

# The Language SW

BNF-converter

June 16, 2021

This document was automatically generated by the *BNF-Converter*. It was generated together with the lexer, the parser, and the abstract syntax module, which guarantees that the document matches with the implementation of the language (provided no hand-hacking has taken place).

## The lexical structure of SW

### Literals

String literals  $\langle String \rangle$  have the form " $x$ ", where  $x$  is any sequence of any characters except " unless preceded by \.

Integer literals  $\langle Int \rangle$  are nonempty sequences of digits.

Numvar literals are recognized by the regular expression  $\text{'\%'}\langle letter \rangle(\langle letter \rangle | \langle digit \rangle)^*$

Stringvar literals are recognized by the regular expression  $\text{'\$'}\langle letter \rangle(\langle letter \rangle | \langle digit \rangle)^*$

Envvar literals are recognized by the regular expression  $\text{'\$_'}\langle letter \rangle(\langle letter \rangle | \langle digit \rangle)^*$

Symvar literals are recognized by the regular expression  $\text{'\&'}\langle letter \rangle(\langle letter \rangle | \langle digit \rangle | \text{'\_'})^*$

SubId literals are recognized by the regular expression  $\text{'\^'}\langle letter \rangle(\langle letter \rangle | \langle digit \rangle | \text{'\_'})^*$

Id literals are recognized by the regular expression  $\langle letter \rangle(\langle letter \rangle | \langle digit \rangle | \text{'\_'})^*$

ValidImport literals are recognized by the regular expression  $\text{'\{'}\langle letter \rangle | \langle digit \rangle | \text{'\_'} | \text{'.'} | \text{'/'} \}^* \text{'\}'}$

Date literals are recognized by the regular expression  $\langle digit \rangle \langle digit \rangle \langle digit \rangle \langle digit \rangle (\text{'-'} \langle digit \rangle \langle digit \rangle)^* \text{'T'} (\text{'.'} | \langle digit \rangle | \text{'-'})^*$

## Reserved words and symbols

The set of reserved words is the set of terminals appearing in the grammar. Those reserved words that consist of non-letter characters are called symbols, and they are treated in a different way from those that are similar to identifiers. The lexer follows rules familiar from languages like Haskell, C, and Java, including longest match and spacing conventions.

The reserved words used in SW are the following:

```
INCLUDE  PREFIX  include
prefix
```

The symbols used in SW are the following:

```
;          {      }
<          -      >
-          (      )
.          /      =
StreamWork: ---   :
```

## Comments

Single-line comments begin with #.

Multiple-line comments are enclosed with {# and #}.

## The syntactic structure of SW

Non-terminals are enclosed between  $\langle$  and  $\rangle$ . The symbols  $::=$  (production),  $|$  (union) and  $\epsilon$  (empty rule) belong to the BNF notation. All other symbols are terminals.

$$\begin{aligned} \langle Valide \rangle & ::= \langle ValidConfig \rangle \\ & \quad | \quad \langle ValidSW \rangle \\ \langle ValidSW \rangle & ::= \langle ListStm \rangle \end{aligned}$$

$$\begin{aligned}
\langle Stm \rangle & ::= \langle Prefix \rangle \langle String \rangle \\
& | \quad \langle Include \rangle \langle String \rangle \\
& | \quad \langle DataFlow \rangle \\
& | \quad \langle Numassgn \rangle \\
& | \quad \langle Strassgn \rangle \\
& | \quad \langle SymAssgn \rangle \\
& | \quad \langle Hermt \rangle \\
& | \quad \langle Subdef \rangle \\
\langle ListStm \rangle & ::= \epsilon \\
& | \quad \langle Stm \rangle ; \langle ListStm \rangle \\
\langle Subdef \rangle & ::= \langle SubId \rangle \{ \langle ListSubnet \rangle \} \\
\langle Subnet \rangle & ::= \langle Hermt \rangle \\
& | \quad \langle DataFlow \rangle \\
& | \quad \langle ExtPortIn \rangle \\
& | \quad \langle ExtPortOut \rangle \\
\langle ListSubnet \rangle & ::= \epsilon \\
& | \quad \langle Subnet \rangle ; \langle ListSubnet \rangle \\
\langle ExtPortIn \rangle & ::= \langle Proc \rangle \langle Prt \rangle \langle Larrow \rangle \langle Tab \rangle \\
& | \quad \langle Tab \rangle \langle Rarrow \rangle \langle Prt \rangle \langle Proc \rangle \\
\langle ExtPortOut \rangle & ::= \langle Tab \rangle \langle Larrow \rangle \langle Prt \rangle \langle Proc \rangle \\
& | \quad \langle Proc \rangle \langle Prt \rangle \langle Rarrow \rangle \langle Tab \rangle \\
\langle Tab \rangle & ::= \langle Numval \rangle \\
& | \quad \langle Symval \rangle \\
\langle DataFlow \rangle & ::= \langle Proc \rangle \langle Prt \rangle \langle Larrow \rangle \langle Prt \rangle \langle Proc \rangle \\
& | \quad \langle Proc \rangle \langle Prt \rangle \langle Rarrow \rangle \langle Prt \rangle \langle Proc \rangle \\
& | \quad \langle DataFlow \rangle \langle Prt \rangle \langle Larrow \rangle \langle Prt \rangle \langle Proc \rangle \\
& | \quad \langle DataFlow \rangle \langle Prt \rangle \langle Rarrow \rangle \langle Prt \rangle \langle Proc \rangle \\
\langle Larrow \rangle & ::= < \langle TypeDef \rangle \langle Buffsize \rangle - \\
\langle Rarrow \rangle & ::= - \langle TypeDef \rangle \langle Buffsize \rangle > \\
\langle TypeDef \rangle & ::= \langle Symvalu \rangle \\
& | \quad \epsilon \\
\langle Buffsize \rangle & ::= \langle Numval \rangle \\
& | \quad \epsilon \\
\langle Hermt \rangle & ::= \langle Symvalu \rangle \langle Comp \rangle \langle ListArgument \rangle \\
& | \quad \langle Symvalu \rangle \langle ListArgument \rangle
\end{aligned}$$

$$\begin{aligned}
\langle \text{Symvalu} \rangle & ::= \langle \text{Symval} \rangle \\
& \quad | \quad - \\
\langle \text{Proc} \rangle & ::= ( \langle \text{Symvalu} \rangle \langle \text{Comp} \rangle \langle \text{ListArgument} \rangle ) \\
& \quad | \quad ( \langle \text{Symvalu} \rangle \langle \text{ListArgument} \rangle ) \\
\langle \text{Prt} \rangle & ::= \langle \text{Numval} \rangle \\
& \quad | \quad \langle \text{Numval} \rangle . \langle \text{Symval} \rangle \\
& \quad | \quad \langle \text{Symval} \rangle . \langle \text{Numval} \rangle \\
& \quad | \quad \langle \text{Symval} \rangle \\
& \quad | \quad \epsilon \\
\langle \text{Comp} \rangle & ::= \langle \text{Symval} \rangle \\
& \quad | \quad \langle \text{SubId} \rangle \\
& \quad | \quad \langle \text{ModPath} \rangle \langle \text{Symval} \rangle \\
& \quad | \quad \langle \text{RemPath} \rangle \\
\langle \text{ModPath} \rangle & ::= / \langle \text{Symval} \rangle / \\
& \quad | \quad \langle \text{Symval} \rangle / \\
& \quad | \quad \langle \text{ModPath} \rangle \langle \text{Symval} \rangle / \\
\langle \text{RemPath} \rangle & ::= \langle \text{ValidImport} \rangle \langle \text{Symval} \rangle \\
\langle \text{Argument} \rangle & ::= \langle \text{Stringval} \rangle \\
\langle \text{ListArgument} \rangle & ::= \epsilon \\
& \quad | \quad \langle \text{Argument} \rangle \langle \text{ListArgument} \rangle \\
\langle \text{Numassgn} \rangle & ::= \langle \text{Numvar} \rangle = \langle \text{Numval} \rangle \\
\langle \text{Strassgn} \rangle & ::= \langle \text{Stringvar} \rangle = \langle \text{Symval} \rangle \\
\langle \text{SymAssgn} \rangle & ::= \langle \text{Symvar} \rangle = \langle \text{Symval} \rangle \\
\langle \text{Numval} \rangle & ::= \langle \text{Integer} \rangle \\
& \quad | \quad \langle \text{Numvar} \rangle \\
\langle \text{Stringval} \rangle & ::= \langle \text{String} \rangle \\
& \quad | \quad \langle \text{Stringvar} \rangle \\
& \quad | \quad \langle \text{Envar} \rangle \\
\langle \text{Symval} \rangle & ::= \langle \text{Symvar} \rangle \\
& \quad | \quad \langle \text{Id} \rangle \\
& \quad | \quad \langle \text{Envar} \rangle \\
\langle \text{Include} \rangle & ::= \text{include} \\
& \quad | \quad \text{INCLUDE}
\end{aligned}$$

$$\begin{aligned}
\langle Prefix \rangle & ::= \text{PREFIX} \\
& | \text{prefix} \\
\langle ValidConfig \rangle & ::= \text{StreamWork: } \langle ListEntry \rangle \\
& | \text{--- } \langle ListEntry \rangle \\
\langle Entry \rangle & ::= \langle KeyVal \rangle \\
& | \langle KeyName \rangle \\
\langle ListEntry \rangle & ::= \epsilon \\
& | \langle Entry \rangle \langle ListEntry \rangle \\
\langle KeyVal \rangle & ::= \langle KeyName \rangle \langle Integer \rangle \\
& | \langle KeyName \rangle \langle String \rangle \\
& | \langle KeyName \rangle \langle Date \rangle \\
\langle KeyName \rangle & ::= \langle Symval \rangle : \\
& | \langle ModPath \rangle \langle Symval \rangle :
\end{aligned}$$