

Moscow ML Language Overview

Version 1.43 of April 1998

Sergei Romanenko
Russian Academy of Sciences
Moscow, Russia

Peter Sestoft
Royal Veterinary and Agricultural University
Copenhagen, Denmark

This is a compact reference to the language implemented by Moscow ML, a subset of Standard ML. For reference material on Standard ML, see Milner, Tofte, Harper and MacQueen: *The Definition of Standard ML*, The MIT Press 1997. Moscow ML implements parts of the SML Basis Library. Section 14 of this manual lists all structure, type, constructor, value, function, and exception identifiers defined by Moscow ML.

Acknowledgement: The present document owes a lot to the *Definition*.

Contents

1	Reserved words	2
2	Comments	2
3	Special constants	2
4	Identifiers	3
5	Infix operators	3
6	Notational conventions used in the grammar	3
7	Grammar for the Standard ML Core language	4
8	Interactive sessions	6
9	Grammar for the Moscow ML module language	7
10	Built-in types, constructors and exceptions	8
11	Built-in variables and functions	9
12	List of all library units	11
13	The preloaded library units	13
14	Moscow ML library index	17

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

1 Reserved words

```
abstype and andalso as case do datatype else
end exception fn fun handle if in infix infixr let
local nonfix of op open orelse raise rec sig signature
struct structure then type val with withtype while
( ) [ ] { } , : :> ; ... _ | = => -> #
```

2 Comments

A comment is any character sequence within comment brackets (`*` and `*`) in which comment brackets are properly nested.

3 Special constants

Integer constants

Examples:	0	<code>~0</code>	4	<code>~04</code>	999999	<code>0xFFFF</code>	<code>~0x1ff</code>
Non-examples:	0.0	<code>~0.0</code>	4.0	<code>1E0</code>	-317	<code>0XFFFF</code>	<code>-0x1ff</code>

Real constants

Examples:	0.7	<code>~0.7</code>	3.32E5	<code>3E~7</code>	<code>~3E~7</code>	<code>3e~7</code>	<code>~3e~7</code>
Non-examples:	23	<code>.3</code>	<code>4.E5</code>	<code>1E2.0</code>	<code>1E+7</code>	<code>1E-7</code>	

Word constants

Examples:	<code>0w0</code>	<code>0w4</code>	<code>0w999999</code>	<code>0wxFFFF</code>	<code>0wx1ff</code>	
Non-examples:	<code>0w0.0</code>	<code>~0w4</code>	<code>-0w4</code>	<code>0w1E0</code>	<code>0wXFFFF</code>	<code>0WxFFFF</code>

String constants

A string constant is a sequence, between quotes (`"`), of zero or more printable characters, spaces, or escape sequences. An escape sequence starts with the escape character `\` and stands for a character sequence:

<code>\a</code>	A single character interpreted by the system as alert (BEL, ASCII 7).
<code>\b</code>	Backspace (BS, ASCII 8).
<code>\t</code>	Horizontal tab (HT, ASCII 9).
<code>\n</code>	Linefeed, also known as newline (LF, ASCII 10).
<code>\v</code>	Vertical tab (VT, ASCII 11).
<code>\f</code>	Form feed (FF, ASCII 12).
<code>\r</code>	Carriage return (CR, ASCII 13).
<code>\^c</code>	The control character <i>c</i> , where <i>c</i> may be any character with ASCII code 64–95 (<code>@</code> to <code>_</code>). The ASCII code of <code>\^c</code> is 64 less than that of <i>c</i> .
<code>\ddd</code>	The character with code <i>ddd</i> (3 decimal digits denoting an integer 0–255).
<code>\"</code>	<code>"</code>
<code>\\</code>	<code>\</code>
<code>\f··f\</code>	This sequence is ignored, where <i>f··f</i> stands for a sequence of one or more formatting characters (such as space, tab, newline, form-feed).

Character constants

A character constant consists of the symbol `#` immediately followed by a string constant of length one.

Examples:	<code>#"a"</code>	<code>#"n"</code>	<code>#"^Z"</code>	<code>#"255"</code>	<code>#"\""</code>
Non-examples:	<code># "a"</code>	<code>#c</code>	<code>#"\""</code>		

4 Identifiers

- **alphanumeric:** a sequence of letters, digits, primes (') and underbars (_) starting with a letter or prime;
- **symbolic:** any non-empty sequence of the following symbols:
! % & \$ # + - / : < = > ? @ \ ~ ' ^ | *

Reserved words (Section 1) are excluded. This means that for example # and | are not identifiers, but ## and |=| are identifiers. There are several classes of identifiers:

var	(value variables)	long
con	(value constructors)	long
excon	(exception constructors)	long
tyvar	(type variables)	
tycon	(type constructors)	long
lab	(record labels)	
unitid	(unit identifiers)	

- A type variable 'a is an alphanumeric identifier starting with a prime.
- A label lab is an identifier, or a positive integral numeral 1 2 3 ... not starting with 0.
- For each identifier class X marked 'long' above there is a class longX of long identifiers, which may have a qualifier consisting of a unit identifier followed by a dot '.':

$$\begin{array}{l} \text{long}x ::= x \quad \text{identifier} \\ \quad \quad \text{unitid}.x \quad \text{qualified identifier} \end{array}$$

5 Infix operators

An identifier may be given infix status by the `infix` or `infixr` directive, which may occur as a declaration. If identifier *id* has infix status, then $exp_1 \textit{id} exp_2$ may occur, in parentheses if necessary, wherever the application $id(exp_1, exp_2)$ or $id\{1=exp_1, 2=exp_2\}$ would otherwise occur. Infix identifiers in patterns are analogous. On the other hand, an occurrence of a qualified identifier, or any identifier prefixed by `op`, is treated as non-infix. The form of the fixity directives is as follows ($n \geq 1$):

$$\begin{array}{llll} \text{infix} & \langle d \rangle & id_1 \cdots id_n & \text{left associative} \\ \text{infixr} & \langle d \rangle & id_1 \cdots id_n & \text{right associative} \\ \text{nonfix} & & id_1 \cdots id_n & \text{non-associative} \end{array}$$

where $\langle d \rangle$ is an optional decimal digit *d* indicating binding precedence. A higher value of *d* indicates tighter binding; the default is 0. Fixity directives are subject to the usual scope rules governing visibility of identifiers declared inside `let` and `local`.

Mixed left-associative operators of the same precedence associate to the left, mixed right-associative operators of the same precedence associate to the right, and it is illegal to mix left- and right-associative operators of the same precedence.

6 Notational conventions used in the grammar

- Each syntax class is defined by a list of alternatives, one alternative on each line. An empty phrase is represented by an empty line.
- The brackets \langle and \rangle enclose optional phrases.
- For any syntax class X (over which *x* ranges) we define the syntax class Xseq (over which *xseq* ranges) as follows:

$$\begin{array}{ll} \textit{xseq} ::= x & \text{(singleton sequence)} \\ & \text{(empty sequence)} \\ & (x_1, \dots, x_n) \quad \text{(sequence, } n \geq 1) \end{array}$$

- Alternative phrases are listed in order of decreasing precedence.
- L and R indicate left and right association.
- The syntax of types binds more tightly than that of expressions.
- Each iterated construct (e.g. *match*) extends as far to the right as possible. Hence a *case* inside a *case*, *fn*, or *fun* may have to be enclosed in parentheses.

7 Grammar for the Standard ML Core language

Expressions and matches

<i>exp</i>	::=	<i>infixp</i>	
		<i>exp</i> : <i>ty</i>	type constraint (L)
		<i>exp</i> ₁ <i>andalso</i> <i>exp</i> ₂	sequential conjunction
		<i>exp</i> ₁ <i>orelse</i> <i>exp</i> ₂	sequential disjunction
		<i>exp</i> <i>handle</i> <i>match</i>	handle exception
		<i>raise</i> <i>exp</i>	raise exception
		<i>if</i> <i>exp</i> ₁ <i>then</i> <i>exp</i> ₂ <i>else</i> <i>exp</i> ₃	conditional
		<i>while</i> <i>exp</i> ₁ <i>do</i> <i>exp</i> ₂	iteration
		<i>case</i> <i>exp</i> <i>of</i> <i>match</i>	case analysis
		<i>fn</i> <i>match</i>	function expression
<i>infixp</i>	::=	<i>appexp</i>	
		<i>infixp</i> ₁ <i>id</i> <i>infixp</i> ₂	infix application
<i>appexp</i>	::=	<i>atexp</i>	
		<i>appexp</i> <i>atexp</i>	application
<i>atexp</i>	::=	<i>scn</i>	special constant (see Section 3)
		$\langle \text{op} \rangle$ <i>longvar</i>	value variable
		$\langle \text{op} \rangle$ <i>longcon</i>	value constructor
		$\langle \text{op} \rangle$ <i>longexcon</i>	exception constructor
		{ <i>exprow</i> }	record
		# <i>lab</i>	record selector
		()	0-tuple
		(<i>exp</i> ₁ , ..., <i>exp</i> _{<i>n</i>})	<i>n</i> -tuple, <i>n</i> ≥ 2
		[<i>exp</i> ₁ , ..., <i>exp</i> _{<i>n</i>}]	list, <i>n</i> ≥ 0
		#[<i>exp</i> ₁ , ..., <i>exp</i> _{<i>n</i>}]	vector, <i>n</i> ≥ 0
		(<i>exp</i> ₁ ; ... ; <i>exp</i> _{<i>n</i>})	sequence, <i>n</i> ≥ 2
		<i>let</i> <i>dec</i> <i>in</i> <i>exp</i> ₁ ; ... ; <i>exp</i> _{<i>n</i>} <i>end</i>	local declaration, <i>n</i> ≥ 1
		(<i>exp</i>)	
<i>exprow</i>	::=	<i>lab</i> = <i>exp</i> \langle , <i>exprow</i> \rangle	expression row
<i>match</i>	::=	<i>mrule</i> \langle <i>match</i> \rangle	
<i>mrule</i>	::=	<i>pat</i> => <i>exp</i>	

Declarations and Bindings

<i>dec</i>	<code>::= val <i>tyvarseq</i> <i>valbind</i> fun <i>tyvarseq</i> <i>fvalbind</i> type <i>typbind</i> datatype <i>datbind</i> < withtype <i>typbind</i> > abstype <i>datbind</i> < withtype <i>typbind</i> > with <i>dec</i> end exception <i>exbind</i> local <i>dec</i>₁ in <i>dec</i>₂ end open <i>unitid</i>₁ ... <i>unitid</i>_{<i>n</i>} <i>dec</i>₁ <;> <i>dec</i>₂ infix <<i>d</i>> <i>id</i>₁ ... <i>id</i>_{<i>n</i>} infixr <<i>d</i>> <i>id</i>₁ ... <i>id</i>_{<i>n</i>} nonfix <i>id</i>₁ ... <i>id</i>_{<i>n</i>}</code>	value declaration function declaration type declaration datatype declaration abstype declaration exception declaration local declaration open declaration, $n \geq 1$ empty declaration sequential declaration infix (left) directive, $n \geq 1$ infix (right) directive, $n \geq 1$ nonfix directive, $n \geq 1$
<i>valbind</i>	<code>::= <i>pat</i> = <i>exp</i> < and <i>valbind</i> > rec <i>valbind</i></code>	value binding recursive binding
<i>fvalbind</i>	<code>::= <op> var <i>atpat</i>₁₁ ... <i>atpat</i>_{1<i>n</i>} <:ty> = <i>exp</i>₁ <op> var <i>atpat</i>₂₁ ... <i>atpat</i>_{2<i>n</i>} <:ty> = <i>exp</i>₂ ... <op> var <i>atpat</i>_{<i>m</i>1} ... <i>atpat</i>_{<i>m</i><i>n</i>} <:ty> = <i>exp</i>_{<i>m</i>} < and <i>fvalbind</i> ></code>	$m, n \geq 1$
<i>typbind</i>	<code>::= <i>tyvarseq</i> <i>tycon</i> = <i>ty</i> < and <i>typbind</i> ></code>	
<i>datbind</i>	<code>::= <i>tyvarseq</i> <i>tycon</i> = <i>conbind</i> < and <i>datbind</i> ></code>	
<i>conbind</i>	<code>::= <op> <i>con</i> <of <i>ty</i>> < <i>conbind</i> ></code>	
<i>exbind</i>	<code>::= <op> <i>excon</i> <of <i>ty</i>> < and <i>exbind</i> > <op> <i>excon</i> = <op> <i>longexcon</i> < and <i>exbind</i> ></code>	

Note: In the *fvalbind* form above, if *var* has infix status then either *op* must be present, or *var* must be infix. Thus, at the start of any clause, *op var (atpat, atpat')* may be written *(atpat var atpat')*. The parentheses may be dropped if ':ty' or '=' follows immediately.

Type expressions

<i>ty</i>	<code>::= <i>tyvar</i> { < <i>tyrow</i> > } <i>tyseq</i> <i>longtycon</i> <i>ty</i>₁ * ... * <i>ty</i>_{<i>n</i>} <i>ty</i>₁ -> <i>ty</i>₂ (<i>ty</i>)</code>	type variable record type expression type construction tuple type, $n \geq 2$ function type expression
<i>tyrow</i>	<code>::= <i>lab</i> : <i>ty</i> < , <i>tyrow</i> ></code>	type-expression row

Patterns

$atpat$	$::=$	$_$ $scon$ $\langle op \rangle var$ $\langle op \rangle longcon$ $\langle op \rangle longexcon$ $\{ \langle patrow \rangle \}$ $()$ (pat_1, \dots, pat_n) $[pat_1, \dots, pat_n]$ $\#[pat_1, \dots, pat_n]$ (pat)	wildcard special constant (see Section 3) variable constructor exception constructor record 0-tuple n -tuple, $n \geq 2$ list, $n \geq 0$ vector, $n \geq 0$
$patrow$	$::=$	\dots $lab = pat \langle , patrow \rangle$ $lab \langle :ty \rangle \langle as pat \rangle \langle , patrow \rangle$	wildcard pattern row label as variable
pat	$::=$	$atpat$ $\langle op \rangle longcon atpat$ $\langle op \rangle longexcon atpat$ $pat_1 con pat_2$ $pat_1 excon pat_2$ $pat : ty$ $\langle op \rangle var \langle :ty \rangle as pat$	atomic pattern value construction exception construction infix value construction infix exception construction typed layered

Syntactic restrictions

- No pattern may bind the same var twice. No expression row, pattern row or type row may bind the same lab twice.
- No binding $valbind$, $tybind$, $datbind$ or $exbind$ may bind the same identifier twice; this applies also to value constructors within a $datbind$.
- In the left side $tyvarseq tycon$ of any $tybind$ or $datbind$, $tyvarseq$ must not contain the same $tyvar$ twice. Any $tyvar$ occurring within the right side must occur in $tyvarseq$.
- For each value binding $pat = exp$ within rec , exp must be of the form $fn match$, possibly enclosed in parentheses, and possibly constrained by one or more type expressions.
- No $datbind$ or $exbind$ may bind $true$, $false$, it , nil , $::$, or ref .

8 Interactive sessions

An expression exp which occurs grammatically at top-level in an interactive session is taken to be an abbreviation for the declaration

```
val it = exp
```

This convention applies to interactive sessions only. In a batch-compiled unit, write `val it = exp` or `val _ = exp` etc.

9 Grammar for the Moscow ML module language

The Moscow ML module language is much simplified compared to the full Standard ML Modules language.

The equivalent of a Standard ML structure is a unit. A unit U consists of a unit interface in file $U.sig$ (corresponding to an SML signature) and a unit body in file $U.sml$ (corresponding to an SML structure). The unit signature can be left out.

Unit Body (in file $unitid.sml$)

$unitbody$	<code>::= structure $unitid$ = struct dec end</code>	structure
	<code>structure $unitid$:> $unitid$ = struct dec end</code>	structure with signature
	<code>dec</code>	structure (old syntax)

Unit Signature (in file $unitid.sig$)

$unitsig$	<code>::= signature $unitid$ = sig $uspec$ end</code>	named signature
	<code>$uspec$</code>	signature (old syntax)
$uspec$	<code>::= val $valdesc$</code>	value specification
	<code>type $typdesc$</code>	abstract type
	<code>type $typbind$</code>	type abbreviation
	<code>eqtype $typdesc$</code>	abstract equality type
	<code>datatype $datdesc$</code>	datatype
	<code>datatype $datdesc$ withtype $typbind$</code>	datatype with typbind
	<code>exception $exdesc$</code>	exception
	<code>local $lspec$ in $uspec$ end</code>	local specifications
	<code>$uspec$ <;> $uspec$</code>	empty
	<code>$uspec$ <;> $uspec$</code>	sequential
$lspec$	<code>::= open $unitid_1 \dots unitid_n$</code>	open other units
	<code>type $typbind$</code>	type abbreviation
	<code>local $lspec$ in $lspec$ end</code>	local specifications
	<code>$lspec$ <;> $lspec$</code>	empty
	<code>$lspec$ <;> $lspec$</code>	sequential
$valdesc$	<code>::= var : ty < and $valdesc$ ></code>	value description
$typdesc$	<code>::= $tyvarseq$ $tycon$ < and $typdesc$ ></code>	type constructor description
$datdesc$	<code>::= $tyvarseq$ $tycon$ = $condesc$ < and $datdesc$ ></code>	datatype description
$condesc$	<code>::= con <of ty> < $condesc$ ></code>	constructor description
$exdesc$	<code>::= $excon$ <of ty> < and $exdesc$ ></code>	exception constructor description

Syntactic restrictions

- In Moscow ML, the $unitid$, if specified in a unit signature or unit body file, must agree with the filename. The name of the constraining signature, if any, must equal that of the structure.
- No specification $valdesc$, $typdesc$, $typbind$, $datdesc$, or $exdesc$ may describe the same identifier twice; this applies also to value constructors within a $datdesc$.
- In the left side $tyvarseq$ $tycon$ in any $typdesc$, $typbind$, or $datdesc$, $tyvarseq$ must not contain the same $tyvar$ twice. Any $tyvar$ occurring within the right side of a $typbind$ or $datdesc$ must occur in $tyvarseq$.
- No type, value, or exception may be specified twice at top-level in a signature.
- No $datdesc$ or $exdesc$ may specify `true`, `false`, `it`, `nil`, `::`, or `ref`.

10 Built-in types, constructors and exceptions

The following types, constructors, and exceptions are available in the initial environment, of the interactive system as well as files compiled with the batch compiler `mosmlc` or the `compile` function.

Built-in types

Type	Values	Admits equality	Constructors and constants
'a array	Arrays	yes	
bool	Booleans	yes	false, true
char	Characters	yes	#"a", #"b", ...
exn	Exceptions	no	
'a frag	Quotation fragments	if 'a does	QUOTE, ANTIQUOTE
int	Integers	yes	241, 0xF1, ...
'a list	Lists	if 'a does	nil, ::
'a option	Optional results	if 'a does	NONE, SOME
order	Comparisons	yes	LESS, EQUAL, GREATER
real	Floating-point numbers	yes	
'a ref	References	yes	ref
string	Strings	yes	
substring	Substrings	no	
unit	The empty tuple ()	yes	
'a vector	Vectors	if 'a does	
word	Words (31-bit)	yes	0w241, 0wxF1, ...
word8	Bytes (8 bit)	yes	0w241, 0wxF1, ...

Built-in exception constructors

```
Bind Chr Domain Div Fail Graphic_failure Interrupt
Io Match Option Ord Overflow Size Subscript
```


11 Built-in variables and functions

For each variable we list its type, effect, and any exception it may raise. Some built-in identifiers are overloaded; this is specified using *overloading classes*. For instance, an identifier whose type involves the overloading class `realint` stands for two functions: one in which `realint` (in the type) is consistently replaced by `int`, and another in which `realint` is consistently replaced by `real`. The overloading classes are:

Overloading class	Corresponding base types
<code>realint</code>	<code>int, real</code>
<code>wordint</code>	<code>int, word, word8</code>
<code>num</code>	<code>int, real, word, word8</code>
<code>numtxt</code>	<code>int, real, word, word8, char, string</code>

When the context does not otherwise resolve the overloading, it defaults to `int`.

Nonfix identifiers in the initial environment

<i>id</i>	type	effect	exception
<code>~</code>	<code>realint -> realint</code>	arithmetic negation	Overflow
<code>!</code>	<code>'a ref -> 'a</code>	dereference	
<code>abs</code>	<code>realint -> realint</code>	absolute value	Overflow
<code>app</code>	<code>('a -> unit) -> 'a list -> unit</code>	apply to all elements	
<code>ceil</code>	<code>real -> int</code>	round towards $+\infty$	Overflow
<code>chr</code>	<code>int -> char</code>	character with number	Chr
<code>concat</code>	<code>string list -> string</code>	concatenate strings	Size
<code>explode</code>	<code>string -> char list</code>	list of characters in string	
<code>false</code>	<code>bool</code>	logical falsehood	
<code>floor</code>	<code>real -> int</code>	round towards $-\infty$	Overflow
<code>foldl</code>	<code>('a*'b->'b)->'b->'a list->'b</code>	fold from left to right	
<code>foldr</code>	<code>('a*'b->'b)->'b->'a list->'b</code>	fold from right to left	
<code>hd</code>	<code>'a list -> 'a</code>	first element	Empty
<code>help</code>	<code>string -> unit</code>	simple help utility	
<code>ignore</code>	<code>'a -> unit</code>	discard argument	
<code>implode</code>	<code>char list -> string</code>	make string from characters	Size
<code>length</code>	<code>'a list -> int</code>	length of list	
<code>map</code>	<code>('a -> 'b) -> 'a list -> 'b list</code>	map over all elements	
<code>nil</code>	<code>'a list</code>	empty list	
<code>not</code>	<code>bool -> bool</code>	logical negation	
<code>null</code>	<code>'a list -> bool</code>	true if list is empty	
<code>ord</code>	<code>char -> int</code>	number of character	
<code>print</code>	<code>string -> unit</code>	print on standard output	
<code>real</code>	<code>int -> real</code>	<code>int</code> to <code>real</code>	
<code>ref</code>	<code>'a -> 'a ref</code>	create reference value	
<code>rev</code>	<code>'a list -> 'a list</code>	reverse list	
<code>round</code>	<code>real -> int</code>	round to nearest integer	Overflow
<code>size</code>	<code>string -> int</code>	length of string	
<code>str</code>	<code>char -> string</code>	create one-character string	
<code>tl</code>	<code>'a list -> 'a list</code>	tail of list	Empty
<code>true</code>	<code>bool</code>	logical truth	
<code>trunc</code>	<code>real -> int</code>	round towards 0	Overflow
<code>vector</code>	<code>'a list -> 'a vector</code>	make vector from list	Size

Infix identifiers in the initial environment

<i>id</i>	type	effect	exception
Infix precedence 7:			
/	real * real -> real	floating-point quotient	Div, Overflow
div	wordint * wordint -> wordint	quotient (round towards $-\infty$)	Div, Overflow
mod	wordint * wordint -> wordint	remainder (of div)	Div, Overflow
*	num * num -> num	product	Overflow
Infix precedence 6:			
+	num * num -> num	sum	Overflow
-	num * num -> num	difference	Overflow
^	string * string -> string	concatenate	Size
Infix precedence 5:			
::	'a * 'a list -> 'a list	cons onto list (R)	
@	'a list * 'a list -> 'a list	append lists (R)	
Infix precedence 4:			
=	''a * ''a -> bool	equal to	
<>	''a * ''a -> bool	not equal to	
<	numtxt * numtxt -> bool	less than	
<=	numtxt * numtxt -> bool	less than or equal to	
>	numtxt * numtxt -> bool	greater than	
>=	numtxt * numtxt -> bool	greater than or equal to	
Infix precedence 3:			
:=	'a ref * 'a -> unit	assignment	
o	('b->'c) * ('a->'b) -> ('a->'c)	function composition	
Infix precedence 0:			
before	'a * 'b -> 'a	return first argument	

Built-in functions available only in the interactive system (unit Meta)

<i>id</i>	type	effect	exception
compile	string -> unit	compile unit (U.sig or U.sml)	Fail
installPP	(ppstream->'a->unit)->unit	install prettyprinter	
load	string -> unit	load unit U and any units it needs	Fail
loadOne	string -> unit	load unit U (only)	Fail
loadPath	string list ref	search path for load, loadOne, use	
printVal	'a -> 'a	print value on stdout	
printDepth	int ref	limit printed data depth	
printLength	int ref	limit printed list and vector length	
quietdec	bool ref	suppress prompt and responses	
quit	unit -> unit	quit the interactive system	
quotation	bool ref	permit quotations in source code	
system	string -> int	execute operating system command	
use	string -> unit	read declarations from file	
valuepoly	bool ref	adopt value polymorphism	
verbose	bool ref	permit feedback from compile	

- These functions are not part of the SML Basis Library.
- The Moscow ML Owner's Manual describes how to use `compile` and `load` to perform separate compilation, and how to use quotations. Evaluating `load U` automatically loads any units needed by U, and does nothing if U is already loaded; whereas `loadOne U` fails if any unit needed by U is not loaded, or if U is already loaded. The `loadPath` variable determines where `load`, `loadOne`, and `use` will look for files.

12 List of all library units

Mosml ML's predefined library units are shown in Figure 1 below. The status of each unit is indicated according to this table:

S	:	the unit belongs to the SML Basis Library.
D	:	the unit is preloaded by default.
F	:	the unit is loaded when option <code>-P full</code> is specified.
N	:	the unit is loaded when option <code>-P nj93</code> is specified.
O	:	the unit is loaded when option <code>-P sm190</code> is specified.

More information about the Moscow ML library:

- The index in Section 14 (page 17) lists all identifiers defined in Moscow ML library units.
- Typing `help "lib"`; in a `mosml` session gives a list of all library units.
- Typing `help "unit"`; in a `mosml` session gives information about library unit *unit*.
- Typing `help "id"`; in a `mosml` session gives information about identifier *id*, regardless which library unit(s) it is defined in.
- On the Internet the same documentation is online at

<http://www.dina.kvl.dk/~sestoft/mosmllib/>

Also, you may download the HTML files documenting the Moscow ML library from the Moscow ML home page and install them locally for faster browsing.

Library unit	Description	Status
Array	Mutable polymorphic arrays	SDF
Array2	Two-dimensional arrays	S
Arraysort	Array sorting (quicksort)	
BasicIO	Input-output as in SML'90	DF
Binarymap	Binary tree implementation of finite maps	
Binaryset	Binary tree implementation of finite sets	
BinIO	Binary input-output streams (imperative)	S F
Bool	Booleans	S F
Byte	Conversions between Word8 and Char	S F
Char	Characters	SDF
CharArray	Mutable arrays of characters	S F
CharVector	Immutable character vectors (that is, strings)	S F
CommandLine	Program name and arguments	S F
Date	From time points to dates and vice versa	S F
Dynarray	Dynamic arrays	
Dynlib	Dynamic linking with C (Linux+Solaris)	
FileSys	File system interface	S F
Gdbm	Persistent hash tables of strings (GNU gdbm)	
General	Various top-level primitives	SD
Graphics	Graphics primitives (DOS only)	
Help	On-line help	DFNO
Int	Integer arithmetic and comparisons	S F
Intmap	Finite maps from integers	
Intset	Finite sets of integers	
List	Lists	SDFNO
ListPair	Pairs of lists	S F
Listsort	List sorting (mergesort)	
Location	Error reporting for lexers and parsers	
Math	Trigonometric and transcendental functions	S F
Meta	Functions specific to the interactive system	
Mosml	Various Moscow ML utilities	F
Mosmlcgi	Utilities for writing CGI programs	
NJ93	Top-level compatibility with SML/NJ 0.93	N
OS	Operating system interface	S F
Option	Partial functions	SDFNO
Path	File pathnames	S F
Polygdbm	Polymorphic persistent hash tables (GNU gdbm)	
Polyhash	Polymorphic hash tables	
PP	General prettyprinters	F
Process	Process interface	S F
Random	Generation of pseudo-random numbers	
Real	Real arithmetic and comparisons	S F
SML90	Top-level compatibility with 1990 Definition	S 0
Splaymap	Splay-tree implementation of finite maps	
Splayset	Splay-tree implementation of finite sets	
String	String utilities	SDF
StringCvt	Conversion to and from strings	S F
Substring	Scanning of substrings	S F
TextIO	Text input-output streams (imperative)	SDF
Time	Time points and durations	S F
Timer	Timing operations	S F
Vector	Immutable vectors	SDF
Weak	Arrays of weak pointers	
Word	Unsigned 31-bit integers ('machine words')	S F
Word8	Unsigned 8-bit integers (bytes)	S F
Word8Array	Mutable arrays of unsigned 8-bit integers	S F
Word8Vector	Immutable vectors of unsigned 8-bit integers	S F

Figure 1: Predefined library units

13 The preloaded library units

The following libraries are preloaded by default: `Array`, `Char`, `List`, `String`, `TextIO`, and `Vector`. To load any other library *lib*, evaluate `load "lib"` in the interactive system.

Notation in the tables below

<i>f</i>	functional argument
<i>n</i>	integer
<i>p</i>	predicate of type ('a -> bool)
<i>s</i>	string
<i>xs, ys</i>	lists

List manipulation functions (unit List)

<i>id</i>	type	effect
@	'a list * 'a list -> 'a list	append
all	('a -> bool) -> 'a list -> bool	if <i>p</i> true of all elements
app	('a -> unit) -> 'a list -> unit	apply <i>f</i> to all elements
concat	'a list list -> 'a list	concatenate lists
drop	'a list * int -> 'a list	drop <i>n</i> first elements
exists	('a -> bool) -> 'a list -> bool	if <i>p</i> true of some element
filter	('a -> bool) -> 'a list -> 'a list	the elements for which <i>p</i> is true
find	('a -> bool) -> 'a list -> 'a option	first element for which <i>p</i> is true
foldl	('a * 'b -> 'b) -> 'b -> 'a list -> 'b	fold from left to right
foldr	('a * 'b -> 'b) -> 'b -> 'a list -> 'b	fold from right to left
hd	'a list -> 'a	first element
last	'a list -> 'a	last element
length	'a list -> int	number of elements
map	('a -> 'b) -> 'a list -> 'b list	results of applying <i>f</i> to all elements
mapPartial	('a -> 'b option) -> 'a list -> 'b list	list of the non-NONE results of <i>f</i>
nth	'a list * int -> 'a	<i>n</i> 'th element (0-based)
null	'a list -> bool	true if list is empty
partition	('a->bool)->'a list->'a list*'a list	compute (true for <i>p</i> , false for <i>p</i>)
rev	'a list -> 'a list	reverse list
revAppend	'a list * 'a list -> 'a list	compute (rev <i>xs</i>) @ <i>ys</i>
tabulate	int * (int -> 'a) -> 'a list	compute [<i>f</i> (0), ..., <i>f</i> (<i>n</i> -1)]
take	'a list * int -> 'a list	take <i>n</i> first elements
tl	'a list -> 'a list	tail of list

- Functions `hd`, `tl`, and `last` may raise exception `Empty`; functions `drop`, `nth`, and `take` may raise exception `Subscript`.
- For a more detailed description, type `help "List"`; or see file `mosml/lib/List.sig`. The `List` unit is loaded and partially opened in the initial environment, making the following functions available: `@`, `app`, `foldl`, `foldr`, `hd`, `length`, `map`, `null`, `rev`, `tl`.

Built-in values and functions for text-mode input/output (unit TextIO)

<i>id</i>	type	effect
closeIn	instream -> unit	close input stream
closeOut	outstream -> unit	close output stream
endOfStream	instream -> bool	true if at end of stream
flushOut	outstream -> unit	flush output to consumer
input	instream -> string	input some characters
input1	instream -> char option	input one character
inputN	instream * int -> string	input at most <i>n</i> characters
inputAll	instream -> string	input all available characters
inputLine	instream -> string	read up to (and including) next end of line
inputNoBlock	instream -> string option	read, if possible without blocking
lookahead	instream -> char option	get next char non-destructively
openAppend	string -> outstream	open file for appending to it
openIn	string -> instream	open file for input
openOut	string -> outstream	open file for output
output	outstream * string -> unit	write string to output stream
output1	outstream * char -> unit	write character to output stream
print	string -> unit	write to standard output
stderr	outstream	standard error output stream
stdin	instream	standard input stream
stdout	outstream	standard output stream

- For a more detailed description, see file `mosml/lib/TextIO.sig`, or type `help "TextIO";`.
- For the corresponding structure `BinIO` for binary (untranslated) input and output, see `help "BinIO"`.
- On error, these functions will raise exception `Io` with an explanatory argument.

String manipulation functions (unit String)

<i>id</i>	type	effect
<code>^</code>	string * string -> string	concatenate strings
collate	(char*char->order)->string*string->order	compare strings
compare	string * string -> order	compare strings
concat	string list -> string	concatenate list of strings
explode	string -> char list	character list from string
extract	string * int * int option -> string	get substring or tail
fields	(char -> bool) -> string -> string list	find (possibly empty) fields
fromCString	string -> string option	parse C escape sequences
fromString	string -> string option	parse ML escape sequences
implode	char list -> string	string from character list
isPrefix	string -> string -> bool	prefix test
maxSize	int	maximal size of a string
size	string -> int	length of string
str	char -> string	make one-character string
sub	string * int -> char	<i>n</i> 'th character (0-based)
substring	string * int * int -> string	get substring (<i>s</i> , <i>first</i> , <i>len</i>)
toCString	string -> string	make C escape sequences
toString	string -> string	make ML escape sequences
tokens	(char -> bool) -> string -> string list	find (non-empty) tokens
translate	(char -> string) -> string -> string	apply <i>f</i> and concatenate

- Functions `^`, `concat`, `implode`, and `translate` may raise exception `Size`, and `sub` and `substring` may raise `Subscript`.
- In addition, the overloaded comparison operators `<`, `<=`, `>`, `>=` work on strings.
- For a more detailed description, see file `mosml/lib/String.sig`, or type `help "String";`.

Vector manipulation functions (unit Vector)

Type `'a vector` is the type of one-dimensional, immutable, zero-based constant time access vectors with elements of type `'a`. Type `'a vector` admits equality if `'a` does.

<i>id</i>	type	effect
<code>app</code>	<code>('a -> unit) -> 'a vector -> unit</code>	apply f left-right
<code>appi</code>	<code>(int * 'a -> unit) -> 'a vector * int * int option -> unit</code>	
<code>concat</code>	<code>'a vector list -> 'a vector</code>	concatenate vectors
<code>extract</code>	<code>'a vector * int * int option -> 'a vector</code>	extract a subvector or tail
<code>foldl</code>	<code>('a * 'b -> 'b) -> 'b -> 'a vector -> 'b</code>	fold f left-right
<code>foldli</code>	<code>(int * 'a * 'b -> 'b) -> 'b -> 'a vector*int*int option -> 'b</code>	
<code>foldr</code>	<code>('a * 'b -> 'b) -> 'b -> 'a vector -> 'b</code>	fold f right-left
<code>foldri</code>	<code>(int * 'a * 'b -> 'b) -> 'b -> 'a vector*int*int option -> 'b</code>	
<code>fromList</code>	<code>'a list -> 'a vector</code>	make vector from the list
<code>length</code>	<code>'a vector -> int</code>	length of the vector
<code>maxLen</code>	<code>int</code>	maximal vector length
<code>sub</code>	<code>'a vector * int -> 'a</code>	n 'th element (0-based)
<code>tabulate</code>	<code>int * (int -> 'a) -> 'a vector</code>	vector of $f(0), \dots, f(n-1)$

- Functions `fromList`, `tabulate`, and `concat` may raise exception `Size`, and `sub` and `extract` may raise `Subscript`.
- For a more detailed description, type `help "Vector"`; or see file `mosml/lib/Vector.sig`.

Array manipulation functions (unit Array)

Type `'a array` is the type of one-dimensional, mutable, zero-based constant time access arrays with elements of type `'a`. Type `'a array` admits equality regardless whether `'a` does.

<i>id</i>	type	effect
<code>app</code>	<code>('a -> unit) -> 'a array -> unit</code>	apply f left-right
<code>appi</code>	<code>(int * 'a -> unit) -> 'a array * int * int option -> unit</code>	
<code>array</code>	<code>int * 'a -> 'a array</code>	create and initialize array
<code>copy</code>	<code>{src : 'a array, si : int, len : int option, dst : 'a array, di : int} -> unit</code>	copy subarray to subarray
<code>copyVec</code>	<code>{src : 'a vector, si : int, len : int option, dst : 'a array, di : int} -> unit</code>	copy subvector to subarray
<code>extract</code>	<code>'a array * int * int option -> 'a vector</code>	extract subarray to vector
<code>foldl</code>	<code>('a * 'b -> 'b) -> 'b -> 'a array -> 'b</code>	fold left-right
<code>foldli</code>	<code>(int * 'a * 'b -> 'b) -> 'b -> 'a array * int * int option -> 'b</code>	
<code>foldr</code>	<code>('a * 'b -> 'b) -> 'b -> 'a array -> 'b</code>	fold right-left
<code>foldri</code>	<code>(int * 'a * 'b -> 'b) -> 'b -> 'a array * int * int option -> 'b</code>	
<code>fromList</code>	<code>'a list -> 'a array</code>	make array from the list
<code>length</code>	<code>'a array -> int</code>	length of the array
<code>maxLen</code>	<code>int</code>	maximal array length
<code>modify</code>	<code>('a -> 'a) -> 'a array -> unit</code>	apply f and update
<code>modifyi</code>	<code>(int * 'a -> 'a) -> 'a array * int * int option -> unit</code>	
<code>sub</code>	<code>'a array * int -> 'a</code>	n 'th element (0-based)
<code>tabulate</code>	<code>int * (int -> 'a) -> 'a array</code>	array of $f(0), \dots, f(n-1)$
<code>update</code>	<code>'a array * int * 'a -> unit</code>	set n 'th element (0-based)

- Functions `array`, `tabulate`, and `fromList` may raise exception `Size`, and `sub`, `update`, `extract`, and `copy` may raise `Subscript`.
- For a more detailed description, type `help "Array"`; or see file `mosml/lib/Array.sig`. The `Array` unit is loaded but not opened in the initial environment.

Character manipulation functions (unit Char)

<i>id</i>	type	effect	exception
<code>chr</code>	<code>int -> char</code>	from character code to character	<code>Chr</code>
<code>compare</code>	<code>char * char -> order</code>	compare character codes	
<code>contains</code>	<code>string -> char -> bool</code>	contained in string	
<code>fromCString</code>	<code>string -> char option</code>	parse C escape sequence	
<code>fromString</code>	<code>string -> char option</code>	parse SML escape sequence	
<code>isAlpha</code>	<code>char -> bool</code>	alphabetic ASCII character	
<code>isAlphaNum</code>	<code>char -> bool</code>	alphanumeric ASCII character	
<code>isAscii</code>	<code>char -> bool</code>	seven-bit ASCII character	
<code>isCntrl</code>	<code>char -> bool</code>	ASCII control character	
<code>isDigit</code>	<code>char -> bool</code>	decimal digit	
<code>isGraph</code>	<code>char -> bool</code>	printable and visible ASCII	
<code>isHexDigit</code>	<code>char -> bool</code>	hexadecimal digit	
<code>isLower</code>	<code>char -> bool</code>	lower case alphabetic (ASCII)	
<code>isPrint</code>	<code>char -> bool</code>	printable ASCII (including space)	
<code>isPunct</code>	<code>char -> bool</code>	printable, but not space or alphanumeric	
<code>isSpace</code>	<code>char -> bool</code>	space and lay-out (HT, CR, LF, VT, FF)	
<code>isUpper</code>	<code>char -> bool</code>	upper case alphabetic (ASCII)	
<code>maxChar</code>	<code>char</code>	last character (in <code><=</code> order)	
<code>maxOrd</code>	<code>int</code>	largest character code	
<code>minChar</code>	<code>char</code>	first character (in <code><=</code> order)	
<code>notContains</code>	<code>string -> char -> bool</code>	not in string	
<code>ord</code>	<code>char -> int</code>	from character to character code	
<code>pred</code>	<code>char -> char</code>	preceding character	<code>Chr</code>
<code>succ</code>	<code>char -> char</code>	succeeding character	<code>Chr</code>
<code>toLower</code>	<code>char -> char</code>	convert to lower case (ASCII)	
<code>toCString</code>	<code>char -> string</code>	make C escape sequence	
<code>toString</code>	<code>char -> string</code>	make SML escape sequence	
<code>toUpper</code>	<code>char -> char</code>	convert to upper case (ASCII)	

- In addition, the overloaded comparison operators `<`, `<=`, `>`, `>=` work on the `char` type.
- For a more detailed description, type `help "Char"`; or see file `mosml/lib/Char.sig`. The `Char` unit is loaded and partially opened in the initial environment, making the functions `chr` and `ord` available.

14 Moscow ML library index

For each identifier, we list its kind (constructor, exception, structure, type, or value) and the structure in which it is defined.

Help on any identifier *id* can be obtained by typing `help "id"`; in a `mosml` session.

- ! (value; General)
- * (value; General, Int, Real, Word, Word8)
- + (value; General, Int, Real, Time, Word, Word8)
- (value; General, Int, Real, Time, Word, Word8)
- / (value; General, Real)
- :: (constructor; General)
- := (value; General)
- < (value; Char, General, Int, Real, String, Time, Word, Word8)
- << (value; Word, Word8)
- <= (value; Char, General, Int, Real, String, Time, Word, Word8)
- <> (value; General)
- = (value; General)
- > (value; Char, General, Int, Real, String, Time, Word, Word8)
- >= (value; Char, General, Int, Real, String, Time, Word, Word8)
- >> (value; Word, Word8)
- @ (value; List)
- ^ (value; String)
- ~ (value; General, Int, Real)
- ~>> (value; Word, Word8)
- A_EXEC (constructor; FileSys)
- A_READ (constructor; FileSys)
- A_WRITE (constructor; FileSys)
- Abs (exception; SML90)
- abs (value; Int, Real)
- access (type; FileSys)
- access (value; FileSys)
- acos (value; Math)
- add (value; Binaryset, Gdbm, Intset, Polygdbm, Splayset)
- add_break (value; PP)
- add_newline (value; PP)
- add_string (value; PP)
- addList (value; Binaryset, Intset, Splayset)
- all (value; List, ListPair, Substring)
- AlreadyThere (exception; Gdbm, Polygdbm)
- andb (value; Word, Word8)
- ANTIQUOTE (constructor; General)
- app (value; Array, Array2, Binarymap, Binaryset, CharArray, CharVector, Gdbm, Intmap, Intset, List, ListPair, NJ93, Option, Polygdbm, Splaymap, Splayset, Substring, Vector, Weak, Word8Array, Word8Vector)
- app1 (value; Dynlib)
- app2 (value; Dynlib)
- app3 (value; Dynlib)
- app4 (value; Dynlib)
- app5 (value; Dynlib)
- appi (value; Array, Array2, CharArray, CharVector, Vector, Weak, Word8Array, Word8Vector)
- apply (value; Polyhash)
- Apr (constructor; Date)
- arctan (value; NJ93, SML90)
- arguments (value; CommandLine)
- argv (value; Mosml)
- Array (structure)
- array (type; Array, Array2, CharArray, Dynarray, Weak, Word8Array)
- array (value; Array, Array2, CharArray, Dynarray, Weak, Word8Array)
- Array2 (structure)
- Arraysort (structure)
- asin (value; Math)
- atan (value; Math)
- atan2 (value; Math)
- atExit (value; Process)
- Aug (constructor; Date)
- backtrack (value; Lexing)
- base (value; Path, Substring)
- BasicIO (structure)
- before (value; General)
- begin_block (value; PP)
- BIN (constructor; StringCvt)
- Binarymap (structure)
- Binaryset (structure)
- Bind (exception; General)
- BinIO (structure)
- Bool (structure)
- bool (type; Bool, General)
- bound (value; Dynarray)
- break_style (type; PP)
- browser (value; Help)
- bucketSizes (value; Polyhash)
- buff_input (value; Nonstdio)
- buff_output (value; Nonstdio)
- Byte (structure)
- bytesToString (value; Byte)
- byteToChar (value; Byte)
- can_input (value; BasicIO, NJ93)
- ceil (value; General, Real)
- ceiling (value; NJ93)
- cgi_annotation_server (value; Mosmlcgi)
- cgi_auth_type (value; Mosmlcgi)
- cgi_content_length (value; Mosmlcgi)
- cgi_content_type (value; Mosmlcgi)
- cgi_field_integer (value; Mosmlcgi)
- cgi_field_string (value; Mosmlcgi)
- cgi_field_strings (value; Mosmlcgi)
- cgi_fieldnames (value; Mosmlcgi)
- cgi_gateway_interface (value; Mosmlcgi)
- cgi_http_accept (value; Mosmlcgi)
- cgi_http_referer (value; Mosmlcgi)
- cgi_http_user_agent (value; Mosmlcgi)
- cgi_part (value; Mosmlcgi)
- cgi_partnames (value; Mosmlcgi)

cgi_parts (value; Mosmlcgi)
 cgi_path_info (value; Mosmlcgi)
 cgi_path_translated (value; Mosmlcgi)
 cgi_query_string (value; Mosmlcgi)
 cgi_remote_addr (value; Mosmlcgi)
 cgi_remote_host (value; Mosmlcgi)
 cgi_remote_ident (value; Mosmlcgi)
 cgi_remote_user (value; Mosmlcgi)
 cgi_request_method (value; Mosmlcgi)
 cgi_script_name (value; Mosmlcgi)
 cgi_server_name (value; Mosmlcgi)
 cgi_server_port (value; Mosmlcgi)
 cgi_server_protocol (value; Mosmlcgi)
 cgi_server_software (value; Mosmlcgi)
 Char (structure)
 char (type; Char, General)
 CharArray (structure)
 charToByte (value; Byte)
 CharVector (structure)
 chDir (value; FileSystem)
 checkCPUTimer (value; Timer)
 checkRealTimer (value; Timer)
 chr (value; Char, NJ93, SML90)
 clear_ppstream (value; PP)
 clearParser (value; Parsing)
 close_in (value; BasicIO, NJ93, SML90)
 close_out (value; BasicIO, NJ93, SML90)
 Closed (exception; Dynlib, Gdbm, Polygdbm)
 closeDir (value; FileSystem)
 closeIn (value; BinIO, TextIO)
 closeOut (value; BinIO, TextIO)
 collate (value; String, Substring)
 ColMajor (constructor; Array2)
 column (value; Array2)
 CommandLine (structure)
 compare (value; Char, Date, FileSystem, Int, Real, String, Substring, Time, Word, Word8)
 compile (value; Meta)
 compose (value; Option)
 composePartial (value; Option)
 concat (value; CharVector, List, Path, String, Substring, Vector, Word8Vector)
 CONSISTENT (constructor; PP)
 contains (value; Char)
 copy (value; Array, Array2, CharArray, Polyhash, Word8Array)
 copyVec (value; Array, CharArray, Word8Array)
 cos (value; Math, NJ93, SML90)
 cosh (value; Math)
 cpu_timer (type; Timer)
 createLexer (value; Lexing)
 createLexerString (value; Lexing)
 cs (type; StringCvt, TextIO)
 currentArc (value; Path)
 Date (exception; Date)
 Date (structure)
 date (type; Date)
 date (value; Date)
 datum (type; Gdbm)
 day (value; Date)
 DEC (constructor; StringCvt)
 Dec (constructor; Date)
 dec (value; NJ93)
 default (value; Dynarray)
 defaultBrowser (value; Help)
 delay (value; Susp)
 delete (value; Binaryset, Intset, Splayset)
 dest_ppstream (value; PP)
 dict (type; Binarymap, Splaymap)
 Diff (exception; SML90)
 difference (value; Binaryset, Intset, Splayset)
 dimensions (value; Array2)
 dir (value; Path)
 dirstream (type; FileSystem)
 displayLines (value; Help)
 Div (exception; General, Real)
 div (value; General, Int, Word, Word8)
 dlclose (value; Dynlib)
 dlHandle (type; Dynlib)
 dlopen (value; Dynlib)
 dlsym (value; Dynlib)
 Domain (exception; General)
 doubleVec (value; Mosml)
 drop (value; List)
 dropl (value; StringCvt, Substring)
 dropr (value; Substring)
 dummyAction (value; Lexing)
 Dynarray (structure)
 Dynlib (structure)
 e (value; Math)
 elem (type; BinIO, CharArray, CharVector, TextIO, Word8Array, Word8Vector)
 Empty (exception; List)
 empty (value; Binaryset, Intmap, Intset, Splayset)
 end_block (value; PP)
 end_of_stream (value; BasicIO, NJ93, SML90)
 endOfStream (value; BinIO, TextIO)
 EQUAL (constructor; General)
 equal (value; Binaryset, Intset, Splayset)
 errLocation (value; Location)
 errMsg (value; Location)
 errorMsg (value; OS)
 errPrompt (value; Location)
 exists (value; List, ListPair)
 exit (value; BasicIO, Process)
 exn (type; General)
 exnMessage (value; Meta)
 exnName (value; Meta)
 Exp (exception; SML90)
 exp (value; Math, NJ93, SML90)
 explode (value; NJ93, SML90, String, Substring)
 ext (value; Path)
 extract (value; Array, CharArray, CharVector, String, Substring, Vector, Word8Array, Word8Vector)
 Fail (exception; General)

Failure (constructor; Mosml)
 failure (value; Process)
 false (constructor; General)
 fast_really_input (value; Nonstdio)
 fastwrite (value; Gdbm, Polygdbm)
 Feb (constructor; Date)
 fields (value; String, Substring)
 file (value; Path)
 file_exists (value; Nonstdio)
 file_id (type; FileSys)
 fileId (value; FileSys)
 fileSize (value; FileSys)
 FileSys (structure)
 filter (value; List, Option, Polyhash)
 find (value; Binarymap, Binaryset, Gdbm, Intset,
 List, Polygdbm, Polyhash, Splaymap, Splayset)
 first (value; Substring)
 FIX (constructor; StringCvt)
 flag (type; Dynlib)
 floatVec (value; Mosml)
 Floor (exception; SML90)
 floor (value; General, Real)
 flush_out (value; BasicIO, NJ93)
 flush_ppstream (value; PP)
 flushOut (value; BinIO, TextIO)
 fmt (value; Date, Int, Real, Time, Word, Word8)
 fold (value; Array2, Gdbm, NJ93, Polygdbm)
 foldi (value; Array2)
 foldl (value; Array, Binarymap, Binaryset,
 CharArray, CharVector, Intmap, Intset, List,
 ListPair, Splaymap, Splayset, Substring,
 Vector, Weak, Word8Array, Word8Vector)
 foldli (value; Array, CharArray, CharVector,
 Vector, Weak, Word8Array, Word8Vector)
 foldr (value; Array, Binarymap, Binaryset,
 CharArray, CharVector, Intmap, Intset, List,
 ListPair, Splaymap, Splayset, Substring,
 Vector, Weak, Word8Array, Word8Vector)
 foldri (value; Array, CharArray, CharVector,
 Vector, Weak, Word8Array, Word8Vector)
 force (value; Susp)
 frag (type; General)
 Fri (constructor; Date)
 fromCString (value; Char, String)
 fromDefault (value; Real)
 fromInt (value; Int, Real, Word, Word8)
 fromLarge (value; Int)
 fromLargeInt (value; Word, Word8)
 fromLargeWord (value; Word, Word8)
 fromList (value; Array, Array2, CharArray,
 CharVector, Dynarray, Vector, Word8Array,
 Word8Vector)
 fromMicroseconds (value; Time)
 fromMilliseconds (value; Time)
 fromReal (value; Time)
 fromSeconds (value; Time)
 fromString (value; Bool, Char, Date, Int, Path,
 Real, String, Time, Word, Word8)
 fromTimeLocal (value; Date)

fromTimeUniv (value; Date)
 fullPath (value; FileSys)

Gdbm (structure)
 GdbmError (exception; Gdbm, Polygdbm)
 GEN (constructor; StringCvt)
 General (structure)
 generator (type; Random)
 get (value; Weak)
 getc (value; Substring)
 getLocation (value; Location)
 getDir (value; FileSys)
 getEnv (value; Process)
 getItem (value; List)
 getLexAbsPos (value; Lexing)
 getLexBuffer (value; Lexing)
 getLexCurrPos (value; Lexing)
 getLexeme (value; Lexing)
 getLexemeChar (value; Lexing)
 getLexemeEnd (value; Lexing)
 getLexemeStart (value; Lexing)
 getLexLastAction (value; Lexing)
 getLexLastPos (value; Lexing)
 getLexStartPos (value; Lexing)
 getNextChar (value; Lexing)
 getOpt (value; Option)
 getParent (value; Path)
 getVolume (value; Path)
 Graphic_failure (exception; General)
 GREATER (constructor; General)

hash (value; FileSys, Polyhash)
 hash_param (value; Polyhash)
 hash_table (type; Polyhash)
 hasKey (value; Gdbm, Polygdbm)
 Hd (exception; NJ93)
 hd (value; List, NJ93)
 Help (structure)
 help (value; Help)
 helpdirs (value; Help)
 HEX (constructor; StringCvt)
 hour (value; Date)

ignore (value; General)
 implode (value; NJ93, SML90, String)
 in_stream_length (value; Nonstdio)
 inc (value; NJ93)
 INCONSISTENT (constructor; PP)
 indexfiles (value; Help)
 input (value; BasicIO, BinIO, NJ93, SML90, TextIO)
 input1 (value; BinIO, TextIO)
 input_binary_int (value; Nonstdio)
 input_char (value; Nonstdio)
 input_line (value; BasicIO, NJ93)
 input_value (value; Nonstdio)
 inputAll (value; BinIO, TextIO)
 inputc (value; BasicIO, NJ93)
 inputLine (value; TextIO)
 inputN (value; BinIO, TextIO)
 inputNoBlock (value; BinIO, TextIO)

```

insert (value; Binarymap, Gdbm, Intmap, Polygdbm,
        Polyhash, Splaymap)
installPP (value; Meta)
instream (type; BasicIO, BinIO, NJ93, SML90,
          TextIO)
Int (structure)
int (type; General, Int)
Interrupt (exception; General)
intersection (value; Binaryset, Intset, Splayset)
Intmap (structure)
intmap (type; Intmap)
Intset (structure)
intset (type; Intset)
Io (exception; General)
isAbsolute (value; Path)
isAlpha (value; Char)
isAlphaNum (value; Char)
isAscii (value; Char)
isCanonical (value; Path)
isCntrl (value; Char)
isdead (value; Weak)
isDigit (value; Char)
isDir (value; FileSys)
isDst (value; Date)
isEmpty (value; Binaryset, Intset, Splayset,
         Substring)
isGraph (value; Char)
isHexDigit (value; Char)
isLink (value; FileSys)
isLower (value; Char)
isPrefix (value; String, Substring)
isPrint (value; Char)
isPunct (value; Char)
isRelative (value; Path)
isSome (value; Option)
isSpace (value; Char)
isSubset (value; Binaryset, Intset, Splayset)
isUpper (value; Char)
isweak (value; Weak)

Jan (constructor; Date)
join (value; Option, Splaytree)
joinBaseExt (value; Path)
joinDirFile (value; Path)
Jul (constructor; Date)
Jun (constructor; Date)

last (value; List)
length (value; Array, CharArray, CharVector, List,
        Vector, Weak, Word8Array, Word8Vector)
LESS (constructor; General)
lexbuf (type; Lexing)
Lexing (structure)
List (structure)
list (type; General, List)
listDir (value; Mosml)
listItems (value; Binarymap, Binaryset, Gdbm,
           Intmap, Intset, Polygdbm, Polyhash, Splaymap,
           Splayset)
listKeys (value; Gdbm, Polygdbm)

ListPair (structure)
Listsort (structure)
ln (value; Math, NJ93, SML90)
load (value; Meta)
loadOne (value; Meta)
loadPath (value; Meta)
Loc (constructor; Location)
localOffset (value; Date)
Location (structure)
Location (type; Location)
log10 (value; Math)
lookahead (value; BasicIO, BinIO, NJ93, SML90,
           TextIO)

map (value; Binarymap, CharVector, Gdbm, Intmap,
     List, ListPair, Option, Polygdbm, Polyhash,
     Splaymap, Vector, Word8Vector)
mapi (value; CharVector, Vector, Word8Vector)
mapPartial (value; List, Option)
Mar (constructor; Date)
Match (exception; General)
Math (structure)
max (value; Int, NJ93, Real, Word, Word8)
maxChar (value; Char)
maxInt (value; Int)
maxLen (value; Array, CharArray, CharVector,
        Vector, Weak, Word8Array, Word8Vector)
maxOrd (value; Char)
maxSize (value; String)
May (constructor; Date)
md5sum (value; Mosml)
member (value; Binaryset, Intset, Splayset)
Meta (structure)
min (value; Int, NJ93, Real, Word, Word8)
minChar (value; Char)
minInt (value; Int)
minute (value; Date)
mk_ppstream (value; PP)
mkAbsolute (value; Path)
mkCanonical (value; Path)
mkDict (value; Binarymap, Splaymap)
mkDir (value; FileSys)
mkLoc (value; Location)
mkPolyTable (value; Polyhash)
mkRelative (value; Path)
mkTable (value; Polyhash)
Mod (exception; SML90)
mod (value; General, Int, Word, Word8)
modify (value; Array, Array2, CharArray, Weak,
        Word8Array)
modifyi (value; Array, Array2, CharArray, Weak,
         Word8Array)
modTime (value; FileSys)
Mon (constructor; Date)
month (type; Date)
month (value; Date)
Mosml (structure)
Mosmlcgi (structure)

name (value; CommandLine)

```

nCols (value; Array2)
Neg (exception; SML90)
NEWDB (constructor; Gdbm)
newgen (value; Random)
newgenseed (value; Random)
nil (constructor; General)
nilLocation (value; Location)
NJ93 (structure)
Nonstdio (structure)
not (value; Bool)
notb (value; Word, Word8)
notContains (value; Char)
NotFound (exception; Binarymap, Binaryset, Gdbm,
Intmap, Intset, Polygdbm, Splaymap, Splayset)
NotWriter (exception; Gdbm, Polygdbm)
Nov (constructor; Date)
now (value; Time)
nRows (value; Array2)
Nth (exception; NJ93)
nth (value; List, NJ93)
NthTail (exception; NJ93)
nthtail (value; NJ93)
null (value; List)
numItems (value; Binarymap, Binaryset, Gdbm,
Intmap, Intset, Polygdbm, Polyhash, Splaymap,
Splayset)
o (value; General)
OCT (constructor; StringCvt)
Oct (constructor; Date)
offset (value; Date)
open_append (value; BasicIO, NJ93)
open_in (value; BasicIO, NJ93, SML90)
open_in_bin (value; BasicIO, NJ93, Nonstdio)
open_out (value; BasicIO, NJ93, SML90)
open_out_bin (value; BasicIO, NJ93, Nonstdio)
open_out_exe (value; Nonstdio)
openAppend (value; BinIO, TextIO)
openDir (value; FileSys)
openIn (value; BinIO, TextIO)
openmode (type; Gdbm)
openOut (value; BinIO, TextIO)
Option (exception; Option)
Option (structure)
orb (value; Word, Word8)
ord (value; Char, NJ93, SML90)
order (type; General)
ordof (value; NJ93)
OS (structure)
Out_of_memory (exception; General)
output (value; BasicIO, BinIO, NJ93, SML90, TextIO)
output1 (value; BinIO, TextIO)
output_binary_int (value; Nonstdio)
output_byte (value; Nonstdio)
output_char (value; Nonstdio)
output_value (value; Nonstdio)
outputc (value; BasicIO, NJ93)
outputSubstr (value; TextIO)
outstream (type; BasicIO, BinIO, NJ93, SML90,
TextIO)
Overflow (exception; General, Real)
packString (value; Byte)
padLeft (value; StringCvt)
padRight (value; StringCvt)
parentArc (value; Path)
ParseError (exception; Parsing)
parseTables (type; Parsing)
Parsing (structure)
part (type; Mosmlcgi)
part_data (value; Mosmlcgi)
part_field_integer (value; Mosmlcgi)
part_field_string (value; Mosmlcgi)
part_field_strings (value; Mosmlcgi)
part_fieldnames (value; Mosmlcgi)
part_type (value; Mosmlcgi)
partition (value; List)
Path (exception; Path)
Path (structure)
peek (value; Binarymap, Binaryset, Gdbm, Intmap,
Polygdbm, Polyhash, Splaymap, Splayset)
peekinsert (value; Polyhash)
peekVal (value; Parsing)
pi (value; Math)
Polygdbm (structure)
Polyhash (structure)
pos_in (value; Nonstdio)
pos_out (value; Nonstdio)
position (value; Substring)
pow (value; Math)
PP (structure)
pp_to_string (value; PP)
ppconsumer (type; PP)
precision (value; Int)
pred (value; Char)
print (value; BasicIO, NJ93, TextIO)
printDepth (value; Meta)
printLength (value; Meta)
printVal (value; Meta)
Process (structure)
Prod (exception; SML90)
quietdec (value; Meta)
quit (value; Meta)
Quot (exception; SML90)
quot (value; Int)
quotation (value; Meta)
QUOTE (constructor; General)
radix (type; StringCvt)
Random (structure)
random (value; Random)
randomlist (value; Random)
range (value; Random)
rangelist (value; Random)
readDir (value; FileSys)
READER (constructor; Gdbm)
reader (type; StringCvt)

readLink (value; FileSys)	slice (value; Substring)
Real (structure)	SML90 (structure)
real (type; General, Math, Real)	sort (value; Arraysort, Listsort)
real (value; General)	sorted (value; Arraysort, Listsort)
real_timer (type; Timer)	Span (exception; Substring)
realfmt (type; StringCvt)	span (value; Substring)
realPath (value; FileSys)	specialfiles (value; Help)
ref (constructor; General)	splay (type; Splaytree)
ref (type; General)	splay (value; Splaytree)
region (type; Array2)	Splaymap (structure)
rem (value; Int)	SplayNil (constructor; Splaytree)
remove (value; Binarymap, FileSys, Gdbm, Intmap, Polygdbm, Polyhash, Splaymap)	SplayObj (constructor; Splaytree)
rename (value; FileSys)	Splayset (structure)
reorganize (value; Gdbm, Polygdbm)	Splaytree (structure)
retrieve (value; Binaryset, Intmap, Splayset)	splitAt (value; Substring)
rev (value; List)	splitBaseExt (value; Path)
revapp (value; Binarymap, Binaryset, Intmap, Intset, NJ93, Splaymap, Splayset)	splitDirFile (value; Path)
revAppend (value; List)	splitl (value; StringCvt, Substring)
revfold (value; NJ93)	splitr (value; Substring)
rewindDir (value; FileSys)	sqrt (value; Math, NJ93, SML90)
rhsEnd (value; Parsing)	startCPUTimer (value; Timer)
rhsStart (value; Parsing)	startRealTimer (value; Timer)
rmDir (value; FileSys)	status (type; Process)
round (value; General, Real)	std_err (value; BasicIO, NJ93)
row (value; Array2)	std_in (value; BasicIO, NJ93, SML90)
RowMajor (constructor; Array2)	std_out (value; BasicIO, NJ93, SML90)
RTLD_LAZY (constructor; Dynlib)	stdErr (value; TextIO)
RTLD_NOW (constructor; Dynlib)	stdIn (value; TextIO)
run (value; Mosml)	stdOut (value; TextIO)
runresult (type; Mosml)	str (value; String)
	String (structure)
sameSign (value; Int, Real)	string (type; General, String)
Sat (constructor; Date)	string (value; Substring)
scan (value; Bool, Date, Int, Real, Time, Word, Word8)	StringCvt (structure)
scanStream (value; TextIO)	stringToBytes (value; Byte)
scanString (value; StringCvt)	sub (value; Array, Array2, CharArray, CharVector, Dynarray, String, Substring, Vector, Weak, Word8Array, Word8Vector)
SCI (constructor; StringCvt)	subArray (value; Dynarray)
second (value; Date)	Subscript (exception; General)
seek_in (value; Nonstdio)	Substring (exception; NJ93)
seek_out (value; Nonstdio)	Substring (structure)
Sep (constructor; Date)	substring (type; Substring)
set (type; Binaryset, Splayset)	substring (value; NJ93, String, Substring)
set (value; Weak)	succ (value; Char)
setLexAbsPos (value; Lexing)	Success (constructor; Mosml)
setLexCurrPos (value; Lexing)	success (value; Process)
setLexLastAction (value; Lexing)	Sum (exception; SML90)
setLexLastPos (value; Lexing)	Sun (constructor; Date)
setLexStartPos (value; Lexing)	Susp (structure)
setTime (value; FileSys)	susp (type; Susp)
sign (value; Int, Real)	symbolEnd (value; Parsing)
sin (value; Math, NJ93, SML90)	symbolStart (value; Parsing)
singleton (value; Binaryset, Intset, Splayset)	symHandle (type; Dynlib)
sinh (value; Math)	SysErr (exception; OS)
Size (exception; General)	syserror (type; OS)
size (value; String, Substring)	system (value; Meta, Process)
skipWS (value; StringCvt)	
	table (type; Gdbm, Polygdbm)

tabulate (value; Array, Array2, CharArray,
 CharVector, Dynarray, List, Vector,
 Word8Array, Word8Vector)
 take (value; List)
 takel (value; StringCvt, Substring)
 taker (value; Substring)
 tan (value; Math)
 tanh (value; Math)
 terminate (value; Process)
 TextIO (structure)
 Thu (constructor; Date)
 Time (exception; Time)
 Time (structure)
 time (type; Time)
 time (value; Mosml)
 Timer (structure)
 Tl (exception; NJ93)
 tl (value; List, NJ93)
 tmpName (value; FileSys)
 toCString (value; Char, String)
 toDefault (value; Real)
 toInt (value; Int, Word, Word8)
 toIntX (value; Word, Word8)
 tokens (value; String, Substring)
 toLarge (value; Int)
 toLargeInt (value; Word, Word8)
 toLargeIntX (value; Word, Word8)
 toLargeWord (value; Word, Word8)
 toLargeWordX (value; Word, Word8)
 toLower (value; Char)
 toMicroseconds (value; Time)
 toMilliseconds (value; Time)
 toReal (value; Time)
 toSeconds (value; Time)
 toString (value; Bool, Char, Date, Int, Path, Real,
 String, Time, Word, Word8)
 totalCPUTimer (value; Timer)
 totalRealTimer (value; Timer)
 toTime (value; Date)
 toUpper (value; Char)
 transform (value; Binarymap, Intmap, Polyhash,
 Splaymap)
 translate (value; String, Substring)
 traversal (type; Array2)
 triml (value; Substring)
 trimr (value; Substring)
 true (constructor; General)
 trunc (value; General, Real)
 truncate (value; NJ93)
 Tue (constructor; Date)

 union (value; Binaryset, Intset, Splayset)
 unit (type; General)
 unpackString (value; Byte)
 unpackStringVec (value; Byte)
 unzip (value; ListPair)
 update (value; Array, Array2, CharArray, Dynarray,
 Weak, Word8Array)
 use (value; Meta)

 validVolume (value; Path)
 valOf (value; Option)
 valuepoly (value; Meta)
 var (value; Dynlib)
 vecDouble (value; Mosml)
 vecFloat (value; Mosml)
 Vector (structure)
 vector (type; BinIO, CharArray, CharVector,
 TextIO, Vector, Word8Array, Word8Vector)
 vector (value; General)
 verbose (value; Meta)

 Weak (structure)
 weak (type; Weak)
 weak (value; Weak)
 Wed (constructor; Date)
 weekday (type; Date)
 weekDay (value; Date)
 welcome (value; Help)
 with_pp (value; PP)
 withtable (value; Gdbm, Polygdbm)
 withtables (value; Gdbm)
 Word (structure)
 word (type; Word, Word8)
 Word8 (structure)
 Word8Array (structure)
 Word8Vector (structure)
 wordSize (value; Word, Word8)
 WRCREAT (constructor; Gdbm)
 WRITER (constructor; Gdbm)

 xL (value; Location)
 xLR (value; Location)
 xorb (value; Word, Word8)
 xR (value; Location)
 xxLR (value; Location)
 xxRL (value; Location)

 year (value; Date)
 yearDay (value; Date)
 yyexit (exception; Parsing)
 yyparse (value; Parsing)

 zeroTime (value; Time)
 zip (value; ListPair)