]     Match hexadecimal digit
     [[:alnum:]]     Match letter or digit
     [[:alpha:]]     Match letter
     [[:cntrl:]]     Match ASCII control character
     [[:digit:]]     Match decimal digit; same as [0-9]
     [[:graph:]]     Same as [:print:] but not [:space:]
     [[:lower:]]     Match lowercase letter
     [[:print:]]     Match printable character
     [[:punct:]]     Match punctuation character
     [[:space:]]     Match SML #" ", #"\r", #"\n", #"\t", #"\v", #"\f"
     [[:upper:]]     Match uppercase letter
     [[:xdigit:]]    Match hexadecimal digit; same as [0-9a-fA-F]
     [[:lower:]æøå]  Match lowercase Danish letters (ISO Latin 1)
```

Remember that backslash (\) must be escaped as "\\" in SML strings.

[regcomp pat cflags] returns a compiled representation of the regular expression pat. Raises Regex in case of failure.

[cflag] is the type of compilation flags with the following meanings:

```
[Extended] : compile as POSIX extended regular expression.
[Icase]    : compile case-insensitive match.
[Newline]  : make the newline character \n significant, so ^ matches
             just after newline (\n), and $ matches just before \n.
```

Example: Match SML integer constant:
```
regcomp "^~?[0-9]+$" [Extended]
```

Example: Match SML alphanumeric identifier:
```
regcomp "^[a-zA-Z0-9][a-zA-Z0-9'_]*$" [Extended]
```

Example: Match SML floating-point constant:
```
regcomp "^[+~]?[0-9]+(\\.[0-9]+|(\\.[0-9]+)?[eE][+~]?[0-9]+)$" [Extended]
```

Example: Match any HTML start tag; make the tag's name into a group:
```
regcomp "<([[:alnum:]]+)[^>]*>" [Extended]
```

[regexec regex eflags s] returns SOME(vec) if some substring of s matches regex, NONE otherwise. In case of success, vec is the match vector, a vector of substrings such that vec[0] is the (longest leftmost) substring of s matching regex, and vec[1], vec[2], ... are substrings matching the parenthesized groups in pat (numbered 1, 2, ... from left to right in the order of their opening parentheses). For a group that does not take part in the match, such as (ab) in "(ab)|(cd)" when matched against the string "xcdy", the corresponding substring is the empty substring at the beginning of the underlying string. For a group that takes part in the match repeatedly, such as the group (b+) in "(a(b+))*" when matched against "babbabbb", the corresponding substring is the last (rightmost) one matched.

[eflag] is the type of end flags with the following meaning:

[Notbol] : do not match ^ at beginning of string.

[Noteol] : do not match $ at end of string.

[regexecBool regex eflags s] returns true if some substring of s matches regex, false otherwise. Equivalent to, but faster than, Option.isSome(regexec regexec eflags s).

[regnexec regex eflags sus] returns SOME(vec) if some substring of sus matches regex, NONE otherwise. The substrings returned in the vector vec will have the same base string as sus. Useful e.g. for splitting a string into fragments separated by substrings matching some regular expression.

[regnexecBool regex eflags sus] returns true if some substring of sus matches regex, false otherwise. Equivalent to, but faster than, Option.isSome(regnexec regexec eflags sus).

```
[regmatch { pat, tgt } cflags eflags] is equivalent to
      regexec (regcomp pat cflags) eflags tgt
but more efficient when the compiled regex is used only once.
```

```
[regmatchBool { pat, tgt } cflags eflags] is equivalent to
      regexecBool (regcomp pat cflags) eflags tgt
but more efficient when the compiled regex is used only once.
```

[replace regex repl s] finds the (disjoint) substrings of s matching regex from left to right, and returns the string obtained from s by applying the replacer list repl to every such substring (see below). Raises Regex if it fails to make progress in decomposing s, that is, if regex matches an empty string at the head of s or immediately after a previous regex match.
Example use: delete all HTML tags from s:
```
     replace (regcomp "<[^>]+>" [Extended]) [] s
```

[replace1 regex repl s] finds the leftmost substring b1 of s matching regex, and returns the string resulting from s by applying the replacer list repl to the match vector vec1 (see below).

Let x0 be a substring matching the entire regex and xi be the substring matching the i'th parenthesized group in regex; thus xi = vec[i] where vec is the match vector (see regexec above). Then a single replacer evaluates to a string as follows:

```
[Str s]      gives the string  s
[Sus i]      gives the string  xi
[Tr (f, i)]  gives the string  f(xi)
[Trs f]      gives the string  f(vec)
```

A replacer list repl evaluates to the concatenation of the results of the replacers. The replacers are applied from left to right.

[substitute regex f s] finds the (disjoint) substrings b1, ..., bn of s matching regex from left to right, and returns the string obtained from s by replacing every bi by f(bi). Function f is applied to the matching substrings from left to right. Raises Regex if it fails to make progress in decomposing s. Equivalent to
```
     replace regex [Tr (f, 0)] s
```

[substitute1 regex f s] finds the leftmost substring b of s matching regex, and returns the string obtained from s by replacing that substring by f(b). Equivalent to
```
     replace1 regex [Tr (f, 0)] s
```

[map regex f s] finds the (disjoint) substrings of s matching regex from left to right, applies f to the match vectors vec1, ..., vecn, and returns the list [f(vec1), ..., f(vecn)]. Raises Regex if it fails to make progress in decomposing s.

[app regex f s] finds the (disjoint) substrings of s matching regex from left to right, and applies f to the match vectors vec1, ..., vecn. Raises Regex if the regex fails to make progress in decomposing s.

```
[fields regex s] returns the list of fields in s, from left to
right.  A field is a (possibly empty) maximal substring of s not
containing any delimiter.  A delimiter is a maximal substring that
matches regex.  The eflags Notbol and Noteol are set.  Raises Regex
if it fails to make progress in decomposing s.
Example use:
     fields (regcomp " *; *" []) "56; 23 ; 22;; 89; 99"

[tokens regex s] returns the list of tokens in s, from left to
right.  A token is a non-empty maximal substring of s not
containing any delimiter.  A delimiter is a maximal substring that
matches regex.  The eflags Notbol and Noteol are set.  Raises Regex
if it fails to make progress in decomposing s.  Equivalent to
     List.filter (not o Substring.isEmpty) (fields regex s)

Two tokens may be separated by more than one delimiter, whereas two
fields are separated by exactly one delimiter.  If the only delimiter
is the character #"|", then
     "abc||def" contains three fields: "abc" and "" and "def"
     "abc||def" contains two tokens:   "abc" and "def"

[fold regex (fa, fb) e s] finds the (disjoint) substrings b1, ...,
bn of s matching regex from left to right, and splits s into the
substrings
     a0, b1, a1, b2, a2, ..., bn, an
where n >= 0 and where a0 is the (possibly empty) substring of s
preceding the first match, and ai is the (possibly empty) substring
between the matches bi and b(i+1).  Then it computes and returns
     fa(an, fb(vecn, ..., fa(a1, fb(vec1, fa(a0, e))) ...))
where veci is the match vector corresponding to bi.  Raises Regex
if it fails to make progress in decomposing s.

If we define the auxiliary functions
     fun fapp f (x, r) = f x :: r
     fun get i vec = Substring.string(Vector.sub(vec, i))
then
     map regex f s  = List.rev (fold regex (#2, fapp f) [] s)
     app regex f s  = fold regex (ignore, f o #1) () s
     fields regex s = List.rev (fold regex (op ::, #2) [] s)
     substitute regex f s =
        Substring.concat(List.rev
           (fold regex (op ::, fapp (Substring.all o f o get 0)) [] s))
```

Module SML90

SML90 -- part of the initial basis of the 1990 Definition

Math

```
val sqrt    : real -> real
val sin     : real -> real
val cos     : real -> real
val arctan  : real -> real
val exp     : real -> real
val ln      : real -> real
```

Strings

```
val chr     : int -> string
val ord     : string -> int

val explode : string -> string list
val implode : string list -> string

exception Abs
      and Diff
      and Exp
      and Floor
      and Neg
      and Prod
      and Sum
      and Mod
      and Quot
```

Input/output

```
type instream and outstream

val std_in        : instream
val open_in       : string -> instream
val input         : instream * int -> string
val lookahead     : instream -> string
val close_in      : instream -> unit
val end_of_stream : instream -> bool

val std_out       : outstream
val open_out      : string -> outstream
val output        : outstream * string -> unit
val close_out     : outstream -> unit
```

Module Signal

Signal -- SML Basis Library

```
eqtype signal

val abrt : signal
val alrm : signal
val bus  : signal
val fpe  : signal
val hup  : signal
val ill  : signal
val int  : signal
val kill : signal
val pipe : signal
val quit : signal
val segv : signal
val term : signal
val usr1 : signal
val usr2 : signal
val chld : signal
val cont : signal
val stop : signal
val tstp : signal
val ttin : signal
val ttou : signal

val toWord   : signal -> Word.word
val fromWord : Word.word -> signal
```

```
   [signal] is the type of Unix/Posix-style signals, which can be sent
   to another process.

   [toWord sig] returns the signal number as an unsigned word.

   [fromWord w] returns the signal whose number is w.

   [abrt] is SIGABRT, the abort signal from abort(3).

   [alrm] is SIGALRM, a timer signal from alarm(1).

   [bus]  is SIGBUS, a bus error.

   [fpe]  is SIGFPE, a floating point exception.

   [hup]  is SIGHUP, a hangup.

   [ill]  is SIGILL, an illegal instruction.

   [int]  is SIGINT, an interrupt.

   [kill] is SIGKILL, the kill signal.

   [pipe] is SIGPIPE, a broken pipe.

   [quit] is SIGQUIT, a quit from keyboard.

   [segv] is SIGSEGV, a segmentation violation.

   [term] is SIGTERM, the termination signal.

   [usr1] is SIGUSR1, the first user signal.

   [usr2] is SIGUSR2, the second user signal.

   [chld] is SIGCHLD, child process stopped or terminated.

   [cont] is SIGCONT, continue if stopped.

   [stop] is SIGSTOP, signal to stop process.
```

```
   [tstp] is SIGTSTP, a stop signal typed at the tty.

   [ttin] is SIGTTIN, tty input for background process.

   [ttou] is SIGTTOU, tty output for background process.
```

Module Socket

```
Socket -- SML Basis Library -- requires Dynlib

type ('addressfam, 'socktype) sock
type 'addressfam sock_addr

Socket types
type dgram                        A datagram socket
type 'a stream                    A stream socket
type passive                      A passive stream
type active                       An active, connected, stream

Socket protocol families
type pf_file                      The Unix file protocol family
type pf_inet                      The Internet protocol family

Address constructors
val fileAddr   : string -> pf_file sock_addr
val inetAddr   : string -> int -> pf_inet sock_addr

Socket constructors
val fileStream : unit -> (pf_file, 'a stream) sock
val fileDgram  : unit -> (pf_file, dgram) sock
val inetStream : unit -> (pf_inet, 'a stream) sock
val inetDgram  : unit -> (pf_inet, dgram) sock

val accept     : ('a, passive stream) sock
                 -> ('a, active stream) sock * 'a sock_addr
val bind       : ('a, 'b) sock * 'a sock_addr -> unit
val connect    : ('a, 'b) sock * 'a sock_addr -> unit
val listen     : ('a, passive stream) sock * int -> unit
val close      : ('a, 'b) sock -> unit

Socket management
datatype shutdown_mode =
    NO_RECVS                      No further receives
  | NO_SENDS                      No further sends
  | NO_RECVS_OR_SENDS             No receives nor sends

val shutdown   : ('a, 'b stream) sock * shutdown_mode -> unit

type sock_desc

val sockDesc   : ('a, 'b) sock -> sock_desc
val sameDesc   : sock_desc * sock_desc -> bool
val compare    : sock_desc * sock_desc -> order
val select     :
    { rds : sock_desc list, wrs : sock_desc list, exs : sock_desc list,
      timeout : Time.time option }
    -> { rds : sock_desc list, wrs : sock_desc list, exs : sock_desc list }

val getinetaddr : pf_inet sock_addr -> string

Sock I/O option types
type out_flags = { don't_route : bool, oob : bool }
type in_flags  = { peek : bool, oob : bool }

type 'a buf = { buf : 'a, ofs : int, size : int option }


Socket output operations
val sendVec    : ('a, active stream) sock * Word8Vector.vector buf -> int
val sendArr    : ('a, active stream) sock * Word8Array.array buf -> int
val sendVec'   : ('a, active stream) sock * Word8Vector.vector buf
                 * out_flags -> int
val sendArr'   : ('a, active stream) sock * Word8Array.array buf
                 * out_flags -> int
val sendVecTo  : ('a, dgram) sock * 'a sock_addr * Word8Vector.vector buf
                 -> int
val sendArrTo  : ('a, dgram) sock * 'a sock_addr * Word8Array.array buf
```

```
                  -> int
val sendVecTo' : ('a, dgram) sock * 'a sock_addr * Word8Vector.vector buf
                 * out_flags -> int
val sendArrTo' : ('a, dgram) sock * 'a sock_addr * Word8Array.array buf
                 * out_flags -> int

Socket input operations
val recvVec      : ('a, active stream) sock * int -> Word8Vector.vector
val recvArr      : ('a, active stream) sock * Word8Array.array buf -> int
val recvVec'     : ('a, active stream) sock * int * in_flags
                   -> Word8Vector.vector
val recvArr'     : ('a, active stream) sock * Word8Array.array buf * in_flags
                   -> int
val recvVecFrom  : ('a, dgram) sock * int
                   -> Word8Vector.vector * 'a sock_addr
val recvArrFrom  : ('a, dgram) sock * Word8Array.array buf
                   -> int * 'a sock_addr
val recvVecFrom' : ('a, dgram) sock * int * in_flags
                   -> Word8Vector.vector * 'a sock_addr
val recvArrFrom' : ('a, dgram) sock * Word8Array.array buf * in_flags
                   -> int * 'a sock_addr
```

```
   Structure Socket defines functions for creating and using sockets,
   a means for communication between SML processes on the same machine
   or via a network.

   [('addressfam, 'socktype) sock] is the type of sockets with address
   family 'addressfam and having type 'socktype.

   ['addressfam sock_addr] is the type of sockets addresses.

   The possible address (protocol) families are

        type pf_file    The Unix address family (file)
        type pf_inet    The Internet address family

   The possible socket types are
        type dgram      datagram sockets
        type 'a stream  stream sockets
        type passive    passive stream sockets
        type active     active, or connected, stream sockets

   [fileAddr fname] returns a socket address for the Unix protocol
   family, created from the given file name fname.

   [inetAddr inetaddr portno] returns a socket address for the
   Internet protocol family, created from the given Internet number
   (e.g. "130.225.40.253") and port number (e.g. 8080).

   [fileStream ()] returns a new stream socket for the Unix protocol
   family.

   [fileDgram ()] returns a new datagram socket for the Unix protocol
   family.

   [inetStream ()] returns a new stream socket for the Internet
   protocol family.

   [inetDgram ()] returns a new datagram socket for the Internet
   protocol family.

   [accept sock] extracts the first connection on the queue of pending
   connections to sock.  Returns (sock', addr) where sock' is a copy
   of the socket sock, bound to that connection, and addr is the
   address of the communications counterpart (the other end of the
   connection).  Blocks if no connections are pending.  The stream
   socket sock must have been assigned a name (with bind) and must be
   listening for connections (following a call to listen).

   [bind sock addr] binds the socket sock to the address addr, that
   is, assigns the name addr to the socket.  Binding a name in the
```

Unix protocol family creates a socket in the file system that must be deleted when it is no longer needed

[connect (sock, addr)] attempts to connect socket sock to the communications peer at address addr. If sock is a datagram socket, then addr is the address to which datagrams is to be sent, and the only address from which datagrams will be accepted. If sock is a stream socket, then addr specifies another socket to which to connect.

[listen (sock, queuelen)] enables the passive stream socket sock to accept incoming connections. The parameter queuelen specifies the maximal number of pending connections. Further connections from clients may be refised when this limit is reached.

[close sock] closes the socket.

[shutdown sock shutdown_mode] shuts down socket sock for further communication, as specified by the shutdown_mode parameter:

[NO_RECVS]	no further receives are allowed;
[NO_SENDS]	no further sends are allowed;
[NO_RECVS_OR_SENDS]	no further receives or sends are allowed.

[getinetaddr addr] returns the Internet number (e.g. "130.225.40.253") of the Internet socket address addr.

['a buf] is the type of records { buf, ofs, size } which represent subvectors or subarrays:
if size = SOME s it represents buf[ofs..ofs+s-1];
if size = NONE   it represents buf[ofs..len-1] where len is buf's length.
When the subbuffer is used in a call, exception Subscript will be raised if ofs < 0 or size < 0 or ofs+size > len.

[sendVec (sock, vecbuf)] transmits the bytes from buffer vecbuf on the active stream socket sock. Returns the number of bytes sent. Blocks until sufficient space is available at the socket.

[sendArr (sock, arrbuf)] is analogous til sendVec.

[sendVec' (sock, vecbuf, out_flags)] transmits the bytes from buffer vecbuf on the active stream socket sock, observing the out_flags. Returns the number of bytes sent. Blocks until sufficient space is available at the socket.

[out_flags] is the type of records { don't_route, oob } in which the field don't_route specifies whether routing should be bypassed, and the field oob specifies whether data should be sent out-of-band.

[sendArr' (sock, arrbuf, out_flags)] is analogous til sendVec'.

[sendVecTo (sock, addr, vecbuf)] transmits the bytes from buffer vecbuf on the datagram socket sock to the target address addr. Returns the number of bytes sent. Blocks until sufficient space is available at the socket.

[sendArrTo (sock, addr, arrbuf)] is analogous til sendVecTo.

[sendVecTo' (sock, addr, vecbuf, out_flags)] transmits the bytes from buffer vecbuf on the datagram socket sock to the target address addr, observing the out_flags. Returns the number of bytes sent. Blocks until sufficient space is available at the socket. See above for a description of vecbuf and out_flags.

[sendArrTo' (sock, addr, arrbuf, out_flags)] is analogous til sendVecTo'.

[recvVec (sock, n)] receives up to n bytes from the active stream socket sock. Returns a byte vector containing the bytes actually received. Blocks until some data become available at the socket, then returns any available data, up to n bytes. Excess data are

not lost; they are available for subsequent receive calls.

[recvArr (sock, arrbuf)] receives bytes from the active stream socket sock into the subarray arrbuf, up to the available space. If #size(arrbuf) = SOME(s) the available space is s bytes; if #size(arrbuf) = NONE the available space is len - #ofs(arrbuf) bytes. Returns the number of bytes actually received. Blocks until some data become available at the socket. Excess data are not lost; they are available for subsequent receive calls.

[recvVec' (sock, n, in_flags)] receives up to n bytes from the active stream socket sock, observing the in_flags. Returns a byte vector containing the bytes actually received. Blocks until some data become available at the socket, then returns any available data, up to n bytes. Data in excess of n bytes are not lost; they are available for subsequent receive calls.

[in_flags] is the type of records { peek, oob } in which the field peek specifies that the data read should not be removed from the receive queue, and the field oob specifies that data may be received out-of-band.

[recvArr' (sock, arrbuf, in_flags)] receives bytes from the active stream socket sock into the subarray arrbuf, observing the in_flags, up to the available space.. Returns the number of bytes actually received. Blocks until some data become available at the socket. Excess data are not lost; they are available for subsequent receive calls.

[recvVecFrom (sock, n)] receives up to n bytes from the datagram socket sock. Returns a byte vector containing the bytes actually received. Blocks until some data become available at the socket, then returns any available data, up to n bytes.

[recvArrFrom (sock, arrbuf)] receives bytes from the datagram socket sock into the subarray arrbuf. Returns the number of bytes actually received. Blocks until some data become available at the socket.

[recvVecFrom' (sock, n, in_flags)] receives up to n bytes from the datagram socket sock, observing the in_flags (see above). Returns (vec, addr) where vec is a byte vector containing the bytes actually received, and addr is the source address of the message. Blocks until some data become available at the socket, then returns any available data, up to n bytes.

[recvArrFrom' (sock, arrbuf, in_flags)] receives bytes from the datagram socket sock into the array buffer arrbuf, observing the in_flags (see above). Returns (n, addr) where n is the number of bytes actually received, and addr is the source address of the message. Blocks until some data become available at the socket.

[sockDesc sock] returns a descriptor for the socket sock, to be used in a call to select.

[compare (sd1, sd2)] compares sd1 and sd2 according to an unspecified total ordering, and returns LESS if sd1 precedes sd2, returns GREATER is sd1 precedes sd2, and returns EQUAL otherwise.

[sameDesc (sd1, sd2)] returns true if sd1 and sd2 describe the same socket. Equivalent to compare(sd1, sd2) = EQUAL.

[select { rds, wrs, exs, timeout }] blocks the calling process until some input/output operations become possible on some sockets. The call will check the sockets described in rds for reading, those in wrs for writing, and those in exs for exceptional conditions. Returns { rds, wrs, exs } where rds now is a list of descriptors of sockets ready for reading, wrs are ready for writing, and exs have exceptional conditions. The order of the socket descriptors in the results is the same as their order in the corresponding arguments. If timeout is NONE then the call blocks until some input/output operations become possible; if timeout is SOME(t) then the call blocks for at most time t.

A server socket is considered ready for reading if there is a pending connection which can be accepted with 'accept'. A client socket is ready for writing when its connection is fully established.

Module Splaymap

Splaymap -- applicative maps implemented by splay-trees
From SML/NJ lib 0.2, copyright 1993 by AT&T Bell Laboratories

```
type ('key, 'a) dict

exception NotFound

val mkDict    : ('_key * '_key -> order) -> ('_key, '_a) dict
val insert    : ('_key, '_a) dict * '_key * '_a -> ('_key, '_a) dict
val find      : ('key, 'a) dict * 'key -> 'a
val peek      : ('key, 'a) dict * 'key -> 'a option
val remove    : ('_key, '_a) dict * '_key -> ('_key, '_a) dict * '_a
val numItems  : ('key, 'a) dict -> int
val listItems : ('key, 'a) dict -> ('key * 'a) list
val app       : ('key * 'a -> unit) -> ('key,'a) dict -> unit
val revapp    : ('key * 'a -> 'b) -> ('key,'a) dict -> unit
val foldr     : ('key * 'a * 'b -> 'b)-> 'b -> ('key,'a) dict -> 'b
val foldl     : ('key * 'a * 'b -> 'b) -> 'b -> ('key,'a) dict -> 'b
val map       : ('_key * 'a -> '_b) -> ('_key,'a) dict -> ('_key, '_b) dict
val transform : ('a -> '_b) -> ('_key,'a) dict -> ('_key, '_b) dict
```

[('key, 'a) dict] is the type of applicative maps from domain type 'key to range type 'a, or equivalently, applicative dictionaries with keys of type 'key and values of type 'a. They are implemented as ordered splay-trees (Sleator and Tarjan).

[mkDict ordr] returns a new, empty map whose keys have ordering ordr.

[insert(m, i, v)] extends (or modifies) map m to map i to v.

[find (m, k)] returns v if m maps k to v; otherwise raises NotFound.

[peek(m, k)] returns SOME v if m maps k to v; otherwise returns NONE.

[remove(m, k)] removes k from the domain of m and returns the modified map and the element v corresponding to k. Raises NotFound if k is not in the domain of m.

[numItems m] returns the number of entries in m (that is, the size of the domain of m).

[listItems m] returns a list of the entries (k, v) of keys k and the corresponding values v in m, in increasing order of k.

[app f m] applies function f to the entries (k, v) in m, in increasing order of k (according to the ordering ordr used to create the map or dictionary).

[revapp f m] applies function f to the entries (k, v) in m, in decreasing order of k.

[foldl f e m] applies the folding function f to the entries (k, v) in m, in increasing order of k.

[foldr f e m] applies the folding function f to the entries (k, v) in m, in decreasing order of k.

[map f m] returns a new map whose entries have form (k, f(k,v)), where (k, v) is an entry in m.

[transform f m] returns a new map whose entries have form (k, f v), where (k, v) is an entry in m.

Module Splayset

Splayset -- applicative sets implemented by splay-trees
From SML/NJ lib 0.2, copyright 1993 by AT&T Bell Laboratories

```
type 'item set

exception NotFound

val empty        : ('_item * '_item -> order) -> '_item set
val singleton    : ('_item * '_item -> order) -> '_item -> '_item set
val add          : '_item set * '_item -> '_item set
val addList      : '_item set * '_item list -> '_item set
val retrieve     : 'item set * 'item -> 'item
val peek         : 'item set * 'item -> 'item option
val isEmpty      : 'item set -> bool
val equal        : 'item set * 'item set -> bool
val isSubset     : 'item set * 'item set -> bool
val member       : 'item set * 'item -> bool
val delete       : '_item set * '_item -> '_item set
val numItems     : 'item set ->  int
val union        : '_item set * '_item set -> '_item set
val intersection : '_item set * '_item set -> '_item set
val difference   : '_item set * '_item set -> '_item set
val listItems    : 'item set -> 'item list
val app          : ('item -> unit) -> 'item set -> unit
val revapp       : ('item -> unit) -> 'item set -> unit
val foldr        : ('item * 'b -> 'b) -> 'b -> 'item set -> 'b
val foldl        : ('item * 'b -> 'b) -> 'b -> 'item set -> 'b
val find         : ('item -> bool) -> 'item set -> 'item option
```

```
   ['item set] is the type of sets of ordered elements of type 'item.
   The ordering relation on the elements is used in the representation
   of the set.  The result of combining two sets with different
   underlying ordering relations is undefined.  The implementation
   uses splay-trees (Sleator and Tarjan).

   [empty ordr] creates a new empty set with the given ordering
   relation.

   [singleton ordr i] creates the singleton set containing i, with the
   given ordering relation.

   [add(s, i)] adds item i to set s.

   [addList(s, xs)] adds all items from the list xs to the set s.

   [retrieve(s, i)] returns i if it is in s; raises NotFound otherwise.

   [peek(s, i)] returns SOME i if i is in s; returns NONE otherwise.

   [isEmpty s] returns true if and only if the set is empty.

   [equal(s1, s2)] returns true if and only if the two sets have the
   same elements.

   [isSubset(s1, s2)] returns true if and only if s1 is a subset of s2.

   [member(s, i)] returns true if and only if i is in s.

   [delete(s, i)] removes item i from s.  Raises NotFound if i is not in s.

   [numItems s] returns the number of items in set s.

   [union(s1, s2)] returns the union of s1 and s2.

   [intersection(s1, s2)] returns the intersectionof s1 and s2.

   [difference(s1, s2)] returns the difference between s1 and s2 (that
   is, the set of elements in s1 but not in s2).
```

```
   [listItems s] returns a list of the items in set s, in increasing
    order.

   [app f s] applies function f to the elements of s, in increasing
   order.

   [revapp f s] applies function f to the elements of s, in decreasing
   order.

   [foldl f e s] applies the folding function f to the entries of the
   set in increasing order.

   [foldr f e s] applies the folding function f to the entries of the
   set in decreasing order.

   [find p s] returns SOME i, where i is an item in s which satisfies
   p, if one exists; otherwise returns NONE.
```

Module String

String -- SML Basis Library

```
local
    type char = Char.char
in
    type string = string
    val maxSize   : int
    val size      : string -> int
    val sub       : string * int -> char
    val substring : string * int * int -> string
    val extract   : string * int * int option -> string
    val concat    : string list -> string
    val ^         : string * string -> string
    val str       : char -> string
    val implode   : char list -> string
    val explode   : string -> char list

    val map       : (char -> char) -> string -> string
    val translate : (char -> string) -> string -> string
    val tokens    : (char -> bool) -> string -> string list
    val fields    : (char -> bool) -> string -> string list
    val isPrefix  : string -> string -> bool

    val compare   : string * string -> order
    val collate   : (char * char -> order) -> string * string -> order

    val fromString  : string -> string option     ML escape sequences
    val toString    : string -> string            ML escape sequences
    val fromCString : string -> string option     C escape sequences
    val toCString   : string -> string            C escape sequences

    val <  : string * string -> bool
    val <= : string * string -> bool
    val >  : string * string -> bool
    val >= : string * string -> bool
end
```

```
   [string] is the type of immutable strings of characters, with
   constant-time indexing.

   [maxSize] is the maximal number of characters in a string.

   [size s] is the number of characters in string s.

   [sub(s, i)] is the i'th character of s, counting from zero.
   Raises Subscript if i<0 or i>=size s.

   [substring(s, i, n)] is the string s[i..i+n-1].  Raises Subscript
   if i<0 or n<0 or i+n>size s.  Equivalent to extract(s, i, SOME n).

   [extract (s, i, NONE)] is the string s[i..size s-1].
   Raises Subscript if i<0 or i>size s.

   [extract (s, i, SOME n)] is the string s[i..i+n-1].
   Raises Subscript if i<0 or n<0 or i+n>size s.

   [concat ss] is the concatenation of all the strings in ss.
   Raises Size if the sum of their sizes is greater than maxSize.

   [s1 ^ s2] is the concatenation of strings s1 and s2.

   [str c] is the string of size one which contains the character c.

   [implode cs] is the string containing the characters in the list cs.
   Equivalent to  concat (List.map str cs).

   [explode s] is the list of characters in the string s.

   [map f s] applies f to every character of s, from left to right,
```

```
   and returns the string consisting of the resulting characters.
   Equivalent to  CharVector.map f s
          and to  implode (List.map f (explode s)).

   [translate f s] applies f to every character of s, from left to
   right, and returns the concatenation of the resulting strings.
   Raises Size if the sum of their sizes is greater than maxSize.
   Equivalent to concat (List.map f (explode s)).

   [tokens p s] returns the list of tokens in s, from left to right,
   where a token is a non-empty maximal substring of s not containing
   any delimiter, and a delimiter is a character satisfying p.

   [fields p s] returns the list of fields in s, from left to right,
   where a field is a (possibly empty) maximal substring of s not
   containing any delimiter, and a delimiter is a character satisfying p.

   Two tokens may be separated by more than one delimiter, whereas two
   fields are separated by exactly one delimiter.  If the only delimiter
   is the character #"|", then
        "abc||def" contains two tokens:   "abc" and "def"
        "abc||def" contains three fields: "abc" and "" and "def"

   [isPrefix s1 s2] is true if s1 is a prefix of s2.
   That is, if there exists a string t such that s1 ^ t = s2.

   [fromString s] scans the string s as an ML source program string,
   converting escape sequences into the appropriate characters.  Does
   not skip leading whitespace.

   [toString s] returns a string corresponding to s, with
   non-printable characters replaced by ML escape sequences.
   Equivalent to String.translate Char.toString.

   [fromCString s] scans the string s as a C source program string,
   converting escape sequences into the appropriate characters.  Does
   not skip leading whitespace.

   [toCString s] returns a string corresponding to s, with
   non-printable characters replaced by C escape sequences.
   Equivalent to String.translate Char.toCString.

   [compare (s1, s2)] does lexicographic comparison, using the
   standard ordering Char.compare on the characters.  Returns LESS,
   EQUAL, or GREATER, according as s1 is less than, equal to, or
   greater than s2.

   [collate cmp (s1, s2)] performs lexicographic comparison, using the
   given ordering cmp on characters.

   [<]
   [<=]
   [>]
   [>=] compare strings lexicographically, using the representation
   ordering on characters.
```

Module StringCvt

```
StringCvt -- SML Basis Library

datatype radix = BIN | OCT | DEC | HEX

datatype realfmt =
    SCI of int option   scientific,  arg = # dec. digits, dflt=6
  | FIX of int option   fixed-point, arg = # dec. digits, dflt=6
  | GEN of int option   auto choice of the above,
                        arg = # significant digits, dflt=12

type cs                 character source state

type ('a, 'b) reader = 'b -> ('a * 'b) option

val scanString : ((char, cs) reader -> ('a, cs) reader) -> string -> 'a option

val splitl     : (char -> bool) -> (char, 'a) reader -> 'a -> string * 'a
val takel      : (char -> bool) -> (char, 'a) reader -> 'a -> string
val dropl      : (char -> bool) -> (char, 'a) reader -> 'a -> 'a
val skipWS     : (char, 'a) reader -> 'a -> 'a

val padLeft    : char -> int -> string -> string
val padRight   : char -> int -> string -> string
```

```
   This structure presents tools for scanning strings and values from
   functional character streams, and for simple formatting.

   [('elm, 'src) reader] is the type of source readers for reading a
   sequence of 'elm values from a source of type 'src.  For instance,
   a character source reader
        getc : (char, cs) reader
   is used for obtaining characters from a functional character source
   src of type cs, one at a time. It should hold that

        getc src = SOME(c, src')        if the next character in src
                                        is c, and src' is the rest of src;
                 = NONE                 if src contains no characters

   A character source scanner takes a character source reader getc as
   argument and uses it to scan a data value from the character
   source.

   [scanString scan s] turns the string s into a character source and
   applies the scanner `scan' to that source.

   [splitl p getc src] returns (pref, suff) where pref is the
   longest prefix (left substring) of src all of whose characters
   satisfy p, and suff is the remainder of src.  That is, the first
   character retrievable from suff, if any, is the leftmost character
   not satisfying p.  Does not skip leading whitespace.

   [takel p getc src] returns the longest prefix (left substring) of
   src all of whose characters satisfy predicate p.  That is, if the
   left-most character does not satisfy p, the result is the empty
   string.  Does not skip leading whitespace.  It holds that
        takel p getc src = #1 (splitl p getc src)

   [dropl p getc src] drops the longest prefix (left substring) of
   src all of whose characters satisfy predicate p.  If all characters
   do, it returns the empty source.  It holds that
        dropl p getc src = #2 (splitl p getc src)

   [skipWS getc src] drops any leading whitespace from src.
   Equivalent to dropl Char.isSpace.

   [padLeft c n s] returns the string s if size s >= n, otherwise pads
   s with (n - size s) copies of the character c on the left.
   In other words, right-justifies s in a field n characters wide.
```

```
   [padRight c n s] returns the string s if size s >= n, otherwise pads
   s with (n - size s) copies of the character c on the right.
   In other words, left-justifies s in a field n characters wide.
```

Module Substring

```
Substring -- SML Basis Library

type substring

val substring : string * int * int -> substring
val extract   : string * int * int option -> substring
val all       : string -> substring
val string    : substring -> string
val base      : substring -> (string * int * int)

val isEmpty   : substring -> bool
val getc      : substring -> (char * substring) option
val first     : substring -> char option
val triml     : int -> substring -> substring
val trimr     : int -> substring -> substring
val sub       : substring * int -> char
val size      : substring -> int
val slice     : substring * int * int option -> substring
val concat    : substring list -> string
val explode   : substring -> char list
val isPrefix  : string -> substring -> bool
val compare   : substring * substring -> order
val collate   : (char * char -> order) -> substring * substring -> order

val dropl     : (char -> bool) -> substring -> substring
val dropr     : (char -> bool) -> substring -> substring
val takel     : (char -> bool) -> substring -> substring
val taker     : (char -> bool) -> substring -> substring
val splitl    : (char -> bool) -> substring -> substring * substring
val splitr    : (char -> bool) -> substring -> substring * substring
val splitAt   : substring * int -> substring * substring

val position  : string -> substring -> substring * substring

exception Span
val span      : substring * substring -> substring

val translate : (char -> string) -> substring -> string

val tokens    : (char -> bool) -> substring -> substring list
val fields    : (char -> bool) -> substring -> substring list

val foldl     : (char * 'a -> 'a) -> 'a -> substring -> 'a
val foldr     : (char * 'a -> 'a) -> 'a -> substring -> 'a
val app       : (char -> unit) -> substring -> unit
```

```
   [substring] is the type of substrings of a basestring, an efficient
   representation of a piece of a string.
   A substring (s,i,n) is valid if 0 <= i <= i+n <= size s,
                 or equivalently, 0 <= i and 0 <= n and i+n <= size s.
   A valid substring (s, i, n) represents the string s[i...i+n-1].
   Invariant in the implementation: Any value of type substring is valid.

   [substring(s, i, n)] creates the substring (s, i, n), consisting of
   the substring of s with length n starting at i.  Raises Subscript
   if i<0 or n<0 or i+n > size s.  Equivalent to extract(s, i, SOME n).

   [extract(s, i, NONE)] creates the substring (s, i, size s-i)
   consisting of the tail of s starting at i.
   Raises Subscript if i<0 or i > size s.

   [extract(s, i, SOME n)] creates the substring (s, i, n),
   consisting of the substring of s with length n starting at i.
   Raises Subscript if i<0 or n<0 or i+n > size s.

   [all s] is the substring (s, 0, size s).

   [string sus] is the string s[i..i+n-1] represented by sus = (s, i, n).
```

```
[base sus] is the concrete triple (s, i, n), where sus = (s, i, n).

[isEmpty (s, i, n)] true if the substring is empty (that is, n = 0).

[getc sus] returns SOME(c, rst) where c is the first character and
rst the remainder of sus, if sus is non-empty; otherwise returns
NONE.  Note that
     #1 o valOf o scanFn Substring.getc
is equivalent to, but more efficient than,
     valOf o StringCvt.scanString scanFn o Substring.string

[first sus] returns SOME c where c is the first character in sus,
if sus is non-empty; otherwise returns NONE.

[triml k sus] returns sus less its leftmost k characters; or the
empty string at the end of sus if it has less than k characters.
Raises Subscript if k < 0, even in the partial application triml(k).

[trimr k sus] returns sus less its rightmost k characters; or the
empty string at the beginning of sus if it has less than k characters.
Raises Subscript if k < 0, even in the partial application triml(k).

[sub (sus, k)] returns the k'th character of the substring; that is,
s(i+k) where sus = (s, i, n).  Raises Subscript if k<0 or k>=n.

[size (s, i, n)] returns the size of the substring, that is, n.

[slice (sus, i', NONE)] returns the substring (s, i+i', n-i'), where
sus = (s, i, n).  Raises Subscript if i' < 0 or i' > n.

[slice (sus, i', SOME n')] returns the substring (s, i+i', n'), where
sus = (s, i, n).  Raises Subscript if i' < 0 or n' < 0 or i'+n' >= n.

[concat suss] returns a string consisting of the concatenation of
the substrings.  Equivalent to String.concat (List.map string suss).

[explode sus] returns the list of characters of sus, that is,
     [s(i), s(i+1), ..., s(i+n-1)]
where sus = (s, i, n).  Equivalent to String.explode(string ss).

[isPrefix s1 s2] is true if s1 is a prefix of s2. That is, if there
exists a string t such that string s1 ^ t = string s2.

[compare (sus1, sus2)] performs lexicographic comparison, using the
standard ordering Char.compare on the characters.  Returns LESS,
EQUAL, or GREATER, according as sus1 is less than, equal to, or
greater than sus2.  Equivalent to, but more efficient than,
    String.compare(string sus1, string sus2).

[collate cmp (sus1, sus2)] performs lexicographic comparison, using the
given ordering cmp on characters.  Equivalent to, but more efficient
than, String.collate cmp (string sus1, string sus2).

[dropl p sus] drops the longest prefix (left substring) of sus all
of whose characters satisfy predicate p.  If all characters do, it
returns the empty substring (s, i+n, 0) where sus = (s, i, n).

[dropr p sus] drops the longest suffix (right substring) of sus all
of whose characters satisfy predicate p.  If all characters do, it
returns the empty substring (s, i, 0) where sus = (s, i, n).

[takel p sus] returns the longest prefix (left substring) of sus
all of whose characters satisfy predicate p.  That is, if the
left-most character does not satisfy p, returns the empty (s, i, 0)
where sus = (s, i, n).

[taker p sus] returns the longest suffix (right substring) of sus
all of whose characters satisfy predicate p.  That is, if the
right-most character satisfies p, returns the empty (s, i+n, 0)
where sus = (s, i, n).

Let p be a predicate and xxxxfyyyyfzzzz a string where all
```

characters in xxxx and zzzz satisfy p, and f a is character
not satisfying p. Then

```
             sus = xxxxfyyyyfzzzz        sus = xxxxzzzz
     ------------------------------------------------------
     dropl p sus =     fyyyyfzzzz
     dropr p sus = xxxxfyyyyf
     takel p sus = xxxx                        xxxxzzzz
     taker p sus =          zzzz               xxxxzzzz
```

It also holds that

```
     concat[takel p sus, dropl p sus] = string sus
     concat[dropr p sus, taker p sus] = string sus
```

[splitl p sus] splits sus into a pair (sus1, sus2) of substrings
where sus1 is the longest prefix (left substring) all of whose
characters satisfy p, and sus2 is the rest. That is, sus2 begins
with the leftmost character not satisfying p. Disregarding
sideeffects, we have:

```
     splitl p sus = (takel p sus, dropl p sus).
```

[splitr p sus] splits sus into a pair (sus1, sus2) of substrings
where sus2 is the longest suffix (right substring) all of whose
characters satisfy p, and sus1 is the rest. That is, sus1 ends
with the rightmost character not satisfying p. Disregarding
sideeffects, we have:

```
     splitr p sus = (dropr p sus, taker p sus)
```

[splitAt (sus, k)] returns the pair (sus1, sus2) of substrings,
where sus1 contains the first k characters of sus, and sus2
contains the rest. Raises Subscript if k < 0 or k > size sus.

```
[position s (s’,i,n)] splits the substring into a pair (pref, suff)
of substrings, where suff is the longest suffix of (s’, i, n) which
has s as a prefix.  More precisely, let m = size s.  If there is a
least index k in i..i+n-m for which s = s’[k..k+m-1],
then the result is       pref = (s’, i, k-i) and suff = (s’, k, n-(k-i));
otherwise the result is  pref = (s’, i, n)   and suff = (s’, i+n, 0).
```

[span (sus1, sus2)] returns a substring spanning from the start of
sus1 to the end of sus2, provided this is well-defined: sus1 and
sus2 must have the same underlying string, and the start of sus1
must not be to the right of the end of sus2; otherwise raises Span.

More precisely, if base(sus1) = (s,i,n) and base(sus2) = (s’,i’,n’)
and s = s’ and i <= i’+n’, then base(join(sus1, sus2)) = (s, i, i’+n’-i).
This may be used to compute ‘span’, ‘union’, and ‘intersection’.

[translate f sus] applies f to every character of sus, from left to
right, and returns the concatenation of the results. Raises Size
if the sum of their sizes is greater than String.maxSize.
Equivalent to String.concat(List.map f (explode sus)).

[tokens p sus] returns the list of tokens in sus, from left to right,
where a token is a non-empty maximal substring of sus not containing
any delimiter, and a delimiter is a character satisfying p.

[fields p sus] returns the list of fields in sus, from left to right,
where a field is a (possibly empty) maximal substring of sus not
containing any delimiter, and a delimiter is a character satisfying p.

Two tokens may be separated by more than one delimiter, whereas two
fields are separated by exactly one delimiter. If the only delimiter
is the character #"|", then

```
     "abc||def" contains two tokens:   "abc" and "def"
     "abc||def" contains three fields: "abc" and "" and "def"
```

[foldl f e sus] folds f over sus from left to right. That is,
evaluates f(s[i+n-1], f(... f(s[i+1], f(s[i] % e)) ...))
tail-recursively, where sus = (s, i, n).
Equivalent to List.foldl f e (explode sus).

[foldr f e sus] folds f over sus from right to left. That is,
evaluates f(s[i], f(s[i+1], f(... f(s[i+n-1] % e) ...)))
tail-recursively, where sus = (s, i, n).
Equivalent to List.foldr f e (explode sus).

[app f sus] applies f to all characters of sus, from left to right.
Equivalent to List.app f (explode sus).

Module Susp

Susp -- support for lazy evaluation

```
type 'a susp

val delay : (unit -> 'a) -> 'a susp
val force : 'a susp -> 'a
```

```
   ['a susp] is the type of lazily evaluated expressions with result
   type 'a.

   [delay (fn () => e)] creates a suspension for the expression e.
   The first time the suspension is forced, the expression e will be
   evaluated, and the result stored in the suspension.  All subsequent
   forcing of the suspension will just return this result, so e is
   evaluated at most once.  If the suspension is never forced, then e
   is never evaluated.

   [force su] forces the suspension su and returns the result of the
   expression e stored in the suspension.
```

Module TextIO

TextIO -- SML Basis Library

```
type elem   = Char.char
type vector = string
```

Text input

```
type instream

val openIn       : string -> instream
val closeIn      : instream -> unit
val input        : instream -> vector
val inputAll     : instream -> vector
val inputNoBlock : instream -> vector option
val input1       : instream -> elem option
val inputN       : instream * int -> vector
val inputLine    : instream -> string
val endOfStream  : instream -> bool
val lookahead    : instream -> elem option
```

type cs *character source state*

```
val scanStream   : ((char, cs) StringCvt.reader -> ('a, cs) StringCvt.reader)
                   -> instream -> 'a option

val stdIn        : instream
```

Text output

```
type outstream

val openOut      : string -> outstream
val openAppend   : string -> outstream
val closeOut     : outstream -> unit
val output       : outstream * vector -> unit
val output1      : outstream * elem -> unit
val outputSubstr : outstream * substring -> unit
val flushOut     : outstream -> unit

val stdOut       : outstream
val stdErr       : outstream

val print        : string -> unit
```

```
   This structure provides input/output functions on text streams.
   The functions are state-based: reading from or writing to a stream
   changes the state of the stream.  The streams are buffered: output
   to a stream may not immediately affect the underlying file or
   device.

   Note that under DOS, Windows, OS/2, and MacOS, text streams will be
   'translated' by converting (e.g.) the double newline CRLF to a
   single newline character \n.

   [instream] is the type of state-based characters input streams.

   [outstream] is the type of state-based character output streams.

   [elem] is the type char of characters.

   [vector] is the type of character vectors, that is, strings.


   TEXT INPUT:

   [openIn s] creates a new instream associated with the file named s.
   Raises Io.Io is file s does not exist or is not accessible.

   [closeIn istr] closes stream istr.  Has no effect if istr is closed
```

already. Further operations on istr will behave as if istr is at end of stream (that is, will return "" or NONE or true).

[input istr] reads some elements from istr, returning a vector v of those elements. The vector will be empty (size v = 0) if and only if istr is at end of stream or is closed. May block (not return until data are available in the external world).

[inputAll istr] reads and returns the string v of all characters remaining in istr up to end of stream.

[inputNoBlock istr] returns SOME(v) if some elements v can be read without blocking; returns SOME("") if it can be determined without blocking that istr is at end of stream; returns NONE otherwise. If istr does not support non-blocking input, raises Io.NonblockingNotSupported.

[input1 istr] returns SOME(e) if at least one element e of istr is available; returns NONE if istr is at end of stream or is closed; blocks if necessary until one of these conditions holds.

[inputN(istr, n)] returns the next n characters from istr as a string, if that many are available; returns all remaining characters if end of stream is reached before n characters are available; blocks if necessary until one of these conditions holds. (This is the behaviour of the 'input' function prescribed in the 1990 Definition of Standard ML).

[inputLine istr] returns one line of text, including the terminating newline character. If end of stream is reached before a newline character, then the remaining part of the stream is returned, with a newline character added. If istr is at end of stream or is closed, then the empty string "" is returned.

[endOfStream istr] returns false if any elements are available in istr; returns true if istr is at end of stream or closed; blocks if necessary until one of these conditions holds.

[lookahead istr] returns SOME(e) where e is the next element in the stream; returns NONE if istr is at end of stream or is closed; blocks if necessary until one of these conditions holds. Does not advance the stream.

[stdIn] is the buffered state-based standard input stream.

[scanStream scan istr] turns the instream istr into a character source and applies the scanner 'scan' to that source. See StringCvt for more on character sources and scanners. The Moscow ML implementation currently can backtrack only 512 characters, and raises Fail if the scanner backtracks further than that.

TEXT OUTPUT:

[openOut s] creates a new outstream associated with the file named s. If file s does not exist, and the directory exists and is writable, then a new file is created. If file s exists, it is truncated (any existing contents are lost).

[openAppend s] creates a new outstream associated with the file named s. If file s does not exist, and the directory exists and is writable, then a new file is created. If file s exists, any existing contents are retained, and output goes at the end of the file.

[closeOut ostr] closes stream ostr; further operations on ostr (except for additional close operations) will raise exception Io.Io.

[output(ostr, v)] writes the string v on outstream ostr.

[output1(ostr, e)] writes the character e on outstream ostr.

[flushOut ostr] flushes the outstream ostr, so that all data written to ostr becomes available to the underlying file or device.

[stdOut] is the buffered state-based standard output stream.

[stdErr] is the unbuffered state-based standard error stream. That is, it is always kept flushed, so flushOut(stdErr) is redundant.

[print s] outputs s to stdOut and flushes immediately.

The functions below are not yet implemented:

[setPosIn(istr, i)] sets istr to the (untranslated) position i. Raises Io.Io if not supported on istr.

[getPosIn istr] returns the (untranslated) current position of istr. Raises Io.Io if not supported on istr.

[endPosIn istr] returns the (untranslated) last position of istr. Because of translation, one cannot expect to read
```
     endPosIn istr - getPosIn istr
```
from the current position.

[getPosOut ostr] returns the current position in stream ostr. Raises Io.Io if not supported on ostr.

[endPosOut ostr] returns the ending position in stream ostr. Raises Io.Io if not supported on ostr.

[setPosOut(ostr, i)] sets the current position in stream to ostr to i. Raises Io.Io if not supported on ostr.

[mkInstream sistr] creates a state-based instream from the functional instream sistr.

[getInstream istr] returns the functional instream underlying the state-based instream istr.

[setInstream(istr, sistr)] redirects istr, so that subsequent input is taken from the functional instream sistr.

[mkOutstream sostr] creates a state-based outstream from the outstream sostr.

[getOutstream ostr] returns the outstream underlying the state-based outstream ostr.

[setOutstream(ostr, sostr)] redirects the outstream ostr so that subsequent output goes to sostr.

Module Time

Time -- SML Basis Library

```
eqtype time

exception Time

val zeroTime         : time
val now              : unit -> time

val toSeconds        : time -> int
val toMilliseconds   : time -> int
val toMicroseconds   : time -> int
val fromSeconds      : int -> time
val fromMilliseconds : int -> time
val fromMicroseconds : int -> time

val fromReal         : real -> time
val toReal           : time -> real

val toString         : time -> string rounded to millisecond precision
val fmt              : int -> time -> string
val fromString       : string -> time option
val scan             : (char, 'a) StringCvt.reader
                       -> (time, 'a) StringCvt.reader

val +       : time * time -> time
val -       : time * time -> time
val <       : time * time -> bool
val <=      : time * time -> bool
val >       : time * time -> bool
val >=      : time * time -> bool

val compare : time * time -> order
```

```
   [time] is a type for representing durations as well as absolute
   points in time (which can be thought of as durations since some
   fixed time zero).

   [zeroTime] represents the 0-second duration, and the origin of time,
   so zeroTime + t = t + zeroTime = t for all t.

   [now ()] returns the point in time at which the application occurs.

   [fromSeconds s] returns the time value corresponding to s seconds.
   Raises Time if s < 0.

   [fromMilliseconds ms] returns the time value corresponding to ms
   milliseconds.  Raises Time if ms < 0.

   [fromMicroseconds us] returns the time value corresponding to us
   microseconds.  Raises Time if us < 0.

   [toSeconds t] returns the number of seconds represented by t,
   truncated.  Raises Overflow if that number is not representable as
   an int.

   [toMilliseconds t] returns the number of milliseconds
   represented by t, truncated.  Raises Overflow if that number is not
   representable as an int.

   [toMicroseconds t] returns the number of microseconds
   represented by t, truncated.  Raises Overflow if t that number is
   not representable as an int.

   [fromReal r] converts a real to a time value representing that
   many seconds.  Raises Time if r < 0 or if r is not representable
   as a time value.  It holds that realToTime 0.0 = zeroTime.

   [toReal t] converts a time the number of seconds it represents;
```

```
hence realToTime and timeToReal are inverses of each other when
defined.  Raises Overflow if t is not representable as a real.

[fmt n t] returns as a string the number of seconds represented by
t, rounded to n decimal digits.  If n <= 0, then no decimal digits
are reported.

[toString t] returns as a string the number of seconds represented
by t, rounded to 3 decimal digits.  Equivalent to (fmt 3 t).

[fromString s] returns SOME t where t is the time value represented
by the string s of form [\n\t ]*([0-9]+(\.[0-9]+)?)|(\.[0-9]+);
or returns NONE if s cannot be parsed as a time value.

[scan getc src], where getc is a character accessor, returns SOME
(t, rest) where t is a time and rest is rest of the input, or NONE
if s cannot be parsed as a time value.

[+] adds two time values. For reals r1, r2 >= 0.0, it holds that
realToTime r1 + realToTime r2 = realToTime(Real.+(r1,r2)).
Raises Overflow if the result is not representable as a time value.

[-] subtracts a time value from another.  That is, t1 - t2 is the
duration from t2 to t1.  Raises Time if t1 < t2 or if the result is
not representable as a time value.  It holds that t - zeroTime = t.

[<]
[<=]
[>]
[>=] compares time values.  For instance, for reals r1, r2 >= 0.0
it holds that realToTime r1 < realToTime r2 iff Real.<(r1, r2)

[compare(t1, t2)] returns LESS, EQUAL, or GREATER, according
as t1 precedes, equals, or follows t2 in time.
```

Module Timer

Timer -- SML Basis Library

```
type cpu_timer
type real_timer

val startCPUTimer  : unit -> cpu_timer
val totalCPUTimer  : unit -> cpu_timer
val checkCPUTimer  : cpu_timer ->
                     { usr : Time.time, sys : Time.time, gc : Time.time }

val startRealTimer : unit -> real_timer
val totalRealTimer : unit -> real_timer
val checkRealTimer : real_timer -> Time.time
```

[cpu_timer] is the type of timers for measuring CPU time consumption (user time, garbage collection time, and system time).

[real_timer] is the type of timers for measuring the passing of real time (wall-clock time).

[startCPUTimer ()] returns a cpu_timer started at the moment of the call.

[totalCPUTimer ()] returns a cpu_timer started at the moment the library was loaded.

[checkCPUTimer tmr] returns {usr, sys, gc} where usr is the amount of user CPU time consumed since tmr was started, gc is the amount of user CPU time spent on garbage collection, and sys is the amount of system CPU time consumed since tmr was started. Note that gc time is included in the usr time. Under MS DOS, usr time and gc time are measured in real time.

[startRealTimer ()] returns a real_timer started at the moment of the call.

[totalRealTimer ()] returns a real_timer started at the moment the library was loaded.

[checkRealTimer tmr] returns the amount of real time that has passed since tmr was started.

Module Unix

Unix -- SML Basis Library

```
type proc
type signal = Signal.signal

val executeInEnv : string * string list * string list -> proc
val execute      : string * string list -> proc
val streamsOf    : proc -> TextIO.instream * TextIO.outstream
val kill         : proc * signal -> unit
val reap         : proc -> Process.status
```

This structure allows Moscow ML programs to start other processes and to communicate with them.

Child processes are not automatically terminated when the parent (ML) process terminates. To forcibly terminate a child process pr, use Unix.kill(pr, Signal.term). Then, to remove the terminated process from the operating system tables, call Unix.reap(pr).

The protocol for communication between the ML program and its child process must be designed with some care, typically using non-blocking input for reading from the child process.

[proc] is the type of processes started by the ML program.

[signal] is the type of Unix-style signals, which can be sent to another process. Signal values must be obtained from the Signal structure.

[execute (cmd, args)] asks the operating system to execute the command cmd with the argument list args, as a separate process. Two pipes connected to the standard input and standard output of the new process are created; these may be obtained using streamsOf. A proc value representing the new process is returned. The new process executes using the same environment as the calling process. Raises Fail in case of failure, e.g. if the process or the pipes cannot be created.

Typically, the cmd argument will be the full pathname of an executable. On Unix systems, simple command searching as done by the shell, allowing cmd to be a relative pathname, can be achieved by using

```
     execute("/bin/sh", "-c" :: concat (cmd :: " " :: args))
```

[executeInEnv (cmd, args, env)] asks the operating system to execute the command cmd with the argument list args in the environment env, as a separate process. Returns a proc value representing the new process. Typically, a string in the env list has the form "NAME=VALUE". See also Process.getEnv.

[streamsOf pr] returns a pair (ins, outs) of input and output streams associated with process pr. The standard output of pr is the source for the input stream ins, and the standard input of pr is the sink for the output stream outs.

[reap pr] closes the input and output streams associated with pr, and then suspends the current (ML) process until the process corresponding to pr terminates. Returns the exit status given by pr when it terminated. Raises Fail in case of failure, e.g. if pr has already been reaped.

Under Unix, information about a terminated process remains in the system tables until the process is reaped. Thus, an ML program using execute or executeInEnv must make sure to reap any process it has created, or else the system tables will fill up.

[kill (pr, s)] sends the signal s to the process pr. Raises Fail in case of failure, e.g. if pr has already been killed.

Module Vector

Vector -- SML Basis Library

```
type 'a vector = 'a vector
val maxLen   : int

val fromList : 'a list -> 'a vector
val tabulate : int * (int -> 'a) -> 'a vector

val length   : 'a vector -> int
val sub      : 'a vector * int -> 'a
val extract  : 'a vector * int * int option -> 'a vector
val concat   : 'a vector list -> 'a vector

val app      : ('a -> unit) -> 'a vector -> unit
val map      : ('a -> 'b) -> 'a vector -> 'b vector
val foldl    : ('a * 'b -> 'b) -> 'b -> 'a vector -> 'b
val foldr    : ('a * 'b -> 'b) -> 'b -> 'a vector -> 'b

val appi     : (int * 'a -> unit) -> 'a vector * int * int option -> unit
val mapi     : (int * 'a -> 'b) -> 'a vector * int * int option -> 'b vector
val foldli   : (int * 'a * 'b -> 'b) -> 'b -> 'a vector*int*int option -> 'b
val foldri   : (int * 'a * 'b -> 'b) -> 'b -> 'a vector*int*int option -> 'b
```

```
   ['ty vector] is the type of one-dimensional, immutable, zero-based
   constant-time-access vectors with elements of type 'ty.
   Type 'ty vector admits equality if 'ty does.  Vectors v1 and v2 are
   equal if they have the same length and their elements are equal.

   [maxLen] is the maximal number of elements in a vector.

   [fromList xs] returns a vector whose elements are those of xs.
   Raises Size if length xs > maxLen.

   [tabulate(n, f)] returns a vector of length n whose elements
   are f 0, f 1, ..., f (n-1), created from left to right.  Raises
   Size if n<0 or n>maxLen.

   [length v] returns the number of elements in v.

   [sub(v, i)] returns the i'th element of v, counting from 0.
   Raises Subscript if i<0 or i>=length v.

   [extract(v, i, NONE)] returns a vector of the elements v[i..length v-1]
   of v.  Raises Subscript if i<0 or i>length v.

   [extract(v, i, SOME n)] returns a vector of the elements v[i..i+n-1]
   of v.  Raises Subscript if i<0 or n<0 or i+n>length v.

   [concat vs] returns a vector which is the concatenation from left
   to right og the vectors in vs.  Raises Size if the sum of the
   sizes of the vectors in vs is larger than maxLen.

   [foldl f e v] folds function f over v from left to right.  That is,
   computes f(v[len-1], f(v[len-2], ..., f(v[1], f(v[0], e)) ...)),
   where len is the length of v.

   [foldr f e v] folds function f over v from right to left.  That is,
   computes f(v[0], f(v[1], ..., f(v[len-2], f(v[len-1], e)) ...)),
   where len is the length of v.

   [app f v] applies f to v[j] for j=0,1,...,length v-1.

   [map f v] applies f to v[j] for j=0,1,...,length v-1 and returns a
   new vector containing the results.


   The following iterators generalize the above ones in two ways:

    * the index j is also being passed to the function being iterated;
```

```
    * the iterators work on a slice (subvector) of a vector.

   The slice (v, i, SOME n) denotes the subvector v[i..i+n-1].  That is,
   v[i] is the first element of the slice, and n is the length of the
   slice.  Valid only if 0 <= i <= i+n <= length v.

   The slice (v, i, NONE) denotes the subvector v[i..length v-1].  That
   is, the slice denotes the suffix of the vector starting at i.  Valid
   only if 0 <= i <= length v.  Equivalent to (v, i, SOME(length v - i)).

       slice            meaning
       ----------------------------------------------------------
       (v, 0, NONE)     the whole vector             v[0..len-1]
       (v, 0, SOME n)   a left subvector (prefix)    v[0..n-1]
       (v, i, NONE)     a right subvector (suffix)   v[i..len-1]
       (v, i, SOME n)   a general slice              v[i..i+n-1]

   [foldli f e (v, i, SOME n)] folds function f over the subvector
   v[i..i+n-1] from left to right.  That is, computes
   f(i+n-1, v[i+n-1], f(..., f(i+1, v[i+1], f(i, v[i], e)) ...)).
   Raises Subscript if i<0 or n<0 or i+n > length v.

   [foldli f e (v, i, NONE)] folds function f over the subvector
   v[i..len-1] from left to right, where len =  length v.  That is,
   computes f(len-1, v[len-1], f(..., f(i+1, v[i+1], f(i, v[i], e)) ...)).
   Raises Subscript if i<0 or i > length v.

   [foldri f e (v, i, SOME n)] folds function f over the subvector
   v[i..i+n-1] from right to left.  That is, computes
   f(i, v[i], f(i+1, v[i+1], ..., f(i+n-1, v[i+n-1], e) ...)).
   Raises Subscript if i<0 or n<0 or i+n > length v.

   [foldri f e (v, i, NONE)] folds function f over the subvector
   v[i..len-1] from right to left, where len = length v.  That is,
   computes f(i, v[i], f(i+1, v[i+1], ..., f(len-1, v[len-1], e) ...)).
   Raises Subscript if i<0 or i > length v.

   [appi f (v, i, SOME n)] applies f to successive pairs (j, v[j]) for
   j=i,i+1,...,i+n-1.  Raises Subscript if i<0 or n<0 or i+n > length v.

   [appi f (v, i, NONE)] applies f to successive pairs (j, v[j]) for
   j=i,i+1,...,len-1, where len = length v.  Raises Subscript if i<0
   or i > length v.

   [mapi f (v, i, SOME n)] applies f to successive pairs (j, v[j]) for
   j=i,i+1,...,i+n-1 and returns a new vector (of length n) containing
   the results.  Raises Subscript if i<0 or n<0 or i+n > length v.

   [mapi f (v, i, NONE)] applies f to successive pairs (j, v[j]) for
   j=i,i+1,...,len-1, where len = length v, and returns a new vector
   (of length len-i) containing the results.  Raises Subscript if i<0
   or i > length v.
```

Module Weak

Weak --- weak pointers and arrays of weak pointers

Single weak pointers

```
type 'a weak
val weak    : 'a -> 'a weak
val set     : 'a weak * 'a -> unit
val get     : 'a weak -> 'a                Raises Fail
val isweak  : 'a weak -> bool
```

Arrays of weak pointers

```
prim_EQtype 'a array

val maxLen  : int

val array   : int -> '_a array             Raises Size
val sub     : 'a array * int -> 'a         Raises Fail and Subscript
val update  : 'a array * int * 'a -> unit   Raises Subscript
val isdead  : 'a array * int -> bool       Raises Subscript
val length  : 'a array -> int

val app     : ('a -> unit) -> 'a array -> unit
val foldl   : ('a * 'b -> 'b) -> 'b -> 'a array -> 'b
val foldr   : ('a * 'b -> 'b) -> 'b -> 'a array -> 'b
val modify  : ('a -> 'a) -> 'a array -> unit

val appi    : (int * 'a -> unit) -> 'a array * int * int option -> unit
val foldli  : (int * 'a * 'b -> 'b) -> 'b -> 'a array * int * int option
              -> 'b
val foldri  : (int * 'a * 'b -> 'b) -> 'b -> 'a array * int * int option
              -> 'b
val modifyi : (int * 'a -> 'a) -> 'a array * int * int option -> unit
```

['a weak] is the type of weak pointers to objects of type 'a. A weak pointer is a pointer that cannot itself keep an object alive. Hence the object pointed to by a weak pointer may be deallocated by the garbage collector if the object is reachable only by weak pointers. In this case, subsequent accesses via the 'get' function will raise Fail "Dangling weak pointer". (We raise an exception instead of returning an option value, because access via a weak pointer to a deallocated object is likely to be a programming error).

Integers, characters, words and booleans will not be deallocated by the garbage collector and will remain reachable forever by a weak pointer. Reals, strings, tuples and other non-nullary constructors may be deallocated by the garbage collector. Constants, even composite ones, will not be deallocated either.

[weak v] creates and returns a weak pointer to value v.

[get w] returns the value pointed to by weak pointer w, if the value is still alive. Otherwise raises Fail "Dangling weak pointer".

[set(w, v)] makes the weak pointer w point to the value v.

[isweak w] returns true if the value pointed to by w is dead; returns false otherwise. If an object is reported to be dead, it remains dead. However, an object is reported to be live just if it has not yet been deallocated by the garbage collector. The allocation of any new value may activate the garbage collector and cause the object to die. Thus
```
     if not (isweak w) then get w else "blah"
```
will not raise exception Fail, whereas the following might:
```
     if not (isweak w) then ([1,2] @ [3,4]; get w) else "blah"
```
because evaluation of the list append may cause w to die.

The value of isweak w is the same as that of

```
      (get w; false) handle Fail _ => true
```
but evaluating the latter expression may have the side effect of keeping w alive for slightly longer, because a pointer to w is returned by get w.

['a array] is the type of arrays of weak pointers to objects of type 'a.

A value of type 'a Weak.weak (above) is equivalent to, but more efficient than, a one-element 'a Weak.array. On the other hand, an 'a Weak.array is more efficient than an ('a Weak.weak) Array.array.

[array n] creates an array of n weak pointers. Initially, any access to the array will raise Fail.

[sub(a, i)] returns the object pointed to by cell i (counting from 0) of the array a, if it is live. Raises Fail "Dangling weak pointer" if cell i has never been updated or if the object pointed to has been deallocated by the garbage collector. Raises Subscript if i<0 or i>=length a. To make 'sub' infix, use the declaration
```
                          infix 9 sub
```

[update(a, i, v)] updates cell i of array a to point (weakly) to the value v. Raises Subscript if i<0 or i>=length a.

[isdead(a, i)] returns true if the object in cell i of array a is dead, and false otherwise. Analogous to isweak; see above.

[length a] returns the number of elements in a.

[maxLen] is the maximal number of elements in an array.

The iterators described below operate on the live elements only. Note that an element a[k] may die in the course of folding f over earlier elements (e.g. a[1] ... a[k-1]). Thus the functions should be used with great care.

[foldl f e a] folds function f over the live elements of a, from left to right.

[foldr f e a] folds function f over the live elements of a, from right to left.

[app f a] applies f to the live elements of a from left to right.

[modify f a] applies f to a[j] and updates a[j] with the result f(a[j]), for each live element a[j], from left to right.

The following iterators generalize the above ones in two ways:

 . the index j is also being passed to the function being iterated;
 . the iterators work on a slice (subarray) of an array.

The slice (a, i, SOME n) denotes the subarray a[i..i+n-1]. That is, a[i] is the first element of the slice, and n is the length of the slice. Valid only if 0 <= i <= i+n <= length a.

The slice (a, i, NONE) denotes the subarray a[i..length a-1]. That is, the slice denotes the suffix of the array starting at i. Valid only if 0 <= i <= length a. Equivalent to (a, i, SOME(length a - i)).

slice	meaning	
(a, 0, NONE)	the whole array	a[0..len-1]
(a, 0, SOME n)	a left subarray (prefix)	a[0..n-1]
(a, i, NONE)	a right subarray (suffix)	a[i..len-1]
(a, i, SOME n)	a general slice	a[i..i+n-1]

[foldli f e (a, i, SOME n)] folds function f over the live elements of the subarray a[i..i+n-1] from left to right. Raises Subscript

```
if i<0 or n<0 or i+n > length a.

[foldli f e (a, i, NONE)] folds function f over the live elements
of the subarray a[i..len-1] from left to right, where len = length
a.  Raises Subscript if i<0 or i > length a.

[foldri f e (a, i, SOME n)] folds function f over the live elements
of the subarray a[i..i+n-1] from right to left.  Raises Subscript
if i<0 or n<0 or i+n > length a.

[foldri f e (a, i, NONE)] folds function f over the live elements
of the subarray a[i..len-1] from right to left, where len = length
a.  Raises Subscript if i<0 or i > length a.

[appi f (a, i, SOME n)] applies f to successive pairs (j, a[j]) for
j=i,i+1,...,i+n-1, provided a[j] is live.  Raises Subscript if i<0
or n<0 or i+n > length a.

[appi f (a, i, NONE)] applies f to successive pairs (j, a[j]) for
j=i,i+1,...,len-1, where len = length a, provided a[j] is live.
Raises Subscript if i<0 or i > length a.

[modifyi f (a, i, SOME n)] applies f to (j, a[j]) and updates a[j]
with the result f(j, a[j]) for j=i,i+1,...,i+n-1, provided a[j] is
live.  Raises Subscript if i<0 or n<0 or i+n > length a.

[modifyi f (a, i, NONE)] applies f to (j, a[j]) and updates a[j]
with the result f(j, a[j]) for j=i,i+1,...,len-1, provided a[j] is
live.  Raises Subscript if i<0 or i > length a.
```

Module Word

Word -- SML Basis Library

```
type word = word

val wordSize   : int

val orb        : word * word -> word
val andb       : word * word -> word
val xorb       : word * word -> word
val notb       : word -> word

val <<         : word * word -> word
val >>         : word * word -> word
val ~>>        : word * word -> word

val +          : word * word -> word
val -          : word * word -> word
val *          : word * word -> word
val div        : word * word -> word
val mod        : word * word -> word

val >          : word * word -> bool
val <          : word * word -> bool
val >=         : word * word -> bool
val <=         : word * word -> bool
val compare    : word * word -> order

val min        : word * word -> word
val max        : word * word -> word

val toString   : word -> string
val fromString : string -> word option
val scan       : StringCvt.radix
                 -> (char, 'a) StringCvt.reader -> (word, 'a) StringCvt.reader
val fmt        : StringCvt.radix -> word -> string

val toInt      : word -> int
val toIntX     : word -> int           with sign extension
val fromInt    : int -> word

val toLargeWord   : word -> word
val toLargeWordX  : word -> word       with sign extension
val fromLargeWord : word -> word

val toLargeInt    : word -> int
val toLargeIntX   : word -> int        with sign extension
val fromLargeInt  : int -> word
```

```
[word] is the type of n-bit words, or n-bit unsigned integers.

[wordSize] is the value of n above.  In Moscow ML, n=31 on 32-bit
machines and n=63 on 64-bit machines.

[orb(w1, w2)] returns the bitwise ‘or’ of w1 and w2.

[andb(w1, w2)] returns the bitwise ‘and’ of w1 and w2.

[xorb(w1, w2)] returns the bitwise ‘exclusive or’ or w1 and w2.

[notb w] returns the bitwise negation of w.

[<<(w, k)] returns the word resulting from shifting w left by k
bits.  The bits shifted in are zero, so this is a logical shift.
Consequently, the result is 0-bits when k >= wordSize.

[>>(w, k)] returns the word resulting from shifting w right by k
bits.  The bits shifted in are zero, so this is a logical shift.
Consequently, the result is 0-bits when k >= wordSize.
```

[~>>(w, k)] returns the word resulting from shifting w right by k bits. The bits shifted in are replications of the left-most bit: the 'sign bit', so this is an arithmetical shift. Consequently, for k >= wordSize and wordToInt w >= 0 the result is all 0-bits, and for k >= wordSize and wordToInt w < 0 the result is all 1-bits.

To make <<, >>, and ~>> infix, use the declaration

```
                      infix 5 << >> ~>>
```

[+]
[-]
[*]
[div]
[mod] represent unsigned integer addition, subtraction, multiplication, division, and remainder, modulus 2 raised to the n'th power, where n=wordSize. The operations (i div j) and (i mod j) raise Div when j=0. Otherwise no exceptions are raised.

[<]
[<=]
[>]
[>=] compare words as unsigned integers.

[compare(w1, w2)] returns LESS, EQUAL, or GREATER, according as w1 is less than, equal to, or greater than w2 (as unsigned integers).

[min(w1, w2)] returns the smaller of w1 and w2 (as unsigned integers).

[max(w1, w2)] returns the larger of w1 and w2 (as unsigned integers).

[fmt radix w] returns a string representing w, in the radix (base) specified by radix.

radix	description		output format
BIN	unsigned binary	(base 2)	[01]+
OCT	unsigned octal	(base 8)	[0-7]+
DEC	unsigned decimal	(base 10)	[0-9]+
HEX	unsigned hexadecimal	(base 16)	[0-9A-F]+

[toString w] returns a string representing w in unsigned hexadecimal format. Equivalent to (fmt HEX w).

[fromString s] returns SOME(w) if a hexadecimal unsigned numeral can be scanned from a prefix of string s, ignoring any initial whitespace; returns NONE otherwise. Raises Overflow if the scanned number cannot be represented as a word. An unsigned hexadecimal numeral must have form, after possible initial whitespace:

```
     [0-9a-fA-F]+
```

[scan radix getc charsrc] attempts to scan an unsigned numeral from the character source charsrc, using the accessor getc, and ignoring any initial whitespace. The radix argument specifies the base of the numeral (BIN, OCT, DEC, HEX). If successful, it returns SOME(w, rest) where w is the value of the numeral scanned, and rest is the unused part of the character source. Raises Overflow if the scanned number cannot be represented as a word. A numeral must have form, after possible initial whitespace:

radix	input format
BIN	(0w)?[0-1]+
OCT	(0w)?[0-7]+
DEC	(0w)?[0-9]+
HEX	(0wx|0wX|0x|0X)?[0-9a-fA-F]+

[toInt w] returns the (signed) integer represented by bit-pattern w.
[toIntX w] returns the (signed) integer represented by bit-pattern w.
[fromInt i] returns the word representing integer i.

[toLargeInt w] returns the (signed) integer represented by bit-pattern w.
[toLargeIntX w] returns the (signed) integer represented by bit-pattern w.

[fromLargeInt i] returns the word representing integer i.

[toLargeWord w] returns w.
[toLargeWordX w] returns w.
[fromLargeWord w] returns w.

Module Word8

Word8 -- SML Basis Library

```
type word = word8

val wordSize   : int

val orb        : word * word -> word
val andb       : word * word -> word
val xorb       : word * word -> word
val notb       : word -> word

val <<         : word * Word.word -> word
val >>         : word * Word.word -> word
val ~>>        : word * Word.word -> word

val +          : word * word -> word
val -          : word * word -> word
val *          : word * word -> word
val div        : word * word -> word
val mod        : word * word -> word

val >          : word * word -> bool
val <          : word * word -> bool
val >=         : word * word -> bool
val <=         : word * word -> bool
val compare    : word * word -> order

val min        : word * word -> word
val max        : word * word -> word

val toString   : word -> string
val fromString : string -> word option
val scan       : StringCvt.radix
               -> (char, 'a) StringCvt.reader -> (word, 'a) StringCvt.reader
val fmt        : StringCvt.radix -> word -> string

val toInt      : word -> int
val toIntX     : word -> int           with sign extension
val fromInt    : int -> word

val toLargeInt    : word -> int
val toLargeIntX   : word -> int        with sign extension
val fromLargeInt  : int -> word

val toLargeWord   : word -> Word.word
val toLargeWordX  : word -> Word.word  with sign extension
val fromLargeWord : Word.word -> word
```

[word] is the type of 8-bit words, or 8-bit unsigned integers in the range 0..255.

[wordSize] equals 8.

[orb(w1, w2)] returns the bitwise 'or' of w1 and w2.

[andb(w1, w2)] returns the bitwise 'and' of w1 and w2.

[xorb(w1, w2)] returns the bitwise 'exclusive or' or w1 and w2.

[notb w] returns the bitwise negation of w.

[<<(w, k)] returns the word resulting from shifting w left by k bits. The bits shifted in are zero, so this is a logical shift. Consequently, the result is 0-bits when k >= wordSize.

[>>(w, k)] returns the word resulting from shifting w right by k bits. The bits shifted in are zero, so this is a logical shift. Consequently, the result is 0-bits when k >= wordSize.

[~>>(w, k)] returns the word resulting from shifting w right by k bits. The bits shifted in are replications of the left-most bit: the 'sign bit', so this is an arithmetical shift. Consequently, for k >= wordSize and wordToInt w >= 0 the result is all 0-bits, and for k >= wordSize and wordToInt w < 0 the result is all 1-bits.

To make <<, >>, and ~>> infix, use the declaration:

```
                       infix 5 << >> ~>>
```

[+]
[-]
[*]
[div]
[mod] represent unsigned integer addition, subtraction, multiplication, division, and remainder, modulus 256. The operations (i div j) and (i mod j) raise Div when j = 0. Otherwise no exceptions are raised.

[<]
[<=]
[>]
[>=] compare words as unsigned integers.

[compare(w1, w2)] returns LESS, EQUAL, or GREATER, according as w1 is less than, equal to, or greater than w2 (as unsigned integers).

[min(w1, w2)] returns the smaller of w1 and w2 (as unsigned integers).

[max(w1, w2)] returns the larger of w1 and w2 (as unsigned integers).

[fmt radix w] returns a string representing w, in the radix (base) specified by radix.

```
  radix    description                     output format
  ------------------------------------------------------
   BIN     unsigned binary      (base  2)  [01]+
   OCT     unsigned octal       (base  8)  [0-7]+
   DEC     unsigned decimal     (base 10)  [0-9]+
   HEX     unsigned hexadecimal (base 16)  [0-9A-F]+
```

[toString w] returns a string representing w in unsigned hexadecimal format. Equivalent to (fmt HEX w).

[fromString s] returns SOME(w) if a hexadecimal unsigned numeral can be scanned from a prefix of string s, ignoring any initial whitespace; returns NONE otherwise. Raises Overflow if the scanned number cannot be represented as a word. An unsigned hexadecimal numeral must have form, after possible initial whitespace:

```
    [0-9a-fA-F]+
```

[scan radix {getc} charsrc] attempts to scan an unsigned numeral from the character source charsrc, using the accessor getc, and ignoring any initial whitespace. The radix argument specifies the base of the numeral (BIN, OCT, DEC, HEX). If successful, it returns SOME(w, rest) where w is the value of the numeral scanned, and rest is the unused part of the character source. Raises Overflow if the scanned number cannot be represented as a word. A numeral must have form, after possible initial whitespace:

```
  radix    input format
  -------------------------------------
   BIN     (0w)?[0-1]+
   OCT     (0w)?[0-7]+
   DEC     (0w)?[0-9]+
   HEX     (0wx|0wX|0x|0X)?[0-9a-fA-F]+
```

[toInt w] returns the integer in the range 0..255 represented by w.

[toIntX w] returns the signed integer (in the range ~128..127) represented by bit-pattern w.

[fromInt i] returns the word holding the 8 least significant bits of i.

```
[toLargeInt w] returns the integer in the range 0..255 represented by w.

[toLargeIntX w] returns the signed integer (in the range ~128..127)
represented by bit-pattern w.

[fromLargeInt i] returns the word holding the 8 least significant bits of i.

[toLargeWord w] returns the Word.word value corresponding to w.

[toLargeWordX w] returns the Word.word value corresponding to w,
with sign extension.  That is, the 8 least significant bits of the
result are those of w, and the remaining bits are all equal to the
most significant bit of w: its ‘sign bit’.

[fromLargeWord w] returns w modulo 256.
```

Module Word8Array

Word8Array -- SML Basis Library

```
eqtype array
type elem   = Word8.word
type vector = Word8Vector.vector

val maxLen   : int

val array    : int * elem -> array
val tabulate : int * (int -> elem) -> array
val fromList : elem list -> array

val length   : array -> int
val sub      : array * int -> elem
val update   : array * int * elem -> unit
val extract  : array * int * int option -> vector

val copy     : {src: array,  si: int, len: int option,
                dst: array, di: int} -> unit
val copyVec  : {src: vector, si: int, len: int option,
                dst: array, di: int} -> unit

val app      : (elem -> unit) -> array -> unit
val foldl    : (elem * 'b -> 'b) -> 'b -> array -> 'b
val foldr    : (elem * 'b -> 'b) -> 'b -> array -> 'b
val modify   : (elem -> elem) -> array -> unit

val appi     : (int * elem -> unit) -> array * int * int option -> unit
val foldli   : (int * elem * 'b -> 'b) -> 'b -> array * int * int option -> 'b
val foldri   : (int * elem * 'b -> 'b) -> 'b -> array * int * int option -> 'b
val modifyi  : (int * elem -> elem) -> array * int * int option -> unit
```

```
   [array] is the type of one-dimensional, mutable, zero-based
   constant-time-access arrays with elements of type Word8.word, that
   is, 8-bit words.  Arrays a1 and a2 are equal if both were created
   by the same call to a primitive (array0, array, tabulate, fromList).

   All operations are as for Array.array.
```

Module Word8Vector

Word8Vector -- SML Basis Library

```
eqtype vector
type elem = Word8.word

val maxLen   : int

val fromList : elem list -> vector
val tabulate : int * (int -> elem) -> vector

val length   : vector -> int
val sub      : vector * int -> elem
val extract  : vector * int * int option -> vector
val concat   : vector list -> vector

val app      : (elem -> unit) -> vector -> unit
val map      : (elem -> elem) -> vector -> vector
val foldl    : (elem * 'b -> 'b) -> 'b -> vector -> 'b
val foldr    : (elem * 'b -> 'b) -> 'b -> vector -> 'b

val appi     : (int * elem -> unit) -> vector * int * int option -> unit
val mapi     : (int * elem -> elem) -> vector * int * int option -> vector
val foldli   : (int * elem * 'b -> 'b) -> 'b -> vector*int*int option -> 'b
val foldri   : (int * elem * 'b -> 'b) -> 'b -> vector*int*int option -> 'b
```

```
   [vector] is the type of one-dimensional, immutable, zero-based
   constant-time-access vectors with elements of type Word8.word, that
   is, 8-bit words.  Type vector admits equality, and vectors v1 and
   v2 are equal if they have the same length and their elements are
   equal.

   All operations are as for Vector.vector.
```

Index