Grammar

This is a grammar of the language presented in an enhanced BNF format:

sym
::= sequence-a
— a production of sym with two alternatives

::= sequence-b

sequence
— an optional sequence

sequence *
— optional sequence zero or more times

ε
— the empty sequence

Short semantic descriptions are in italics. Where ambiguous, the first of several alternatives is favored.

[This grammer is not totally acurate. It doesn't handle operator precedence.]

Language Elements

script
::= block

::= decl * primExp

A script is either a block of code, or a primitive.

block
::= decl * expression . * return expression .

a block consists of optional declarations followed by a sequence of expressions, which are executed in turn. The block may end in a return in which case the script (not just the block) is exited. The return may have an expression whose value is returned as the value of script. In absence of a return, the value of a block is the value of the last expression, and a script will return no value.

decl
::= $ formals .

formals
::= formal , formal *

::= ε

formal
::= functor word
named argument

::= - word
positional argument

::= functor
named argument with same local name

::= word
local variable

primExp
::= primitive word , message .

::= primitive word , number .

The optional word is the name of the primitive set, and either the message or number is key to which primitive.
expression
::= expHead expTail *

The base expression is executed, then the tail expressions left to right. Finally head expressions, right to left.

expHead
::= message value argsSet

::= value message argsSet

::= valSet

in the first two forms, the message is sent to the value with arguments; in the third form the result is the value

expTail
::= ; message argsSet

::= ops argsSet

the message (or operator) is sent to the value from the right with arguments

argsSet
::= args := expression

If there is an assignment then the whole message that these args are for is an assignment message; the value of the expression forms the last positional parameter to the message.

args
::= arg , arg *

::= ε

arg
::= keyword value
named argument with value

::= keyword
‘flag’ argument, implied value of true

::= value
positional argument

valSet
::= word := expression

::= word ?= expression
(3)

::= value := expression
(2)

The value of the expression to the right of the assignment symbol (the := or ?=) is assigned to the variable or value. The value of this assignment is the value of the expression.

1) word may be a local or a property of self

2) conditional assignment is made to either an argument or local (assignment is only made if the argument wasn't passed in, or if the local hasn't been assigned to yet

3) the property named by the value is assigned to (for example foo.x or foo @ 3).

value
::= primary member * !

::= primary member * ops

::= operator value ops

the members are evaluated left to right, each names a property that is fetched from the value to the left; If there is an exclamation, then the value is checked for undefined and a run-time error is generated if it is

opsSet
::= ops := expression

ops
::= operator value ops

primary
::= word ?
(1)

::= self

::= number

::= string

::= symbol

::= special

::= (expression)
value is the value of the expression

::= [block]
value is a block object

1) the word may be either a local, an argument, a global, or a property of self whose value is fetched. In the case of locals or arguments, the optional question mark results in a value of true or false, depending of if the value is defined or not.

Lexical Elements

. , ; ( ) [ ] ? ! := ?= $ -
the literal punctuation marks and symbols

return
the bold literal: return

self
any of the literals: self this super

number
a decimal number with optional fractional part and exponent, or a binary, octal, or hexadecimal number; may have a leading minus sign

string
a string of characters between double quotes; must appear on one line; can contain quoted special characters using the backslash notation

symbol
a dollar sign followed directly by any legal word or operator

special
any of the literals: true false nil ???

word
any non-bold sequence of alphabetic, numeric, or dash characters that begins with an alphabetic

functor
a bold version of word

operator
any sequence of the operator characters: ! # $ % & * + - / : < = > ? @ \ ^ | ~ except the sequences:
$:= ?= ? !

member
a word preceded by a period