27. List pseudo-methods library

This clause describes the pseudo-methods used to work with lists.

27.1 Pseudo-methods overview

A *pseudo-method* is a type of method unique to the ***e*** language. Pseudo-methods are ***e*** macros that look like methods. They have the following characteristics:

- Unlike methods, pseudo-methods are not restricted to structs.
- They can be applied to any expression, including literal values, scalars, and compound arithmetic expressions.
- Pseudo-methods cannot be extended.
- Pseudo-methods are defined by using **define as** (see 16.2).
- List pseudo-methods are associated with list data types, as opposed to being within the scope of a struct.

If a method is added that uses the same name as one of the pseudo-methods for a built-in struct, that user-defined method shall take precedence over the built-in struct.

See also 4.10.5, 4.15, 5.1, and Clause 28.

27.2 Using list pseudo-methods

A list pseudo-method can be used to operate on a (previously declared) list field or variable by attaching the pseudo-method name, preceded by a period (`.`), to the list name. Any parameters required by the pseudo-method go in parentheses [`( )`] after the pseudo-method name.

Example

The following calls the **apply()** pseudo-method for the list named `p_list`, with the expression `.length + 2` as a parameter. The pseudo-method returns a list of numbers found by adding `2` to the `length` field value in each item in the list.

```
n_list = p_list.apply(.length + 2)
```

Many list pseudo-methods take expressions as parameters and operate on every item in the list. In those pseudo-methods, the **it** variable can be used in an expression to refer to the current list item, and the **index** variable can be used to refer to the current item's list index number.

Pseudo-methods that return values can only be used in expressions.

27.3 Pseudo-methods to modify lists

This subclause describes the pseudo-methods that change one or more items in a list.

See also 4.16.2, 10.5.1, 20.1.2, 28.4.1, and 29.1.1.

27.3.1 add(item)

Purpose	Add an item to the end of a list	
Category	Pseudo-method	
Syntax	*list*.**add(***item*: list-item-type**)**	
Parameters	*list*	A list.
	item	An item of list-item type, which is to be added to the list. The item is added at index `list.size()`, e.g., if the list contains five items, the last item is at index `list.size()-1` or `4`. Adding an item to this list places it at index `5`.
Return value	None	

This adds the *item* to the end of the *list*. If the item is a struct, no new struct instance is generated; a pointer to the existing instance of the struct is simply added to the list.

Syntax example:

```
var i_list : list of int;
i_list.add(5)
```

27.3.2 add(list)

Purpose	Add a list to the end of another list	
Category	Pseudo-method	
Syntax	*list_1*.**add(***list_2*: list**)**	
Parameters	*list_1*	A list.
	list_2	An item of the same type as *list_1*, which is to be added to the end of *list_1*. The list is added at index `list.size()`, e.g., if the list contains five items, the last item is at index `list.size()-1` or `4`. Adding an item to this list places it at index `5`.
Return value	None	

This adds a copy of *list_2* to the end of *list_1*.

Syntax example:

```
i_list.add(l_list)
```

27.3.3 add0(item)

Purpose	Add an item to the head of a list	
Category	Pseudo-method	
Syntax	*list*.**add0(***item*: list-type**)**	
Parameters	*list*	A list.
	item	An item of the same type as the list items, which is to be added to the head of the list.
Return value	None	

This adds a new item to an existing list. The item is placed at the head of the existing list, as the first position (that is, at index `0`). All subsequent items are then reindexed by incrementing their old index by `1`.

If the item is a struct, no new struct instance is generated: a pointer to the existing instance of the struct is simply added to the list.

Syntax example:

```
var l_list : list of int = {4; 6; 8};
l_list.add0(2)
```

27.3.4 add0(list)

Purpose	Add a list to the head of another list	
Category	Pseudo-method	
Syntax	*list_1*.**add0(***list_2*: list**)**	
Parameters	*list_1*	A list.
	list_2	An item of the same type as *list_1*, which is to be added to the end of *list_1* (at *list_1* index `0`).
Return value	None	

This adds a new list to an existing list. A copy of the *list_2* list is placed at the head of the existing *list_1* list, starting at the first *list_1* index. All subsequent items are then reindexed by incrementing their old index by the size of the new list being added.

Syntax example:

```
var i_list : list of int = {1; 3; 5};
var l_list : list of int = {2; 4; 6};
i_list.add0(l_list)
```

27.3.5 clear()

Purpose	Delete all items from a list	
Category	Pseudo-method	
Syntax	*list*.**clear()**	
Parameters	*list*	A list.
Return value	None	

This deletes all items in the list.

27.3.6 delete()

Purpose	Delete an item from a list	
Category	Pseudo-method	
Syntax	*list*.**delete(***index*: int**)**	
Parameters	*list*	A list.
	index	The index of the item to delete from the list.
Return value	None	

This removes item number *index* from *list* (indexes start counting from `0`). The indexes of the remaining items are adjusted to keep the numbering sequential. If the index does not exist in the list, an error shall be issued.

NOTE—*list*.**delete()** cannot be used to delete a range of items (in a single call).

Syntax example:

```
var i_list : list of int = {2; 4; 6; 8};
i_list.delete(2)
```

27.3.7 fast_delete()

Purpose	Delete an item without adjusting all indexes	
Category	Pseudo-method	
Syntax	*list*.**fast_delete(***index*: int**)**	
Parameters	*list*	A list.
	index	The index of the item to delete from the list.
Return value	None	

This removes item number *index* from *list* (indexes start counting from `0`). The index of the last item in the list is changed to the index of the item that was deleted, so all items following the deleted item keep their original indexes, except the original last index is removed. If the index does not exist in the list, an error shall be issued.

Syntax example:

```
var l_list : list of int = {2; 4; 6; 8};
l_list.fast_delete(2)
```

27.3.8 insert(index, item)

Purpose	Insert an item in a list at a specified index	
Category	Pseudo-method	
Syntax	*list*.**insert(***index*: int**,** *item*: list-type**)**	
Parameters	*list*	A list.
	index	The index in the *list* where the *item* is to be inserted.
	item	An *item* of the same type as the *list*.
Return value	None	

This inserts the *item* at the *index* location in the *list*. If *index* is the size of the list, then the *item* is simply added at the end of the list. All indexes in the list are adjusted to keep the numbering correct. If the number of items in the list is smaller than *index*, an error shall be issued.

If the item is a struct, no new struct instance is generated: a pointer to the existing instance of the struct is simply added to the list.

Syntax example:

```
var l_list := {10; 20; 30; 40; 50};
l_list.insert(3, 99)
```

27.3.9 insert(index, list)

Purpose	Insert a list in another list starting at a specified index	
Category	Pseudo-method	
Syntax	*list_1***.insert(***index*: int**,** *list_2*: list**)**	
Parameters	*list_1*	A list.
	index	The index of the position in *list_1* where *list_2* is to be inserted.
	list_2	The list to insert into *list_1*.
Return value	None	

This inserts all items of *list_2* into *list_1* starting at *index*. The *index* shall be a positive integer. The size of the new list size is equal to the sum of the sizes of *list_1* and *list_2*. If the number of items in *list_1* is smaller than *index*, an error shall be issued.

Syntax example:

```
var l_list := {10; 20; 30; 40; 50};
var m_list := {11; 12; 13};
l_list.insert(1, m_list)
```

27.3.10 pop()

Purpose	Remove and return the last list item	
Category	Pseudo-method	
Syntax	*list***.pop()**: list-type	
Parameters	*list*	A list.
Return value	The last item	

This removes the last item [the item at index `list.size() - 1`] in the *list* and returns it. If the list is empty, an error shall be issued.

NOTE—*list***.top()** can be used to return the last item in *list* without removing it from the list (see 27.4.26).

Syntax example:

```
var i_list := {10; 20; 30};
var i_item : int;
i_item = i_list.pop()
```

27.3.11 pop0()

Purpose	Remove and return the first list item
Category	Pseudo-method
Syntax	*list*.**pop0()**: list-type
Parameters	*list* A list.
Return value	The first item

If the *list* is empty, this method shall issue an error. Otherwise, it removes the first item (the item at index `0`) from the *list* and returns that item. It then subtracts `1` from the index of each item remaining in the list.

NOTE—*list*.**top0()** can be used to return the first item in *list* without removing it from the list (see 27.4.27).

Syntax example:

```
var i_list := {10; 20; 30};
var i_item : int;
i_item = i_list.pop0()
```

27.3.12 push()

Purpose	Add an item to the end of a list [same as add(item)]	
Category	Pseudo-method	
Syntax	*list*.**push(***item*: list-type**)**	
Parameters	*list*	A list.
	item	An item of the same type as the *list* type, which is to be added to the list. The item is added at index `list.size()`, e.g., if the list contains five items, the last item is at index `list.size()-1` or `4`. Adding an item to this list places it at index `5`.
Return value	None	

This pseudo-method performs the same function as **add(item)** (see 27.3.1). If the item is a struct, no new struct instance is generated; a pointer to the existing instance of the struct is simply added to the list.

Syntax example:

```
var i_list : list of int;
i_list.push(5)
```

27.3.13 push0()

Purpose	Add an item to the head of a list [same as add0(item)]	
Category	Pseudo-method	
Syntax	*list*.**push0(***item*: list-type**)**	
Parameters	*list*	A list.
	item	An item of the same type as the list items, which is to be added to the head of the list.
Return value	None	

This pseudo-method performs the same function as **add0(item)** (see 27.3.3). If the item is a struct, no new struct instance is generated; a pointer to the existing instance of the struct is simply added to the list.

Syntax example:

```
var l_list : list of int = {4; 6; 8};
l_list.push0(2)
```

27.3.14 push(list)

Purpose	Add a list to the end of another list [same as add(item)]	
Category	Pseudo-method	
Syntax	*list_1*.**push(***list_2*: list**)**	
Parameters	*list_1*	A list.
	list_2	An item of the same type as *list_1*, which is to be added to the end of *list_1*. The list is added at index `list.size()`, e.g., if the list contains five items, the last item is at index `list.size()-1` or `4`. Adding an item to this list places it at index `5`.
Return value	None	

This pseudo-method performs the same function as **add(list)** (see 27.3.2); it adds *list_2* to the end of *list_1*.

Syntax example:

```
i_list.push(l_list)
```

27.3.15 push0(list)

Purpose	Add a list to the head of another list [same as add0(list)]	
Category	Pseudo-method	
Syntax	*list_1*.**push0(***list_2*: list**)**	
Parameters	*list_1*	A list.
	list_2	An item of the same type as *list_1*, which is to be added to the end of *list_1* (at *list_1* index `0`).
Return value	None	

This pseudo-method performs the same function as **add0(list)** (see 27.3.4); it adds a new list to an existing list. The *list_2* list is placed at the head of the existing *list_1* list, starting at the first *list_1* index. All subsequent items are then reindexed by incrementing their old index by the size of the new list being added.

Syntax example:

```
var i_list : list of int = {1; 3; 5};
var l_list : list of int = {2; 4; 6};
i_list.push0(l_list)
```

27.3.16 resize()

Purpose	Change the size of a list	
Category	Pseudo-method	
Syntax	*list*.**resize(***size*: int [**,** *full*: bool**,** *filler*: exp**,** *keep_old*: bool]**)**	
Parameters	*list*	A list.
	size	A positive integer specifying the desired size.
	full	A Boolean value specifying all items are to be filled with filler (defaults to TRUE).
	filler	An item of the same type of the list items; used as a filler when *full* is TRUE.
	keep_old	A Boolean value specifying whether to keep existing items already in the list (defaults to FALSE).
Return value	None	

This clears the list and increases or decreases the list size according to the new size.

— If only the second parameter, *size*, is used, this method allocates a new list of the given size and all items are initialized to the default value for the list type.

— If any of the three parameters after *size* are used, all three of them shall be used.

— If *full* is TRUE, this method sets all new items to have *filler* as their value.

To resize a list and keep its old values, set both *full* and *keep_old* to TRUE. If the list is made longer, additional items with the value of *filler* are appended to the list. The following details the behavior of this method for all combinations of *full* and *keep_old*:

a) *full* is FALSE, *keep_old* is FALSE

An empty list (that is, a list of zero size) is created and memory is allocated for a list of the given *size*.

b) *full* is TRUE, *keep_old* is FALSE

The list is resized to *size* and filled completely with *filler*.

c) *full* is FALSE, *keep_old* is TRUE

1) If *size* is greater than the size of the existing list, the list is enlarged to the new *size*, and the new positions are filled with the default value of the list type.

2) If *size* is less than or equal to the size of the existing list, the list is shortened to the new *size*, and all of the existing values up to that size are retained.

d) *full* is TRUE, *keep_old* is TRUE

1) If *size* is greater than the size of the existing list, the list is enlarged to the new *size* and the new positions are filled with *filler*.

2) If *size* is less than or equal to the size of the existing list, the list is shortened to the new *size* and all of the existing values up to that size are retained.

Syntax example:

```
var r_list := {2; 3; 5; 6; 8; 9};
r_list.resize(10, TRUE, 1, TRUE)
```

27.4 General list pseudo-methods

This subclause describes the syntax for pseudo-methods that perform various operations on lists.

27.4.1 all_different()

Purpose	Returns TRUE if evaluation of the expression returns a unique value for each of the list elements	
Category	Pseudo-method	
Syntax	*list*.**all_different(***item*: exp [, *bool_exp*: bool]**)**: bool	
Parameters	*list*	A list.
	item	Any expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
	bool_exp	Any Boolean expression. Optional. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number. If not given in the method call, defaults to TRUE
Return value	A Boolean value	

Returns TRUE if, and only if, evaluation of the expression returns a unique value for each of the list elements, except (if *bool_exp* is specified) those elements for which *bool_exp* evaluates to FALSE. In other words, no two items (or expressions) in the list for which the *bool_exp* is TRUE (which is the default if no *bool_exp* is specified) have the same value.

Syntax example:

```
struct packet {
   x: byte;
   y: byte;
};

extend sys {
   packets: list of packet;
   keep packets.all_different(.x+.y);
   L:list of uint;
   keep L.all_different(it);
};

var l: list of int = {UNDEF,3,2,1,UNDEF,4,UNDEF,6};
print l.all_different(it , it != UNDEF);
```

Prints TRUE because all the elements that are different from UNDEF are also different from each other.

27.4.2 apply()

Purpose	Perform a computation on each item in a list	
Category	Pseudo-method	
Syntax	*list*.**apply(***expr*: exp**)**: list	
Parameters	*list*	A list.
	expr	Any expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The changed list	

This applies the *expr* to each item in the *list* and returns the changed list. The expression *list***.apply(it.***field***)** is the same as *list***.***field* when *field* is a scalar type. The two expressions are different, however, if the field is not a scalar.

Example

Assuming `data` is a list of byte, the first expression returns a list containing the first byte of `data` of each packet item. The second expression is a single item, which is the first item in the concatenated list of all `data` fields in all packet items.

```
packets.apply(it.data[0]);
packets.data[0]
```

Syntax example:

```
var p_list := {1; 3; 5};
var n_list : list of int;
n_list = p_list.apply(it * 2)
```

27.4.3 copy()

Purpose	Make a shallow copy of a list	
Category	Predefined method of any struct or unit	
Syntax	*list*.**copy()**: list	
Parameters	*list*	A list.
Return value	None	

This is a specific case of *exp*.**copy()** (see 28.4.1), where *exp* is the name of a *list*.

Syntax example:

```
var strlist_1 : list of string = {"A"; "B"; "C"};
var strlist_2 : list of string;
strlist_2 = strlist_1.copy()
```

27.4.4 count()

Purpose	Return the number of items that satisfy a given condition	
Category	Pseudo-method	
Syntax	*list*.**count(***exp*: bool**)**: int	
Parameters	*list*	A list.
	exp	A Boolean expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The number of items	

This returns the number of items for which the *exp* is TRUE.

Syntax example:

```
var ct : int;
ct = instr_list.count(it.op1 > 200)
```

27.4.5 exists()

Purpose	Check if an index exists in a list	
Category	Pseudo-method	
Syntax	*list*.**exists(***index*: int**)**: bool	
Parameters	*list*	A list.
	index	An integer expression representing an index to the list.
Return value	A Boolean value	

This returns TRUE if an item with the *index* number exists in the *list* or returns FALSE if the index does not exist.

Syntax example:

```
var i_chk : bool;
i_chk = packets.exists(5)
```

27.4.6 first()

Purpose	Get the first item that satisfies a given condition	
Category	Pseudo-method	
Syntax	*list*.**first(***exp*: bool**)**: list-type	
Parameters	*list*	A list.
	exp	A Boolean expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The first matching item	

This returns the first item for which *exp* is TRUE and stops executing.

If there is no such item, the default for the item's type is returned (see 5.1). For a list of scalars, a value of zero (0) is returned if there is no such item. Since zero (0) might be confused with a value found, it is safer to use *list*.**first_index()** for lists of scalars.

Syntax example:

```
var i_item : instr;
i_item = instr_list.first(it.op1 > 15)
```

27.4.7 first_index()

Purpose	Get the index of the first item that satisfies a given condition	
Category	Pseudo-method	
Syntax	*list*.**first_index(***exp*: bool**)**: int	
Parameters	*list*	A list.
	exp	A Boolean expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The index of the first matching item	

This returns the index of the first item for which *exp* is `TRUE` and stops executing. Otherwise, it returns `UNDEF` (if there is no such item).

Syntax example:

```
var i_item : int;
i_item = instr_list.first_index(it.op1 > 15)
```

27.4.8 flatten()

Purpose	Get a list of the base elements that make up the sub-lists in a multi-dimensional list	
Category	Pseudo-method	
Syntax	*list*.**flatten()**: *list*	
Parameters	*list*	A list.
Return value	A list	

Returns a regular (one-dimensional) list that contains all the base elements that are contained in the list. If the multidimensional list is a keyed list, a regular list is still returned.

Syntax example:

Generates a list containing the numbers 1 to 6, with the number 4 twice:

```
var matrix: list of list of int = {{1;2;3;4};{4;5;6}};
var l: list of int = matrix.flatten();
```

27.4.9 get_indices()

Purpose	Return a sublist of the targeted list	
Category	Pseudo-method	
Syntax	*list*.**get_indices(***index-list*: list of int**)**: list-type	
Parameters	*list*	A list.
	index-list	A list of indexes within the list. Each index needs to exist in the list.
Return value	A new list	

This copies the items in *list* that have the indexes specified in *index-list* and returns a new list containing those items. If the *index-list* is empty, an empty list is returned.

Syntax example:

```
var i_list : list of packet;
i_list = packets.get_indices({0; 1; 2})
```

27.4.10 has()

Purpose	Check that a list has at least one item that satisfies a given condition	
Category	Pseudo-method	
Syntax	*list*.**has(***exp*: bool**)**: bool	
Parameters	*list*	A list.
	exp	A Boolean expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	A Boolean value	

This returns TRUE if the *list* contains at least one item for which the *exp* is TRUE. Otherwise, it returns FALSE (if the *exp* is not TRUE for any item).

Syntax example:

```
var i_ck : bool;
i_ck = sys.instr_list.has(it.op1 > 31)
```

27.4.11 is_a_permutation()

Purpose	Check that two lists contain exactly the same items
Category	Pseudo-method
Syntax	*list_1*.**is_a_permutation(***list_2*: list**)**: bool
Parameters	*list_1* A list.
	list_2 An list to compare to *list_1*. This shall be a convertible type of *list_1*.
Return value	A Boolean value

This returns TRUE if *list_2* contains the same items as *list_1*; otherwise, it returns FALSE (if any items in one list are not in the other list).

— The order of the items in the two lists does not need to be the same, but the number of items shall be the same for both lists, i.e., items that are repeated in one list shall appear the same number of times in the other list.

— If the lists are lists of structs, *list_1***.is_a_permutation(***list_2***)** compares the addresses of the struct items, not their contents.

— A *convertible type* is one that automatically converts to match the relevant type.

NOTE—This pseudo-method can be used in a **keep** constraint to fill *list_1* with the same items contained in the *list_2*, although not necessarily in the same order.

Syntax example:

```
var lc : bool;
lc = packets_1a.is_a_permutation(packets_1b)
```

27.4.12 is_empty()

Purpose	Check if a list is empty
Category	Pseudo-method
Syntax	*list*.**is_empty()**: bool
Parameters	*list* A list.
Return value	A Boolean value

This returns TRUE if *list* is empty; otherwise, it returns FALSE (if the list is not empty).

Syntax example:

```
var no_l : bool;
no_l = packets.is_empty()
```

27.4.13 last()

Purpose	Get the last item that satisfies a given condition	
Category	Pseudo-method	
Syntax	*list*.**last(***exp*: bool**)**: list-type	
Parameters	*list*	A list.
	exp	A Boolean expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The last matching item	

This returns the first item for which *exp* is TRUE and stops executing.

If there is no such item, the default for the item's type is returned (see 5.1). For a list of scalars, a value of zero (0) is returned if there is no such item. Since zero (0) might be confused with a value found, it is safer to use *list*.**last_index()** for lists of scalars.

Syntax example:

```
var i_item : instr;
i_item = sys.instr_list.last(it.op1 > 15)
```

27.4.14 last_index()

Purpose	Get the index of the last item that satisfies a given condition	
Category	Pseudo-method	
Syntax	*list*.**last_index(***exp*: bool**)**: int	
Parameters	*list*	A list.
	exp	A Boolean expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The index of the last matching item	

This returns the index of the last item for which *exp* is TRUE and stops executing; otherwise, it returns UNDEF (if there is no such item).

Syntax example:

```
var i_item : int;
i_item = instr_list.last_index(it.op1 > 15)
```

27.4.15 max()

Purpose	Get the item with the maximum value of a given expression	
Category	Pseudo-method	
Syntax	*list*.**max(***exp*: numeric-type**)**: list-type	
Parameters	*list*	A list.
	exp	An integer expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The matching item	

This returns the item for which the *exp* evaluates to the largest value. If more than one item results in the same maximum value, the item latest in the list is returned. If the *list* is empty, an error shall be issued.

Syntax example:

```
var high_item : item_instance;
high_item = item_list.max(it.f_1 + it.f_2)
```

27.4.16 max_index()

Purpose	Get the index of the item with the maximum value of a given expression	
Category	Pseudo-method	
Syntax	*list*.**max_index(***exp*: numeric-type**)**: int	
Parameters	*list*	A list.
	exp	An integer expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The index of the matching item	

This returns the index of the item for which the *exp* evaluates to the largest value. If more than one item results in the same maximum value, the index of item latest in the list is returned. If the *list* is empty, an error shall be issued.

Syntax example:

```
var item_index : index;
item_index = sys.item_list.max_index(it.f_1 + it.f_2)
```

27.4.17 max_value()

Purpose	Return the maximum value found by evaluating a given expression for all items	
Category	Pseudo-method	
Syntax	*list*.**max_value(***exp*: numeric-type**)**: exp-type	
Parameters	*list*	A list.
	exp	An integer expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The matching value	

This returns the largest integer value found by evaluating the *exp* for every item in the list.

For lists of integer types, Table 39 shows what is returned when the *list* is empty.

Table 39—Empty list max_value() return values

List item type	Value returned
Signed integer	`MIN_INT` (see 4.1.4.4)
Unsigned integer	zero (`0`)
Long integer	error

Syntax example:

```
var item_val : int;
item_val = sys.item_list.max_value(it.f_1 + it.f_2)
```

27.4.18 min()

Purpose	Get the item with the minimum value of a given expression	
Category	Pseudo-method	
Syntax	*list*.**min(***exp*: numeric-type**)**: list-type	
Parameters	*list*	A list.
	exp	An integer expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The matching item	

This returns the item for which the *exp* evaluates to the smallest value. If more than one item results in the same minimum value, the item latest in the list is returned. If the *list* is empty, an error shall be issued.

Syntax example:

```
var low_item : item_instance;
low_item = sys.item_list.min(it.f_1 + it.f_2)
```

27.4.19 min_index()

Purpose	Get the index of the item with the minimum value of a given expression	
Category	Pseudo-method	
Syntax	*list*.**min_index(***exp*: numeric-type**)**: int	
Parameters	*list*	A list.
	exp	An integer expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The index of the matching item	

This returns the index of the item for which the specified *exp* gives the minimal value. If more than one item results in the same minimum value, the index of the item latest in the list is returned. If the *list* is empty, an error shall be issued.

Syntax example:

```
var item_index : index;
item_index = sys.item_list.min_index(it.f_1 + it.f_2)
```

27.4.20 min_value()

Purpose	Return the minimum value found by evaluating a given expression for all items	
Category	Pseudo-method	
Syntax	*list*.**min_value(***exp*: numeric-type**)**: exp-type	
Parameters	*list*	A list.
	exp	An integer expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The matching value	

This returns the smallest integer value found by evaluating the *exp* for every item in the list.

For lists of integer types, Table 39 shows what is returned when the *list* is empty.

Syntax example:

```
var item_val : int;
item_val = sys.item_list.min_value(it.f_1 + it.f_2)
```

27.4.21 reverse()

Purpose	Reverse the order of a list
Category	Pseudo-method
Syntax	*list*.**reverse()**: list
Parameters	*list* A list.
Return value	The changed list

This returns a new list of all the items in *list* in reverse order.

Syntax example:

```
var s_list := {"A"; "B"; "C"; "D"};
var r_list := s_list.reverse()
```

27.4.22 size()

Purpose	Return the size of a list
Category	Pseudo-method
Syntax	*list*.**size()**: int
Parameters	*list* A list.
Return value	The list size

This returns an integer equal to the number of items in the *list*. See 10.2.7.3 for more information about constraining the size of lists. See also 4.12.1 and 10.4.1.

NOTE—To control the list size, use a construct like **keep** *list*.**size() ==** *n*, where *n* is an integer expression. Another way to specify an exact size of a list is by using the *list*[*n*] index syntax in the list declaration, such as `p_list[n]: list of p`.

Syntax example:

```
print packets.size()
```

27.4.23 sort()

Purpose	Sort a list	
Category	Pseudo-method	
Syntax	*list*.**sort(***sort-exp*: exp**)**: list	
Parameters	*list*	A list of integers, strings, enumerated items, or Boolean values to sort.
	sort-exp	A scalar or nonscalar expression. The expression can contain references to fields or structs. The **it** variable can be used to refer to the current list item.
Return value	The changed list	

This returns a new list of all the items in *list*, sorted in increasing order of the values of the *sort-exp*. If the *sort-exp* is a scalar (or string) value, the list is sorted by value. If the *sort-exp* is a nonscalar, the list is sorted by address.

Syntax example:

```
var s_list : list of packet;
s_list = packets.sort(it.f_1 + it.f_2)
```

27.4.24 sort_by_field()

Purpose	Sort a list of structs by a selected field	
Category	Pseudo-method	
Syntax	*struct-list*.**sort_by_field(***field*: field-name**)**: list	
Parameters	*list*	A list of structs.
	field	The name of a field of the list's struct type. Enter the name of the field only, without a preceding period (`.`) or the term `it`.
Return value	The changed list	

This returns a new list of all the items in *struct-list*, sorted in increasing order of their *field* values.

NOTE—The *list*.**sort()** pseudo-method returns the same value as the *list*.**sort_by_field()** pseudo-method, but *list*.**sort_by_field()** is more efficient.

Syntax example:

```
var s_list : list of packet;
s_list = sys.packets.sort_by_field(length)
```

27.4.25 split()

Purpose	Splits a list at each point where an expression is TRUE	
Category	Pseudo-method	
Syntax	*list*.**split(***split-exp*: exp**)**: list of struct-list-holder	
Parameters	*list*	A list (of any type).
	split-exp	An expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The list of struct-list-holder	

Since ***e*** does not support lists of lists, this pseudo-method returns a list of type *struct-list-holder*.

— The *struct-list-holder* type is a struct with a single field, `value: list of any-struct`.

— A *struct-list-holder* is a list of structs, with each struct containing a list of items of the original *list* type.

— Each *struct-list-holder* in the returned list contains consecutive items from the *list* that have the same *split-exp* value.

Any fields used in the expression shall be defined in the base type definition, not in **when** subtypes.

Syntax example:

```
var sl_hold := s_list.split(it.f_1 == 16)
```

27.4.26 top()

Purpose	Return the last item in a list	
Category	Pseudo-method	
Syntax	*list*.**top()**: list-item	
Parameters	*list*	A list.
Return value	The last item	

This returns the last item in the *list* without removing it from the list. If the list is empty, an error shall be issued.

Syntax example:

```
var pk : packet;
pk = sys.packets.top()
```

27.4.27 top0()

Purpose	Return the first item in a list
Category	Pseudo-method
Syntax	*list*.**top0()**: list-item
Parameters	*list* A list.
Return value	The first item

This returns the first item in the *list* without removing it from the list. If the list is empty, an error shall be issued.

NOTE—This pseudo-method can be used with **pop0( )** to emulate queues.

Syntax example:

```
var pk : packet;
pk = sys.packets.top0()
```

27.4.28 unique()

Purpose	Collapse consecutive items that have the same value into one item	
Category	Pseudo-method	
Syntax	*list*.**unique(***select-exp*: exp**)**: list	
Parameters	*list*	A list of type *struct-list-holder.*
	split-exp	An expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The changed list	

This returns a new list of all the distinct values in *list*. In the new list, all consecutive occurrences of items for which the value of *exp* are the same are collapsed into one item.

Syntax example:

```
var u_list : list of l_item;
u_list = sys.l_list.unique(it.f_1)
```

27.4.29 all()

Purpose	Get all items that satisfy a condition	
Category	Pseudo-method	
Syntax	*list*.**all(***exp*: bool**)**: list	
Parameters	*list*	A list.
	exp	A Boolean expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	A list of the matching items	

This returns a list of all the items in *list* for which *exp* is TRUE. If no items satisfy the Boolean expression, an empty list is returned. See also 4.16.1.

Syntax example:

```
var l_2 : list of packet;
l_2 = sys.packets.all(it.length > 64)
```

27.4.30 all_indices()

Purpose	Get indexes of all items that satisfy a condition	
Category	Pseudo-method	
Syntax	*list*.**all_indices(***exp*: bool**)**: list of int	
Parameters	*list*	A list.
	exp	A Boolean expression.
Return value	A list of the indexes for all the matching items	

Returns a list of all indexes of items in *list* for which *exp* is TRUE. If no items satisfy the Boolean expression, an empty list is returned.

NOTE—Using **all_indices( )** on an empty list produces another empty list. Trying to use this result in a **gen keeping** constraint can cause a generation contradiction error.

Syntax example:

```
var l_2 : list of int;
l_2 = sys.packets.all_indices(it.length > 5)
```

27.5 Math and logic pseudo-methods

This subclause describes the syntax for pseudo-methods that perform arithmetic or logical operations to compute a value using all items in a list.

27.5.1 and_all()

Purpose	Compute the logical AND of all items	
Category	Pseudo-method	
Syntax	*list*.**and_all(***exp*: bool**)**: bool	
Parameters	*list*	A list.
	exp	A Boolean expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	A Boolean value	

Returns TRUE if all values of the *exp* are true; otherwise, it returns FALSE (if the *exp* is false for any item in the *list*). It stops computation once a FALSE is established. If the list is empty, this returns TRUE.

Syntax example:

```
var bool_val : bool;
bool_val = m_list.and_all(it >= 1)
```

27.5.2 or_all()

Purpose	Compute the logical OR of all items	
Category	Pseudo-method	
Syntax	*list*.**or_all(***exp*: bool**)**: bool	
Parameters	*list*	A list.
	exp	A Boolean expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	A Boolean value	

This returns a TRUE if any value of the *exp* is true; otherwise, it returns FALSE (if the *exp* is false for every item in the list or the list is empty). It stops computation once a TRUE is established.

Syntax example:

```
var bool_val : bool;
bool_val = m_list.or_all(it >= 100)
```

27.5.3 average()

Purpose	Compute the average of an expression for all items	
Category	Pseudo-method	
Syntax	*list*.**average(***exp*: numeric-type**)**: numeric-type	
Parameters	*list*	A list.
	exp	An integer expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The integer average	

This returns the integer average of the *exp* computed for all the items in the *list*. It returns `UNDEF` if the list is empty.

Syntax example:

```
var list_ave : int;
list_ave = sys.item_list.average(it.f_1 * it.f_2)
```

27.5.4 product()

Purpose	Compute the product of an expression for all items	
Category	Pseudo-method	
Syntax	*list*.**product(***exp*: numeric-type**)**: numeric-type	
Parameters	*list*	A list.
	exp	An integer expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The integer product	

This returns the integer product of the *exp* computed over all the items in the *list*. It returns `1` if the list is empty.

Syntax example:

```
var list_prod : int;
list_prod = sys.item_list.product(it.f_1)
```

27.5.5 sum()

Purpose	Compute the sum of all items	
Category	Pseudo-method	
Syntax	*list*.**sum(***exp*: numeric-type**)**: numeric-type	
Parameters	*list*	A list.
	exp	An integer expression. The **it** variable can be used to refer to the current list item, and the **index** variable can be used to refer to its index number.
Return value	The integer sum	

This returns the integer sum of the *exp* computed over all the items in the *list*. It returns `0` if the list is empty.

Syntax example:

```
var op_sum : int;
op_sum = sys.instr_list.sum(.op1)
```

27.6 List CRC pseudo-methods

This subclause describes the syntax for pseudo-methods that perform cyclic redundancy check (CRC) functions on lists. See also 20.1.1 and 20.1.2.

27.6.1 crc_8()

Purpose	Compute the CRC8 of a list of bits or a list of bytes	
Category	Pseudo-method	
Syntax	*list*.**crc_8(***from-byte*: int**,** *num-bytes*: int**)**: int	
Parameters	*list*	A list of bits or bytes.
	from-byte	The index number of the starting byte.
	num-bytes	The number of bytes to use.
Return value	The integer value	

This reads the *list* byte-by-byte and returns the integer value of the CRC8 function of a list of bits or bytes. Only the least significant byte is used in the result.

The CRC is computed starting with the *from-byte*, for *num-bytes*. If *from-byte* or *from-byte+num-bytes* is not in the range of the list, an error shall be issued.

NOTE—The algorithm for computing CRC8 is specific for the ATM HEC (header error control) computation. The code used for HEC is a cyclic code with the following generating polynomial:

```
x**8 + x**2 + x + 1
```

Syntax example:

```
print b_data.crc_8(2, 4)
```

27.6.2 crc_32()

Purpose	Compute the CRC32 of a list of bits or a list of bytes	
Category	Pseudo-method	
Syntax	*list*.**crc_32(***from-byte*: int**,** *num-bytes*: int**)**: int	
Parameters	*list*	A list of bits or bytes.
	from-byte	The index number of the starting byte.
	num-bytes	The number of bytes to use.
Return value	The integer value	

This reads the *list* byte-by-byte and returns the integer value of the CRC32 function of a list of bits or bytes. Only the least significant word is used in the result.

The CRC is computed starting with the *from-byte*, for *num-bytes*. If *from-byte* or *from-byte+num-bytes* is not in the range of the list, an error shall be issued.

NOTE—The algorithm for computing CRC32 generates a 32-bit CRC that is used for messages up to 64 kB in length. Such a CRC can detect 99.999999977% of all errors. The generator polynomial for the 32-bit CRC used for both Ethernet and token ring is:

```
x**32 + x**26 + x**23 + x**22 + x**16 + x**12 + x**11 + x**10 + x**8 + x**7
   + x**5 + x**4 +x**2 + x + 1
```

Syntax example:

```
print b_data.crc_32(2, 4)
```

27.6.3 crc_32_flip()

Purpose	Compute the CRC32 of a list of bits or a list of bytes, flipping the bits	
Category	Pseudo-method	
Syntax	*list*.**crc_32_flip(***from-byte*: int**,** *num-bytes*: int**)**: int	
Parameters	*list*	A list of bits or bytes.
	from-byte	The index number of the starting byte.
	num-bytes	The number of bytes to use.
Return value	The integer value	

This reads the *list* byte-by-byte and returns the integer value of the CRC32 function of a list of bits or bytes, with the bits flipped. Only the least significant word is used in the result. The bits are flipped as follows:

a) The bits inside each byte of the input are flipped.

b) The bits in the result are flipped.

The CRC is computed starting with the *from-byte*, for *num-bytes*. If *from-byte* or *from-byte+num-bytes* is not in the range of the list, an error shall be issued.

Syntax example:

```
print b_data.crc_32_flip(2, 4)
```

27.7 Keyed list pseudo-methods

This subclause describes the syntax for pseudo-methods that can be used only on keyed lists. Using one of these methods on a regular list shall result in an error.

Keyed lists are list in which each item has a key associated with it. For a list of structs, the key typically is the name of a particular field in each struct. Each unique value for that field can be used as a key.

— For a list of scalars, the key can be the **it** variable, referring to each item.

— When creating a keyed list, the key shall have a unique value for each item.

— Keyed lists can be searched quickly, by searching on a key value.

27.7.1 key()

Purpose	Get the item that has a particular key	
Category	Pseudo-method	
Syntax	*list*.**key(***key-exp*: exp**)**: list-item	
Parameters	*list*	A keyed list.
	key-exp	The key of the item to return.
Return value	The matching list item	

This returns the list item that has the specified key. If there is no such item, the default for the item's type is returned (see 5.1). For a list of scalars, a value of zero (`0`) is returned if there is no such item. Since zero (`0`) might be confused with a value found, do not use zero (`0`) as a key for scalar lists.

Syntax example:

```
var loc_list_item : location;
var i_key         : uint;
i_key = 5;
loc_list_item = locations.key(i_key)
```

27.7.2 key_index()

Purpose	Get the index of an item that has a particular key	
Category	Pseudo-method	
Syntax	*list*.**key_index(***key-exp*: exp**)**: int	
Parameters	*list*	A keyed list.
	key-exp	The key of the item for which the index is to be returned.
Return value	The index of the matching list item	

This returns the integer index of the item that has the specified key; otherwise, it returns `UNDEF` (if no item with that key exists in the list).

Syntax example:

```
var loc_list_ix : int;
loc_list_ix = locations.key_index(i)
```

27.7.3 key_exists()

Purpose	Check that a particular key is in a list	
Category	Pseudo-method	
Syntax	*list*.**key_exists(***key-exp*: exp**)**: bool	
Parameters	*list*	A keyed list.
	key-exp	The key for which to search.
Return value	A Boolean value	

This returns TRUE if the key exists in the list; otherwise, it returns FALSE.

Syntax example:

```
var loc_list_k : bool;
var i          := 5;
loc_list_k = locations.key_exists(i)
```

27.7.4 Restrictions on keyed lists

a) *list*.**resize()** cannot be used on keyed lists.

b) Keyed lists and regular (unkeyed) lists are different types. Assignment is not allowed between a keyed list and a regular list.

c) Keyed lists cannot be generated. Trying to generate a keyed list shall result in an error. Therefore, keyed lists need to be defined with the do-not-generate sign (!).