

SMEIL Language Reference

Grammar

$\langle module \rangle$	$::= \{ \langle import-stm \rangle \} \langle entity \rangle$ $\{ \langle entity \rangle \}$
$\langle import-stm \rangle$	$::= \text{'import'} \langle import-name \rangle [\langle qualified-specifier \rangle] \text{';'}$ $ \text{'from'} \langle import-name \rangle$ $\text{'import'} \langle ident \rangle \{ \text{'}, \langle ident \rangle \} [\langle qualified-specifier \rangle]$ ';'
$\langle import-name \rangle$	$::= \langle ident \rangle \{ \text{'.'} \langle ident \rangle \}$
$\langle qualified-specifier \rangle$	$::= \text{'as'} \langle ident \rangle$
$\langle entity \rangle$	$::= \langle network \rangle$ $ \langle process \rangle$
$\langle network \rangle$	$::= \text{'network'} \langle ident \rangle \text{'('} [\langle params \rangle] \text{'})'}$ $\text{'{' } \{ \langle network-decl \rangle \} \text{'}'}$
$\langle process \rangle$	$::= [\text{'sync'} \text{'async'}] \text{'proc'} \langle ident \rangle$ $\text{'('} [\langle params \rangle] \text{'})' \{ \langle process-decl \rangle \}$ $\text{'{' } \{ \langle statement \rangle \} \text{'}'}$
$\langle network-decl \rangle$	$::= \langle inst-decl \rangle$ $ \langle bus-decl \rangle$ $ \langle const-decl \rangle$ $ \langle gen-decl \rangle$
$\langle process-decl \rangle$	$::= \langle var-decl \rangle$ $ \langle const-decl \rangle$ $ \langle bus-decl \rangle$ $ \langle enum-decl \rangle$ $ \langle inst-decl \rangle$ $ \langle gen-decl \rangle$
$\langle params \rangle$	$::= \langle param \rangle \{ \text{'}, \langle param \rangle \}$

$\langle param \rangle$::= ['[' [$\langle integer \rangle$] ']'] $\langle direction \rangle$ $\langle ident \rangle$
$\langle direction \rangle$::= 'in' (input signal) 'out' (output signal) 'const' (constant input value)
$\langle var-decl \rangle$::= 'var' $\langle ident \rangle$ ':' $\langle type-name \rangle$ ['=' $\langle expression \rangle$] [$\langle range \rangle$] ';' ,
$\langle range \rangle$::= 'range' $\langle expression \rangle$ 'to' $\langle expression \rangle$
$\langle enum \rangle$::= 'enum' $\langle ident \rangle$ '{' $\langle enum-field \rangle$ { ',' $\langle enum-field \rangle$ } '}' ';' ,
$\langle enum-field \rangle$::= $\langle ident \rangle$ ['=' $\langle integer \rangle$]
$\langle const-decl \rangle$::= 'const' $\langle ident \rangle$ ':' $\langle type-name \rangle$ '=' $\langle expression \rangle$ ';' ,
$\langle bus-decl \rangle$::= ['exposed'] 'bus' $\langle ident \rangle$ '{' $\langle bus-signal-decls \rangle$ '}' ';' ,
$\langle bus-signal-decls \rangle$::= $\langle bus-signal-decl \rangle$ { $\langle bus-signal-decl \rangle$ }
$\langle bus-signal-decl \rangle$::= $\langle ident \rangle$ ':' $\langle type \rangle$ ['=' $\langle expression \rangle$] [$\langle range \rangle$] ';' ,
$\langle inst-decl \rangle$::= 'instance' $\langle instance-name \rangle$ 'of' $\langle ident \rangle$ '(' [$\langle param-map \rangle$ { ',' $\langle param-map \rangle$ }] ')' ';' ,
$\langle instance-name \rangle$::= $\langle ident \rangle$ '[' $\langle expression \rangle$ ']' (indexed instance) $\langle ident \rangle$ (named instance) '_' (anonymous instance)
$\langle param-map \rangle$::= [$\langle ident \rangle$ ':'] $\langle expression \rangle$
$\langle gen-decl \rangle$::= 'generate' $\langle ident \rangle$ '=' $\langle expression \rangle$ 'to' $\langle expression \rangle$ '{' { $\langle network-decl \rangle$ } '}' ,
$\langle statement \rangle$::= $\langle name \rangle$ '=' $\langle expression \rangle$ ';' (assignment) 'if' '(' $\langle expression \rangle$ ')' '{' { $\langle statement \rangle$ } '}' { $\langle elif-block \rangle$ } [$\langle else-block \rangle$] 'for' $\langle ident \rangle$ '=' $\langle expression \rangle$ 'to' $\langle expression \rangle$ '{' { $\langle statement \rangle$ } '}' 'switch' $\langle expression \rangle$ '{' $\langle switch-case \rangle$ { $\langle switch-case \rangle$ } ['default' '{' $\langle statement \rangle$ { $\langle statement \rangle$ } '}'] '}' 'trace' '(' $\langle format-string \rangle$ { ',' $\langle expression \rangle$ } ')' ';' ,

	'assert' '(' $\langle expression \rangle$ [',' $\langle string-literal \rangle$] ')' ';' 'break' ';'
$\langle switch-case \rangle$::= 'case' $\langle expression \rangle$ '{' { $\langle statement \rangle$ } '}'
$\langle elif-block \rangle$::= 'elif' '(' $\langle expression \rangle$ ')' '{' { $\langle statement \rangle$ } '}'
$\langle else-block \rangle$::= 'else' '{' { $\langle statement \rangle$ } '}'
$\langle format-string \rangle$::= '"' { $\langle format-string-part \rangle$ } '"'
$\langle format-string-part \rangle$::= '{' '}' (placeholder string) $\langle string-char \rangle$
$\langle expression \rangle$::= $\langle name \rangle$ $\langle literal \rangle$ $\langle expression \rangle$ $\langle bin-op \rangle$ $\langle expression \rangle$ $\langle un-op \rangle$ $\langle expression \rangle$ '(' $\langle expression \rangle$ ')'
$\langle bin-op \rangle$::= '+' (addition) '-' (subtraction) '*' (multiplication) '/' (division) '%' (modulo) '==' (equal) '!=' (not equal) '<<' (shift left) '>>' (shift right) '<' (less than) '>' (greater than) '>=' (greater than or equal) '<=' (less than or equal) '&' (bitwise-and) ' ' (bitwise-or) '^' (bitwise-xor) '&&' (logical conjunction) ' ' (logical disjunction)
$\langle un-op \rangle$::= '-' (negation) '+' (identity) '!' (logical negation) '~' (bitwise-not)
$\langle literal \rangle$::= $\langle integer \rangle$ $\langle floating \rangle$ $\langle string-literal \rangle$

	'[' <i><integer></i> { ',' <i><integer></i> } ']' (Array literal) 'true' 'false' 'U' (Undefined value)
<i><string-literal></i>	::= '"' { <i><string-char></i> } '"'
<i><type></i>	::= 'i' <i><integer></i> (signed integer) 'int' (arbitrary-width signed integer) 'u' <i><integer></i> (unsigned integer) 'uint' (arbitrary-width unsigned integer) 'f32' (single-precision floating point) 'f64' (double-precision floating point) 'bool' (boolean value) '[' [<i><expression></i>] ']' <i><type></i> (array of type)
<i><ident></i>	::= <i><letter></i> { <i><letter></i> <i><number></i> '_' '-' } (identifier)
<i><name></i>	::= <i><ident></i> <i><name></i> '.' <i><name></i> (hierarchical accessor) <i><name></i> '[' <i><array-index></i> ']' (array element access)
<i><array-index></i>	::= '*' (wildcard) <i><expression></i> (element index)
<i><integer></i>	::= <i><number></i> { <i><number></i> } (decimal number) '0x' <i><hex-digit></i> { <i><hex-digit></i> } (hexadecimal number) '0o' <i><octal-digit></i> { <i><octal-digit></i> } (octal number)
<i><floating></i>	::= { <i><number></i> } '.' <i><number></i> { <i><number></i> }
<i><number></i>	::= '0' - '9'
<i><letter></i>	::= 'a' - 'z' 'A' - 'Z'
<i><hex-digit></i>	::= <i><number></i> 'a' - 'f' 'A' - 'F'
<i><octal-digit></i>	::= '0' - '7'
<i><string-char></i>	::= (ISO-8859-1 char with value > 26)

Operator precedence

Precedence	Operators
0	+ - ! ~ (unary)
1	* / %
2	+ -
3	<< >>
4	< > <= >=
5	== !=
6	& ^
7	&&
8	

Keywords

- as
- async
- barrier
- break
- bus
- case
- const
- default
- elif
- else
- enum
- exposed
- for
- from
- func
- generate
- if
- import
- in
- instance
- network
- of
- out
- proc
- range
- return
- switch
- sync
- to
- unique
- var
- where