

23. Predefined methods library 1

A significant part of *e* functionality is implemented as set of predefined methods defined directly under the **global** and **sys** *structs*. Furthermore, every *struct* inherits a set of predefined methods. Some of these methods can be extended to add functionality and some of them are empty, allowing for user-definition. 5

Three other predefined *structs*, **semaphore**, **rdv_semaphore**, and **locker**, provide predefined methods that are useful in controlling TCMs and in controlling resource sharing between TCMs (see Clause 28). Then, there are pseudo-methods. Calls to pseudo-methods look like method calls. However, they are associated not with *struct* expressions, but with other kinds of expressions. 10

NOTE—Use the **pre_generate()** or **post_generate()** methods (see 23.1.2.2 and 23.1.2.3) to extend a struct or unit. 15

23.1 Predefined methods of any struct 15

This section defines the methods available for any instantiated user-defined struct or unit.

23.1.1 Setting unit relationships 20

These methods can be used to get or set unit-related information.

23.1.1.1 get_unit() 25

Returns a reference to the unit (see 5.4.1).

23.1.1.2 set_unit() 30

Changes the parent unit of a *struct* (see 5.4.4).

23.1.2 Methods called during execution phases 35

These methods are used to manipulate *structs* during execution.

23.1.2.1 The init() method of any_struct 40

Purpose	Customize the initialization of a struct
Category	Predefined method of any struct or unit
Syntax	[<i>exp</i>]. init()
Parameters	<i>exp</i> An expression that returns a unit or a <i>struct</i> .

The **init()** method of a struct is called when a new instance of the struct is created. This method can be extended to set different values for fields (other than their default values). By default, all fields of scalar type are initialized to zero (0). The initial value of a *struct* or list is **NULL**; unless the list is a sized list of scalars, in which case, it is initialized to the proper size with each item set to the default value. 45

The following considerations also apply.

- Initialize the non-generated fields of a *struct*, especially fields of an enumerated scalar type or unsized lists. 55

- Enumerated scalar types are initialized to zero (0), even if that is not a legal value for that type.
- Initialize any fields that might be sampled before being assigned.
- Size or initialize any lists that might be filled with data from the DUT.
- Unpacking data from the DUT into an unsized, uninitialized list shall cause a runtime error.
- If a field is initialized, but not marked as non-generated, the initialization is overwritten during generation. To mark a field as non-generated, place a ! character in front of the field name.
- *See also:* 17.4.2, 3.1, and 2.11.4.

Syntax example:

```
init() is also {
    is_ok = TRUE;
    list_subs = {320; 330; 340; 350; 360};
    list_color = {black; red; green; blue; yellow; white};
};
```

23.1.2.2 pre_generate()

Simplifies constraint expressions before they are analyzed by the constraint solver (see 7.3.2).

23.1.2.3 post_generate()

Derives more complex expressions or values from the generated values (see 7.3.3).

23.1.2.4 The run() method of any_struct

Purpose	Recommended place for starting TCMs
Category	Method of any struct or unit
Syntax	[<i>exp.</i>] run()
Parameters	<i>exp</i> An expression that returns a unit or a <i>struct</i> .

When a test is executed, the **global.run_test()** method call the **run()** methods of all *structs* under **sys**, starting from **sys** in depth-first search order. After this initial pass, when any struct is generated (with the **gen** action) or allocated (with **new**), its **run()** method is also invoked. This ensures:

- a) the **run()** method of each *struct* instance is called exactly once, thus avoiding multiple instances of the same started TCM;
- b) TCMs do not start and events do not occur before the *e* program is ready to accept them; and
- c) the **run()** method is called after generation and uses the generated values.

If multiple tests are run in the same session, the **run()** method is called once for each test in the session. The **init()** method is called only once before the first test.

This method can be extended to start user-defined TCMs. The method is initially empty. *See also:* 15.2.2.

Syntax example:

```
run() is also {
    start monitor();
};
```

23.1.2.5 The `quit()` method of any_struct

Purpose	Kill all threads of a struct or unit instance
Category	Predefined method of any struct or unit
Syntax	<code>[exp.]quit()</code>
Parameters	<i>exp</i> An expression that returns a unit or a <i>struct</i> .

This method deactivates a *struct* instance, killing all threads associated with the *struct* and enabling garbage collection. The `quit()` method emits a quit event for that *struct* instance at the end of the current tick [and](#) kills any TCM threads that were started within the *struct* in which the `quit()` method is called. All attached events and `expect` members of the *struct* that are still running are also killed.

A *thread* is any started TCM. If a started TCM calls other TCMs, those TCMs are considered subthreads of the started TCM thread, and the `quit()` method kills those subthreads, too. If a *struct* has more than one started TCM, each TCM runs on a separate, parallel thread. Each thread shares a unique identifier, or thread handle, with its subthreads. ****predefined methods of the scheduler??**

The `quit()` method is called by the `global.stop_run()` method (see 25.5). It can also be called explicitly.

Syntax example:

```
packet.quit();
```

23.1.3 Methods called for customizing specific operations

These methods define how to pack, unpack, or print *struct* information.

23.1.3.1 `do_pack()`

Packs the physical fields of the *struct* (see 17.4.1.1.1).

23.1.3.2 `do_unpack()`

Unpacks a packed list of bit into a *struct* (see 17.4.1.1.2).

23.1.3.3 The `do_print()` method of any_struct

Purpose	Print struct info
Category	Predefined method of any struct or unit
Syntax	<code>[exp.]do_print()</code>
Parameters	<i>exp</i> An expression that returns a unit or a <i>struct</i> .

This method controls the printing of information about a particular *struct*. It can be extended to customize the way information is displayed. This method is called by the `print` action whenever a *struct* is printed.

Syntax example:

```
do_print() is first {
    outf("Struct %s :", me.s);
};
```

23.1.3.4 The print_line() method of any_struct

Purpose	Print a struct or a unit in a single line
Category	Predefined method of any struct or unit
Syntax	[<i>exp</i> .]print_line(NULL <i>struct-type.type</i> ())
Parameters	<i>exp</i> An expression that returns a unit or a <i>struct</i> .
	NULL <i>struct-type.type</i> () To print a row representation of the <i>struct</i> or unit, the parameter is NULL. To print the header for the list, the parameter is of the form: <i>struct-type.type</i> ()

This method prints lists of structs of a common *struct* type in a tabulated table format. Each *struct* in the list is printed in a single line of the table.

There is a limit on the number of fields printed in each line when printing the *structs* — those fields that fit into a single line are printed — the rest are not printed at all. Each field is printed in a separate column and there is a limitation on the column width. When a field exceeds this width, it is truncated and an asterisk (*) is placed as the last character of that field's value.

Syntax example:

```
sys.pmi[0].print_line(sys.pmi[0].type());
sys.pmi[0].print_line(NULL);
```

23.2 Methods and predefined attributes of unit any_unit

The predefined methods for **any_unit** include:

- hdl_path()
- full_hdl_path()
- e_path()
- agent()
- get_parent_unit()

For details about each of these, see 5.3.

23.3 Pseudo-methods

Pseudo-methods calls look like method calls, but unlike methods they are not associated with *structs* and are applied to other types of expressions, such as lists. Pseudo-methods cannot be changed or extended through use of the **is only**, **is also** or **is first** constructs.

23.3.1 The `copy()` method of `any_struct`

Purpose	Make a shallow copy
Category	Predefined method of any struct or unit
Syntax	<code>exp.copy(): exp</code>
Parameters	<code>exp</code> Any legal <i>e</i> expression.

This returns a shallow, non-recursive copy of the expression. If the expression is a list or a *struct* that contains other lists or *structs*, the second-level items are not duplicated; instead, they are copied by reference. Table 1 details how the copy is made, depending on the type of the expression:

Table 1—Copying process

Expression	Procedure
scalar	The scalar value is simply assigned as in a normal assignment.
string	The whole string is copied.
scalar list	A new list with the same size as the original list is allocated and the contents of the original list are duplicated.
list of <i>structs</i>	A new list with the same size as the original list is allocated and the contents of the list is copied by reference, i.e., each item in the new list points to the corresponding item in the original list.
<i>struct</i>	A new <i>struct</i> instance with the same type as the original <i>struct</i> is allocated and all scalar fields are duplicated. All compound fields (lists or <i>structs</i>) in the new <i>struct</i> instance point to the corresponding fields in the original <i>struct</i> .

The following considerations also apply.

- Do not use the assignment operator (=) to copy *structs* or lists into other data objects. The assignment operator simply manipulates pointers to the data being assigned and does not create new *struct* instances or lists.
- Use the **deep_copy()** method (see 24.1.1) to create a recursive copy of a *struct* or list that contains compound fields or items.

Syntax example:

```
var pmv: packet = sys.pmi.copy();
```

23.3.2 `as_a()`

Converts an expression from one data type to another (see 3.8.1).

23.3.3 `get_enclosing_unit()`

Returns a reference to the nearest higher-level unit instance of the specified type (see 5.4.2).

23.3.4 try_enclosing_unit()

Returns a reference to the nearest higher-level unit instance of the specified type, without issuing a runtime error if no unit instance of the specified type is found (see 5.4.3).

23.3.5 type()

Purpose	Get a handle for the type of an expression
Category	Pseudo-method
Syntax	<i>exp.type()</i> : type_descriptor
Parameters	<i>exp</i> Any legal <i>e</i> expression.

This returns a handle for the type of an expression. ****strongly discouraged??**

Syntax example:

```
if pkt1.type() == pkt2.type() then {out("Got a match!");}
```

23.4 Coverage methods

The **covers struct** is a predefined struct containing methods used for coverage and coverage grading. With the exception of the **set_external_cover()** and **write_cover_file()** methods, all of these methods are methods of the **covers struct**:

- include_tests()
- set_weight()
- set_at_least()
- set_cover()
- get_contributing_runs()
- get_unique_buckets()
- set_external_cover()
- write_cover_file()
- get_overall_grade()
- get_ecov_name()
- get_test_name()
- get_seed()

For details about each of these, see 12.9.