Appendix A: Primitive Data Types and Operations

The implementation of the XE built-in library is based on the following data types and operations. These operations are of very low level and provide a means for cheating the compiler type checking. Thus they should not be used by the common user.

A.1 Primitive Type Generator _ATOM

The type generator `_atom` is meant only for implementing self-contained library types and should not be used for any other purposes. Types `_atom[`*n, true*`]` model bit vectors of n bytes of length complying signed integer arithmetics. Types `_atom[`*n, false*`]` model bit vectors of n bytes of length complying unsigned integer arithmetics. Objects of types generated by `_atom` are immutable.

There does **not** exist a type `_atom[`*n, s*`]` for each value of the `int` parameter n and the `bool` parameter s. The legal parameter values are implementation dependend. Typically types `_atom[1, false]` (a byte), `_atom[2, false]` (a word), `_atom[2, true]` (an int), and `_atom[4, true]` (longint) exist. Some of the operations may not be implemented for all types generated.

Objects of type `_atom[`*n, s*`]` use the integer literal form described in section 2.4 of the XE Reference Manual.

Operations

add = **proc**(a, b: _atom[n, s]) **returns**(_atom[n, s])

Description Integer addition. If arithmetics of the computer use the two's complement representation this can be used also for unsigned integers.

sub = **proc**(a, b: _atom[n, s]) **returns**(_atom[n, s])

Description Integer subtraction. If arithmetics of the computer use the two's complement representation this can be used also for unsigned integers.

mul = **proc**(a, b: _atom[n, s]) **returns**(_atom[n, s])

Description Integer multiplication.

div = **proc**(a, b: _atom[n, s]) **returns**(_atom[n, s])

Requires The value of b should never be zero. The value of a should not be `minint` if b is -1, so before invocation of this procedure the legality of the result should be ensured.

Description Integer division. Returns the integer quotient q of dividing a by b so that $\exists\ m : (0 \leq m <\mid b \mid) \wedge (a = b * q + m)$ holds.

mod = **proc**(a, b: _atom[n, s]) **returns**(_atom[n, s])

Requires The value of b should never be zero, so before invocation of this procedure the legality of the result should be ensured.

Description Integer modulo. Returns the integer remainder m of dividing a by b so that $\exists\ m : (0 \leq m <\mid b \mid) \wedge (a = b * q + m)$ holds.

bitand = **proc**(a, b: _atom[n, s]) **returns**(_atom[n, s])

Description Returns the `_atom[n, s]` where bits are set according to logical *and*, i.e. the i^{th} bit of the result is one only if the i^{th} bits of both a and b are ones.

bitor = **proc**(a, b: _atom[n, s]) **returns**(_atom[n, s])

Description Returns the `_atom[n, s]` where bits are set according to logical *or*, i.e. the i^{th} bit of the result is one only if at least one of the i^{th} bits of a and b is one.

bitnot = **proc**(a: _atom[n, s]) **returns**(_atom[n, s])

Description Returns the `_atom[n, s]` where bits are set according to logical *not*, i.e. the i^{th} bit of the result is zero only if the i^{th} bit of a is one.

lognot = **proc**(a: _atom[n, s]) **returns**(_atom[n, s])

Description Returns the logical not of the least significant bit.

gt = **proc**(a, b: _atom[n, s]) **returns**(bool)

Description Returns `true` if the signed integer value of a is greater than integer value of b.

ge = **proc**(a, b: _atom[n, s]) **returns**(bool)

Description Returns `true` if the signed integer value of a is greater than or equal to integer value of b.

lt = **proc**(a, b: _atom[n, s]) **returns**(bool)

Description Returns `true` if the signed integer value of a is less than integer value of b.

le = **proc**(a, b: _atom[n, s]) **returns**(bool)

Description Returns `true` if the signed integer value of a is less than or equal to integer value of b.

equal = **proc**(a, b: _atom[n, s]) **returns**(bool)

Description Returns `true` if a = b.

rotate = **proc**(a: _atom[n, s], i: int) **returns**(_atom[n, s])

Description Returns the `_atom[n, s]` where the bits of a are rotated left i positions, negative values of i rotate the bits right and if $i = 0$ then a is returned.

shift = **proc**(a: _atom[n, s], i: int) **returns**(_atom[n, s])

Description Returns the `_atom[n, s]` where the bits of a are shifted left i positions, negative values of i shift the bits right and if $i = 0$ then a is returned.

A.2 Primitive Type Generator _FIELDNAME

The type generator `fieldname` is meant only for implementing library types and should not be used for any other purposes. Objects of type generated by `_fieldname` model names written in field lists. Objects of these types are immutable.

For every field list F = $\{N_1 : T_1, ..., N_n : T : T_n\}$ there exists a type `_fieldname{F}`. The order of the fields is significant, i.e. for each permutation there exists a distinct type.

Operations

There is no constructor because objects can be created by applying the `_make` low level routine to `tag` values created by the compiler (`tag` and `_fieldname` have the same representation).

parse = **proc**(s: string) **returns**(_fieldname{F}, bool)

Description Returns the name corresponding to the given string. The value of the second result is `true` if s is not any of the field names.

unparse = **proc**(f: _fieldname{F}) **returns**(string)

Description Returns a string representing name f.

count = **proc**() **returns**(int)

Description Returns the total number of fields (i.e., the n).

uniquetypecode = **proc**(i: int) **returns**(int)

Requires i must be a legal field index.
Description Returns the type code of the field i unique with respect to other fields. Unique means that if there is the same type in some other field, a new typecode is created to avoid conflicts.

index = **proc**(i: int) **returns**(int)

Requires i must be a legal unique type code.
Description Returns the field index of given a unique type code.

offset = **proc**(i: int) **returns**(int)

Requires i must be a legal field index of a nonvoid field.
Description Returns the offset (for records) of the given field.

size = **proc**() **returns**(word)

Description Returns allocation size of a record in allocation units.

A.3 Primitive Operations of XE

These operations are meant only for implementing library types of XE and should not be used for any other purposes. The routines are not proper XE-routines because they can directly manipulate memory and thus the representation of any XE-object, so the strict typing rules are ignored by these routines.

The objects of XE that are not self-contained are manipuled using pointers pointing to the first byte of the object in memory. All operations require that the memory references caused by the argument values are legal.

_get_field = **proc**[t1, t2: type](b: t1, i: int) **returns**(t2)

Requires b must be an object (handle) returned by _alloc.
Description Gets an object (or pointer) of type $t2$ from location $addr(o) + i$.

__get_field = **proc**[t1, t2: type](b: t1, i: int) **returns**(t2)

Requires b must be an object (handle) returned by _alloc.
Description Gets an object (or pointer) of type $t2$ from location $addr(o) + i$. This is a low level version of `_get_field`.

_set_field = **proc**[t1, t2: type](b: t1, i: int, e: t2))

Requires b must be object (handle) returned by _alloc.
Description Places an object (or pointer) of type $t2$ to location $addr(o) + i$.

__set_field = **proc**[t1, t2: type](b: t1, i: int, e: t2))

Requires b must be object (handle) returned by _alloc.
Description Places an object (or pointer) of type $t2$ to location $addr(o) + i$. This is a low level version of `_set_field`.

_alloc = **proc**[t: type](n: word) **returns**(t)

Description Allocates a block of n units of memory and returns the address of the block. Note that via the handle r returned you should only manipulate the bytes $addr(r), ..., addr(r) + u * (n-1)$, where u is the unit size in bytes.

_referencesize = **proc**[t: type]() **returns**(int)

Description Returns the size of the pointer or the size of the self-contained object of type t.

_maxreferencesize = **proc**() **returns**(int)

Description Maximum size of pointer or self contained object for all types. Returns a constant in an implementation.

_nullpointer = **proc**[t: type]() **returns**(t)

Description Returns a pointer to a constant address never used by _alloc.

_make = **proc**[t1, t2: type](o: t1) **returns**(t2)

Description Converts from an arbitary type to another. This is a compile time operation, it does not take any running time because it leaves the representation untouched. Can be used also for address arithmetic.

_equal = **proc**[t1, t2: type](o1: t2, o2: t2)

Requires t1 and t2 must be pointer types.
Description returns **true** if o1 and o2 point to the same location.

_atom_pad = **proc**[m1: int, t1: bool, m2: int, t2: bool](a: _atom[m1, t1]) **returns**(_atom[m2, t2])

Description If $m2 > m1$, performs padding without sign extension, i.e. integer values of a and the result are not necessarily equal. If $m2 < m1$, performs truncating unpadding, no signal is raised even if integer values of a and the result are not equal. If $m = n$ then a is returned.

A.4 Primitive Type _STREAM

_stream contains primitive operations for simple file I/O.

Operations

create = **proc**(i: int) **returns**(_stream)

Description Returns a _stream corresponding to given i (Typically a file handle id of the operation system).

open = **proc**(s: string, modenr: int) **returns**(_stream)

Description Opens a new stream to the file s (*modenr*: 0 = read, 1 = write, 2 = append). Returns the *invalid stream* if open fails.

close = **proc**(s: _stream)

> **Description** Closes *s*.

invalid = **proc**(s: _stream) **returns**(bool)

> **Description** Returns **true** if *s* is the *invalid stream*.

seek = **proc**(s: _stream, position: int)

> **Description** Seeks the given *position* in *s*.

write = **proc**(s: _stream, buffer: array[char], first, count: int) **returns**(int)

> **Description** Writes characters *buffer[first], ..., buffer[first + count - 1]* to _stream *s*. Returns the number of characters written.

read = **proc**(s: _stream, buffer: array[char], first, count: int) **returns**(int)

> **Description** Reads *count* chars and stores them to buffer.