Tokens:

` Comment

‘…’ Character Token

“…” Keyword or Literally string –interpreted by options set

…=“…” Regex pattern for defining Tokens –quotes in the pattern must be escaped ‘\”’

^… State

~… Negate option, used to turn off an option that otherwise would be on (e.g. via

>…

@ Error Catch

# Symbol ID

$ Symbol Value

$name

?...: Set Option ‘…’

? “…” Conditional keyword, keyword is only defined for grammar context in which it is found

& Action Commands, e.g. &Warn

% Group and pattern reference e.g. !Symbol = (?<symbol>.\*)

! Default value/definition

%^ Precedence: No Association

%< Precedence: Left association

%> Precedence: Right association

\* Zero or More List Items

+ Concatenates tokens such that no text can be ignored between them. Used for multi-character operators etc. e.g. ‘>=’. LISTS: One or More List Items

. Defines the type for a grammar element

[] Option/List

() Explicitly defines association of grammar elements, e.g. G1 G2 | G3 G4 != G1 (G2 | G3) G4

{} Defines the action (code) of Gramma Elements

= Sets the declared grammar element to the value of this grammar element,

Or, if defining a Token Regex

<> Causes state to return to previous state

Lists and Optional Grammar:

‘[’ <GrammarList> [‘{‘ <ActionList> ‘}’] ‘]’

‘[’ <TerminalGrammarList> ‘;’ <RecursiveGrammarList> ‘]’ [ListOption]

TerminalGrammarList =

GrammarList = [Grammar {action1}; + {action2} ‘.’ {action3} ]+

= [Grammar {action1}; Grammar {action2} ‘.’ {action3}]+

= Grammar {action1} | Grammar {action2} ‘.’ {action3} GrammarList