SPARK-PL Dictionary (1.2)

Alexey Solovyev

Abstract

A short description of all SPARK-PL commands.

Table of Contents

1. Constants 6
1.1. null 6
1.2. pi 6
1.3. true 6
1.4. false 6
1.5. circle 6
1.6. square 6
1.7. square2 6
1.8. torus 7
1.9. Colors 7
2. Operations 7
2.1. ^ 7
2.2. *, / 7
2.3. * (dot) 8
2.4. % 8
2.5. +, - 8
2.6. + (string concatenation) 8
2.7. <, > , >=, <= 8
2.8. ==, != 8
2.9. - (unary) 9
2.10. and, or 9
2.11. not 9
2.12. =, +=, *=, -=, /= 9
2.13. is 9
2.14. as 10
3. Math commands 10
3.1. abs 10
3.2. acos 10
3.3. asin 10
3.4. atan 11
3.5. atan2 11
3.6. ceil 11
3.7. cos 11
3.8. exp 11
3.9. floor 11
3.10. log 11
3.11. round 12
3.12. sgn 12
3.13. sin 12
3.14. sqrt 12
3.15. tan 12

4. Commands 12
- 4.1. add-grid-space 13
- 4.2. add-physical-space 13
- 4.3. add-standard-space 13
- 4.4. add-standard3d-space 14
- 4.5. agents 15
- 4.6. agents-at 15
- 4.7. agents-at-as 16
- 4.8. agents-as 16
- 4.9. agents-here 16
- 4.10. agents-here-as 16
- 4.11. agents-number 16
- 4.12. agents-number-as 17
- 4.13. all-agents-at 17
- 4.14. all-agents-here 17
- 4.15. ask 17
- 4.16. break 18
- 4.17. complex 18
- 4.18. continue 18
- 4.19. count 19
- 4.20. create 19
- 4.21. create-circle 19
- 4.22. create-rect 20
- 4.23. create-file-reader 20
- 4.24. create-file-writer 21
- 4.25. create-new-file 21
- 4.26. create-unique-file 22
- 4.27. create-one 22
- 4.28. create-grid 22
- 4.29. create-grid3d 23
- 4.30. create-grid-in-space 23
- 4.31. create-list 23
- 4.32. create-vector 24
- 4.33. current-time-millis 24
- 4.34. delaunay-triangulation 24
- 4.35. distance 24
- 4.36. diffuse 25
- 4.37. evaporate 25
- 4.38. exit 25
- 4.39. for 25
- 4.40. fprint 26
- 4.41. get 26
- 4.42. get-grid 26
- 4.43. get-space 26
- 4.44. if 26
- 4.45. ifelse 27
- 4.46. interpolate 27
- 4.47. is-key-pressed 27
- 4.48. is-mouse-button-pressed 27
- 4.49. kill 27
- 4.50. max 28
- 4.51. min 28
- 4.52. mouse-position 28
- 4.53. myself 28

4.54. next-key-event 28
4.55. next-mouse-event 29
4.56. normal-random 29
4.57. num2int 29
4.58. one-of 29
4.59. parent 29
4.60. print 29
4.61. random-in-interval 30
4.62. random 30
4.63. random-vector 30
4.64. random-vector3 30
4.65. random-vector-of-length 30
4.66. random-vector3-of-length 30
4.67. repeat 31
4.68. return 31
4.69. self 31
4.70. set 31
4.71. space-xmin 31
4.72. space-xmax 31
4.73. space-ymin 32
4.74. space-ymax 32
4.75. space-xsize 32
4.76. space-ysize 32
4.77. sum 32
4.78. this 32
4.79. to-string 32
4.80. truncate 33
4.81. vector-in-direction 33
4.82. while 33
5. string 33
5.1. split 33
5.2. trim 34
5.3. to-num 34
5.4. to-int 34
5.5. starts-with 34
5.6. substring 35
5.7. substring-end 35
5.8. char-at 35
6. List 35
6.1. add 35
6.2. add-all 36
6.3. clear 36
6.4. set 36
6.5. remove-at 37
6.6. remove 37
6.7. remove-all 37
6.8. contains 38
7. vector 38
7.1. x 38
7.2. y 38
7.3. z 38
7.4. length 38
7.5. length-squared 38
7.6. normalize 38

7.7. truncate-length 39
7.8. copy 39
7.9. dot 39
7.10. cross 39
7.11. parallel-component 39
7.12. perpendicular-component 39
8. complex 40
8.1. re 40
8.2. im 40
8.3. abs 40
8.4. arg 40
8.5. reciprocal 40
8.6. conjugate 40
9. Reader 41
9.1. close 41
9.2. read-line 41
9.3. read-num 41
9.4. read-bool 41
9.5. read-int 42
9.6. read-vector 42
10. Writer 42
10.1. close 42
10.2. write-line 42
10.3. write-num 42
10.4. write-bool 42
10.5. write-vector 42
10.6. write-object 43
11. KeyEvent 43
11.1. pressed 43
11.2. name 43
11.3. space-name 43
12. MouseEvent 43
12.1. event-type 43
12.2. position 44
12.3. buttons 44
12.4. wheel 44
12.5. space-name 44
13. Space 44
13.1. create 44
13.2. create-one 44
13.3. x-min 45
13.4. x-max 45
13.5. y-min 45
13.6. y-max 45
13.7. x-size 45
13.8. y-size 45
13.9. agents-at 46
13.10. agents-at-as 46
13.11. all-agents-at 46
13.12. agents-here 46
13.13. agents-here-as 46
13.14. all-agents-here 47
14. Agent 47
14.1. die 47

14.2. is-dead 47
14.3. all-links 47
14.4. link 47
14.5. link-of-type 48
14.6. links-of-type 48
15. AbstractLink 48
15.1. connect 48
16. Link : AbstractLink 48
16.1. color 48
16.2. width 48
16.3. end1 48
16.4. end2 49
16.5. distance 49
17. SpaceAgent : Agent 49
17.1. move 49
17.2. get-agent-space 49
17.3. move-to 49
17.4. move-to-space 49
17.5. set-random-position 50
17.6. hatch 50
17.7. hatch-one 50
17.8. position 50
17.9. color 50
17.10. radius 50
17.11. label 50
17.12. alpha 51
17.13. rotation 51
17.14. fixed-rotation 51
17.15. restitution 51
17.16. shape 51
17.17. update-proxy 51
17.18. set-collision-category 51
17.19. add-collision-category 52
17.20. add-collision-mask 52
17.21. remove-collision-category 52
17.22. remove-collision-mask 52
17.23. apply-force 52
17.24. apply-impulse 52
17.25. get-velocity 53
18. grid 53
18.1. max 53
18.2. min 53
18.3. total-value 53
18.4. total-value-in-region 53
18.5. multiply 53
18.6. evaporate 53
18.7. diffuse 54
18.8. set-value 54
18.9. data-at 54
18.10. set-data-at 54
18.11. get-xsize 54
18.12. get-ysize 54
18.13. get-smooth-gradient 55
18.14. value-at 55

18.15. set-value-at 55
18.16. add-value-at 55
18.17. value-here 55
18.18. set-value-here 55
18.19. add-value-here 56
18.20. value 56

1. Constants

1.1. null

```
null : $NullObject
```

The null object (reference). Can be used for testing results of some operations.

Example

```
 var a = self as GoodAgent
 if a != null
 [
  do-something
 ]
```

1.2. pi

Mathematical pi constant, 3.1415926…

1.3. true

Logical true value. Can be assigned only to a Boolean variable

1.4. false

Logical false value.

1.5. circle

A circle shape.

1.6. square

A square shape. Affects only visual appearance of the corresponding space agent.

1.7. square2

A square shape. Affects both visual appearance and the collision detection algorithm of the corresponding space agent.

1.8. torus

A torus shape. Note: right now shapes can be used only in the following context: agent SomeAgent : SomeAgentDerivedFromSpaceAgent to create super 0.5 circle end That is, shapes are available only for agents derived from SpaceAgent, and shapes should be assigned using super command (which can be used without errors also only for SpaceAgent) with two arguments: radius and shape.

1.9. Colors

black, white, grey, red, green, blue, cyan, magenta, yellow, brown, orange, violet, pink. Note: all colors are treated as vectors. So, instead of colors it is possible to use vectors in the RGB format where each component is between 0 and 1 (smaller and bigger values are ignored).

2. Operations

Note: all operations (with few obvious exceptions) require two arguments. All operations are infix which means that the arguments are written from left and right sides from the operation symbol (e.g. 2 * 3).

Note: infix operations have precedence, that is, the order in which they are applied. The precedence is usual so no problems should occur but in any case it is possible to use parentheses to change the precedence.

Note: ignore type expressions after colons if you are not familiar with them, instead read descriptions.

2.1. ^

```
^ : double->double->double
```

Power. Works only for numbers.

Example:

```
a^b, 2^3.14, pi^pi
```

2.2. *, /

```
*,/ : double -> double -> double, double -> vector -> vector, vector -> double ->
*,/ : double -> complex, complex -> double, complex -> complex
```

Multiplication, division. Works with real and complex numbers and with a vector on one side and a real number on another side.

Example:

```
var x : double
var v : vector
x = x * 3
v = v / 4
v = v * 2
```

```
x = 2 * 2
```

2.3. * (dot)

```
* : vector -> vector -> number
```

A dot product of two vectors.

2.4. %

```
% : double -> double -> double
```

Returns the residual of a division

Example:

```
if tick % 10 == 0 [ print "True if the tick is a multiple of ten" ]
```

2.5. +, -

```
+, - : double -> double -> double, vector -> vector -> vector
+, - : double -> complex -> complex, complex -> double -> complex, complex -> comp
```

Addition, subtraction: work both for numbers and vectors.

2.6. + (string concatenation)

```
+ : string -> string -> string, string -> double -> string, string -> boolean -> s
+ : string -> vector -> string
```

Concatenates two strings or a string and a primitive value.

2.7. <, > , >=, <=

```
<, > , >=, <=: double -> double -> boolean
```

Comparison operations. Work for numbers only.

2.8. ==, !=

```
==, != : double -> double -> boolean, boolean -> boolean -> boolean
==, != : $NullObject -> $NullObject -> $NullObject
```

Equality, inequality. Work for any types. The only condition is that the types are compatible (that is, do not compare a vector and a number, etc.) Advanced note: actually, it is possible to compare, for instance, a vector and a grid since both of them are objects of the type $NullObject.

2.9. - (unary)

```
- (unary) : double -> double, complex -> complex, vector -> vector
```

Unary minus. Also works as a negation operation for vectors.

2.10. and, or

```
and, or : boolean ->boolean -> boolean
```

Logical operators AND and OR. Can be applied only to boolean arguments.

2.11. not

```
not : boolean -> boolean
```

Logical negation.

Example:

```
If not true == false [ print "OK" ]
```

2.12. =, +=, *=, -=, /=

```
=, +=, *=, -=, /=
```

Assignment operations from programming languages like C++, Java, C#.

+=, -= work with both numbers and vectors.

*=, /= work with numbers and with a vector on LHS (the left hand side) and a number on RHS.

*= works with a grid on LHS and a number on RHS.

2.13. is

```
variable is type : boolean

variable : $Object
type : $type
```

Checks whether an object on LHS is of type on RHS. Returns a boolean value.

Example:

```
var v : vector
if v is vector [ print "A vector is a vector"]
if v is grid [print "A vector is a grid"]
```

In this example only the first sentence will be printed out.

2.14. as

```
variable as type : extends $Object

variable : $Object
type : $type
```

Converts a given variable to a given type and returns a conversion result. If the conversion is not possible then the special null value is returned.

Example:

```
var agent-list = all-agents-here
ask agent-list
[
 var good-agent = self as GoodAgent
 if good-agent == null [ continue ]
 ; do something with good-agent
]
```

In this example a list of all agents at the position of the calling (current) agent is obtained. By default, the type of each agent in this list is SpaceAgent. Then all agents in the list are processed (by ask command). New variable 'good-agent' is created with the value of the currently processed agent converted to GoodAgent type. If the conversion was unsuccessful then continue with the next agent in the list.

3. Math commands

3.1. abs

```
abs value : number
value : number, complex
```

Returns the absolute value of a given number.

3.2. acos

```
acos value : number
value : number
```

Returns the arccosine of the argument measured in radians.

3.3. asin

```
asin value : number
value : number
```

Returns the arcsine of the argument measured in radians.

3.4. atan

```
atan value : number
value : number
```

Returns the arctangent of the argument measured in radians.

3.5. atan2

```
atan2 x y : number
x : number
y : number
```

Converts rectangular coordinates (x,y) to polar (r, theta) and returns theta.

3.6. ceil

```
ceil value : number
value : number
```

Returns the smallest value that is not less than the argument and is equal to an integer.

3.7. cos

```
cos value : number, complex
value : number, complex
```

Returns the cosine of the argument measured in radians.

3.8. exp

```
exp value : number, complex
value : number, complex
```

Returns the exponent of the argument.

3.9. floor

```
floor number : number
number : number
```

Rounds the number towards zero.

3.10. log

```
log value : number
```

```
value : number
```

Returns the natural logarithm of the argument.

3.11. round

```
round number : number
number : number
```

Rounds a number.

3.12. sgn

```
sgn number : number
number : number
```

Returns the sign of the argument.

3.13. sin

```
sin value : number, complex
value : number, complex
```

Returns the sine of the argument measured in radians.

3.14. sqrt

```
sqrt number : number
number : number
```

Returns the square root of a number.

3.15. tan

```
tan number : number, complex
number : number, complex
```

Returns the tangent root of the argument measured in radians.

4. Commands

Note: command descriptions are in the following format: Command name followed by arguments separated by whitespaces. If a command has a return value then after the colon the type of the return value is given. On the next lines arguments listed with their types after colons.

Note: special types (types which cannot be declared explicitly) are started with $ symbol.

List of special types:

$type	A name of any type (except number/double).
$integer	Type of integer numbers.
$NullObject	Any object type which can assume the null value.
$Object : $NullObject	Any object type (not 'double' or 'boolean').

4.1. add-grid-space

```
add-grid-space name x-min x-max y-min y-max wrap-x wrap-y : Space
name : string
x-min : number
y-min : number
x-max : number
y-max : number
wrap-x : boolean
wrap-y : boolean
```

Adds a new Grid space into a model. Call it in the model setup method. 'name' is the name of a new space. It can be used for getting a reference to this space with 'get-space' command.

Note: it is required to have one space declared with 'space' keyword. That space is considered to be the default space and is always used when no explicit reference to a space is given. Also, that default space has the name 'space' so it is prohibited to use the name 'space' for additional spaces.

Example: see 'add-standard-space'

4.2. add-physical-space

```
add-physical-space name x-min x-max y-min y-max wrap-x wrap-y : Space
name : string
x-min : number
y-min : number
x-max : number
y-max : number
wrap-x : boolean
wrap-y : boolean
```

Adds a new Physical space into a model. Call it in the model setup method. 'name' is the name of a new space. It can be used for getting a reference to this space with 'get-space' command.

Note: it is required to have one space declared with 'space' keyword. That space is considered to be the default space and is always used when no explicit reference to a space is given. Also, that default space has the name 'space' so it is prohibited to use the name 'space' for additional spaces.

Example: see 'add-standard-space'

4.3. add-standard-space

```
add-standard-space name x-min x-max y-min y-max wrap-x wrap-y : Space
name : string
```

```
x-min : number
y-min : number
x-max : number
y-max : number
wrap-x : boolean
wrap-y : boolean
```

Adds a new standard space into a model. Call it in the model setup method. 'name' is the name of a new space. It can be used for getting a reference to this space with 'get-space' command.

Note: it is required to have one space declared with 'space' keyword. That space is considered to be the default space and is always used when no explicit reference to a space is given. Also, that default space has the name 'space' so it is prohibited to use the name 'space' for additional spaces.

Example:

```
; Default space
space GridSpace -20 20 -20 20 true true

global another-space : Space

to setup
 another-space = add-standard-space "One more space"
                                       (-10) 10 (-10) 10
                                       true false

  ; Agents will be created in default space
 create SomeAgent 2

 ; Agents will be created in another-space
 ask another-space
 [
  create SomeAgent 20
  ask create-one AnotherAgent
   [
   set-random-position
   ]
 ]
end
```

In this example a new space is created inside setup method. In this case grids cannot be created inside that new space. See 'create-grid-in-space' command to resolve this issue.

4.4. add-standard3d-space

```
add-grid-space name x-min x-max y-min y-max z-min z-max wrap-x wrap-y wrap-z : Spa
name : string
x-min : number
y-min : number
z-min : number
x-max : number
y-max : number
```

```
z-max : number
wrap-x : boolean
wrap-y : boolean
wrap-z : boolean
```

Adds a new Standard3d space into a model. Call it in the model setup method. 'name' is the name of a new space. It can be used for getting a reference to this space with 'get-space' command.

Note: it is required to have one space declared with 'space' keyword. That space is considered to be the default space and is always used when no explicit reference to a space is given. Also, that default space has the name 'space' so it is prohibited to use the name 'space' for additional spaces.

Example: see 'add-standard-space'

4.5. agents

```
agents type : List<type>
type : $type
```

Returns a list of all agents of a specific type.

Example:

```
var all-good-agents = agents GoodAgent
var number-of-macrophages = count agents Macrophage
```

There is a better way to get the number of specific agents, that is, use agents-number command.

Note: only the agents which have precisely the requested type are returned.

```
agent Agent1 : Agent

agent Agent2 : Agent1

to get-agents1
 ; Only agents of type 'Agent1' will be returned.
 ; No agents of type Agent2 will be in the returned agent list.
 var agents1 = agents Agent1
end
```

4.6. agents-at

```
agents-at type point radius : List<type>
type : $type
point : vector
radius : number
```

Returns all agents in the default space of a specific type at the specific circular region (given by its center ‘point’ and radius ‘radius’). The radius cannot be negative. Zero radius means that the agents at a point are requested.

Example:

```
var good-agents-at-the-origin = agents-at GoodAgent [0,0,0] 0
```

4.7. agents-at-as

```
agents-at-as type point radius : List<type>
type : $type
point : vector
radius : number
```

Returns all agents in the default space derived from a specific type at the specific circular region (given by its center ‘point’ and radius ‘radius’). The radius cannot be negative. Zero radius means that the agents at a point are requested.

4.8. agents-as

```
agents-as type : List<type>
type : $type
```

Returns all agents derived from a specific type.

4.9. agents-here

```
agents-here type : List<type>
type : $type
```

Returns all agents of a specific type intersecting with the current active agent.

Note: only agents can call this function.

Note: agents in the same space as the calling agent are returned.

Note: the calling agent is also returned if its type is the same as the requested one.

4.10. agents-here-as

```
agents-here-as type : List<type>
type : $type
```

Returns all agents derived from a specific type intersecting with the current active agent.

Note: only agents can call this function.

Note: agents in the same space as the calling agent are returned.

Note: the calling agent is also returned if its type is derived form the requested one.

4.11. agents-number

```
agents-number type : number
type : $type
```

Returns the number of agents of a specific type.

4.12. agents-number-as

```
agents-number-as type : number
type : $type
```

Returns the number of agents derived from a specific type.

4.13. all-agents-at

```
all-agents-at point radius : List<SpaceAgent>
point : vector
radius : number
```

Returns all agents in the default space at a given circular region. Type of all returned agents is SpaceAgent. Look at 'as' operation example to see how to convert this type to other types.

4.14. all-agents-here

```
all-agents-here : List<SpaceAgent>
```

Returns all agents at the position of the calling agent.

Note: agents in the same space as the calling agent are returned.

4.15. ask

```
ask list [commands]
list : List, Array, $Object
```

Iterates through all objects in the list and performs commands for each object. Commands are given in brackets. 'self' inside the command block refers to the currently processed object from the list. 'myself' inside the command block refers to the object which called 'ask' command.

Note: if list == null, then no commands will be executed.

Example:

```
ask all-agents-here
[
 var good-agent = self as GoodAgent
 ask good-agent
 [
  ; the same as self.color = blue
  color = blue
  ; actually, the previous command do the same
  ; thing as
  ; myself.color = blue
  ; because good-agent variable refers to the same object as
  ; self variable from the outer ask command block.
 ]
]
```

Note: in the example above 'as' command could return null value, but it is not a problem since 'ask' command correctly works with null list and object arguments.

4.16. break

```
break
```

This command is equivalent to 'break' keyword in C, C++, Java, C#. It can be used only inside commands which iterate over some values. This command tells to stop the current iterative process ('while', 'for', 'repeat', or 'ask') and continue executing commands after that process.

Example:

```
repeat 100
[
 if true [ break ]
]
var y = 0
for x = 1:100
[
 if x == 50 [ break ]
 y += x
]
```

4.17. complex

```
complex re im : complex
re : double
im : double
```

Creates a complex number with the given real and imaginary parts.

4.18. continue

```
continue
```

This command is equivalent to 'continue' keyword in C, C++, Java, C#. It can be used only inside commands which iterate over some values. This command tells to start the next iteration immediately.

Example:

```
repeat 100
[
 if true [ continue ]
]
var y = 0
for x = 1:100
[
 if x == 50 [ continue ]
 y += x
]
```

4.19. count

```
count list : number
list : List, Array
```

Returns the number of elements in a list (array).

4.20. create

```
create type number : Array<type>
type : $type
number : number
```

Creates 'number' of objects of type 'type'. Cannot be used with types 'number' and 'boolean'. Commonly used for creating new agents.

Example:

```
create GoodAgent 1000
create BadAgent 1
var vectors = create vector 100 ; creates an array of vectors
ask create SomeAgent 20
[
 set-random-position
]
```

4.21. create-circle

```
create-circle radius : ShapeInfo
radius : double
```

Creates a circular shape description. Used for changing shapes of agents during a simulation process.

Note: this command works only for physical agents in a physical space.

Example

```
; SimpleAgent
agent SimpleAgent : SpaceAgent
```

```
to create
        super 0.4 dynamic-circle
        ; equivalent to
        ; radius = 0.7
        shape = create-circle 0.7
end

; Model
model Demo

space PhysicalSpace2d -10 10 -10 10 false false
```

4.22. create-rect

```
create-rect hx hy : ShapeInfo
hx : double
hy : double
```

Creates a rectangular shape description. Used for changing shapes of agents during a simulation process.

Note: this command works only for physical agents in a physical space.

Example

```
; SimpleAgent
agent SimpleAgent : SpaceAgent

to create
        super 0.4 dynamic-circle
        shape = create-rect 0.2 1
end

; Model
model Demo

space PhysicalSpace2d -10 10 -10 10 false false
```

4.23. create-file-reader

```
create-file-reader fname : Reader
fname : string
```

Creates a reader object for the given text file. Returns null if the file with the given name does not exist or if it cannot be open.

Example

```
var lines = create-list string

ask create-file-reader "data.txt"
```

```
[
 ; Read all lines
 while true
 [
  var line = read-line
  if line == null
   [ break ]

  ; Ignore comments
  if line.starts-with "#"
   [ continue ]

  lines.add line
 ]

 ; Close the reader
 close
]
```

4.24. create-file-writer

```
create-file-writer fname : Writer
fname : string
```

Creates a writer object for the given text file. Returns null if the file with the given name cannot be open. This command creates a new file if there is no file with the given name. Note: files are created in the output folder of a running SPARK model.

Example

```
ask create-file-writer "data.txt"
[
 write-num 1
 write-num 2

 ; Close the writer
 close
]
```

4.25. create-new-file

```
create-new-file file-name
file-name : string
```

Creates a new file with the given name. If a file with the given name exists, then it will be deleted and a new file with the same name will be created. Note: files are created in the output folder of a running SPARK model.

Example

```
create-new-file "test.log"
```

4.26. create-unique-file

```
create-unique-file name-prefix
name-prefix : string
```

Creates a new file with the given name. If another file with the given name exists, then the name is modified such that a new file has its own unique name. Note: files are created in the output folder of a running SPARK model.

Example

```
create-unique-file "test"
```

4.27. create-one

```
create-one type : Type of this command equals to 'type'
type : $type
```

Creates one object of a specific type. The main difference form the 'create' command with number = 1 is that the type of this command is different.

Example

```
(create-one GoodAgent).color = red
; it is not possible to write the following
; (create GoodAgent 1).color = red
; because the command in parentheses has type Array
```

4.28. create-grid

```
create-grid name x-size y-size : grid
name : string
x-size : number
y-size : number
```

Creates a grid with a given name and of a given dimension in the default space. This command can be used in the grid declaration code inside a model class.

Note: if no grid initialization is given for a grid, then the default initialization will be created with the name equals to variable's name and with the dimension of the default space. So the only use of this command is to create grids of different resolutions.

Note: in the current implementation 'name' should be the same as variable's name.

Example:

```
model Model
```

```
global data = create-grid "data" 10 10
```

4.29. create-grid3d

```
create-grid3d name x-size y-size z-size : grid3d
name : string
x-size : number
y-size : number
z-size : number
```

Creates a 3-dimensional grid. See create-grid for additional information.

4.30. create-grid-in-space

```
create-grid space name x-size y-size : grid
space : Space, string
name : string
x-size : number
y-size : number
```

Creates a grid with a given name and of a given dimension in a given space. This command can be used in the grid declaration code inside a model class.

Note: the space should be initialized before this command.

Note: in the current implementation 'name' should be the same as variable's name.

Example:

```
model Model

; Default space
space StandardSpace -20 20 -20 20 true true

; Additional space
global space2 = add-standard-space "space2" (-10) 10 (-10) 10 true false

; Grid will be created in the default space
global data = create-grid "data" 10 10

; Grid will be created in space2
global data2 = create-grid-in-space space2 "data2"
                                    space2.x-size space2.y-size

; Grid will be created in the default space with the default size
global data3 : grid
```

4.31. create-list

```
create-list type : List<$type>
```

```
type : $type
```

Creates a list which can contain object of the given type.

Example:

```
; Create a list of strings
var list = create-list string

list.add "first"
list.add "second"
```

4.32. create-vector

```
create-vector x y z : vector
x : number
y : number
z : number
```

Creates a new vector with the given components

4.33. current-time-millis

```
current-time-millis : double
```

Returns the current system time in milliseconds.

4.34. delaunay-triangulation

```
delaunay-triangulation list-of-agents link-type
list-of-agents : List<SpaceAgent>
link-type : $type (should be derived from SpaceLink)
```

Computes Delaunay triangulation of points representing centers of agents in the list. Agents in the list will be connected by links of the given type when the triangulation is computed.

Example:

```
; Kill all existing MyLink's
kill agents MyLink
; Compute Delaunay triangulation and connect agents with MyLink's
delaunay-triangulation (agents TestAgent) MyLink
```

4.35. distance

```
distance agent1 agent2 : vector
agent1 : SpaceAgent
agent2 : SpaceAgent
```

```
distance pos1 pos2 : vector
pos1 : vector
pos2 : vector
```

Returns the vector v equals to agent2.position – agent1.position (pos2 - pos1). It does more: in the torus topology the shortest vector-distance is returned.

Example:

```
; get the vector-distance between agents and then compute the length of this vecto
var distance-between-agents = (distance agent1 agent2).length
```

4.36. diffuse

```
diffuse grid value
grid : grid (DataLayer), grid3d
value : number
```

Performs the diffusion operation on a given grid. 'value' is the diffusion coefficient.

The diffusion operation is performed as follows. Each data layer cell shares (coefficient * 100) percents of its value with its eight neighbors. The coefficient should be between 0 and 1 for a well-defined behavior.

4.37. evaporate

```
evaporate grid value
grid : grid (DataLayer), grid3d
value : number
```

Performs the evaporation operation on a given grid. 'value' is the evaporation coefficient.

Evaporation simply means multiplication of all data layer values by the given value.

4.38. exit

```
exit
```

This command ends the current method. Only methods that do not return any value can use this command. Other methods should use 'return'.

4.39. for

```
for var-name = from : step : to [] (a special syntax, not a type definition)
for from : to [] (another form with step = 1 by default)
from : double
step : double
to : double
```

A loop command for iterating values of the given variable from 'from' to 'to' with the step size 'step'.

Note: var-name should be a unique name, no other local variables with the same name should exist in the method using a 'for' loop.

4.40. fprint

```
fprint file-name object
file-name : string
object : $Object, string, number
```

Prints a text representation of the given object into the given file. If a file with the given name doesn't exist, then a new file will be created. If a file with the given name exists, then a new text line will be appended to the end of this file.

Example

```
fprint "test.log" "2 + 2 = "
fprint "test.log" (2 + 2)
; The following two lines will be added to the file "test.log":
; 2 + 2 =
; 4
```

See create-writer and Writer object for another way of creating files in SPARK.

4.41. get

```
get list index
list : List, Array
index : number
```

Returns an element from the list at the position specified by the index.

4.42. get-grid

```
get-grid name : $DataLayer
name : string
```

Returns a grid with the given name.

4.43. get-space

```
get-space name : Space
name : string
```

Returns a space with the given name. Default space has name 'space'.

4.44. if

```
if condition [commands]
condition : boolean
```

A standard if control statement. If the condition is true, then the commands will be executed.

4.45. ifelse

```
ifelse condition [commands1][command2]
condition : boolean
```

A standard if-else control statement. If the condition is true, then commands from the first block will be executed, otherwise commands from the second block will be performed.

4.46. interpolate

```
interpolate v1 v2 t  : vector (number)
v1 : vector (number)
v2 : vector (number)
t : number
```

This command is a slightly optimized version of v1 * (1 - t) + t * v2

4.47. is-key-pressed

```
is-key-pressed key-name : boolean
key-name : string
```

Returns true if the key with the given name is pressed.

4.48. is-mouse-button-pressed

```
is-mouse-button-pressed button : boolean
button : double
```

Returns true if the given mouse button is pressed (1 = left button, 2 = right button).

4.49. kill

```
kill agent-list
agent-list : List, Array, $Object
```

Kills all agents in the list, that is, this command calls 'die' for each agent in the list. This command is a shortcut for

```
ask agent-list
[
 die
]
```

4.50. max

```
max a b : number
a : number
b : number
```

Returns the maximum of two numbers.

4.51. min

```
min a b : number
a : number
b : number
```

Returns the minimum of two numbers.

4.52. mouse-position

```
mouse-position : vector
```

Returns the current mouse position.

4.53. myself

```
myself : $Object
```

Returns a reference to the calling agent. See 'ask' for more information.

4.54. next-key-event

```
next-key-event : KeyEvent
```

Returns the next keyboard event from the keyboard event queue. Returns null if the queue is empty.

Example

```
var left-pressed : bool

; Process all keyboard events
while true
[
 var event = next-key-event
 if event == null
  [ break ]
 ; Left arrow event
 if event.name == "left"
 [
  left-pressed = event.pressed
 ]
]
```

4.55. next-mouse-event

```
next-mouse-event : MouseEvent
```

Returns the next mouse event from the mouse event queue. Returns null if the queue is empty

4.56. normal-random

```
normal-random mean std : double
mean : double
std : double
```

Returns a normally distributed random number.

4.57. num2int

```
num2int n : $integer
n : double
```

Converts the given floating point number to an integer number.

4.58. one-of

```
one-of list : Type of elements of list
list : List, Array
```

Returns one randomly selected element of the given list.

Note: the list argument should be not null.

4.59. parent

```
parent : $Object
```

Returns a reference to the parent class of the current class.

4.60. print

```
print str
str : string, double, boolean, $Object
```

Prints out a string

4.61. random-in-interval

```
random-in-interval a b : number
a : number
b : number
```

Returns a uniformly distributed random number in the interval [a,b).

4.62. random

```
random a : number
a : number
```

Returns a uniformly distributed random number in the interval [0, a)

4.63. random-vector

```
random-vector a b : vector
a : number
b : number
```

Returns a random vector with uniformly distributed components in the interval [a, b).

Note: this function returns a random 2-d vector with the third component 0.

4.64. random-vector3

```
random-vector a b : vector
a : number
b : number
```

Returns a random vector with uniformly distributed components in the interval [a, b).

Note: this function returns a random 3-d vector.

4.65. random-vector-of-length

```
random-vector-of-length length : vector
length : number
```

Returns a random vector of the given length.

Note: this function returns a random 2-d vector with the third component 0.

4.66. random-vector3-of-length

```
random-vector-of-length length : vector
length : number
```

Returns a random vector of the given length.

Note: this function returns a random 3-d vector.

4.67. repeat

```
repeat number [commands]
number : number
```

Repeats commands the given number of times.

4.68. return

```
return value
```

Returns the given 'value' as method’s value.

Note: the type of 'value' should be compatible with method’s return type.

4.69. self

```
self : $Object
```

Returns a reference to the active object (agent).

4.70. set

```
set list index value
list : List, Array
index : number
value : Type of elements of the list
```

Sets a new value for the element at the specific position of the list.

4.71. space-xmin

```
space-xmin : number
```

Returns the minimum x coordinate of the default space.

4.72. space-xmax

```
space-xmax : number
```

Returns the maximum x coordinate of the default space.

4.73. space-ymin

```
space-ymin : number
```

Returns the minimum y coordinate of the default space.

4.74. space-ymax

```
space-ymax : number
```

Returns the maximum y coordinate of the default space.

4.75. space-xsize

```
space-xsize : number
```

Returns the size of the default space along x-axis. That is, space-xmax – space-xmin.

4.76. space-ysize

```
space-ysize : number
```

Returns the size of the default space along y-axis. That is, space-ymax – space-ymin.

4.77. sum

```
sum list : number
list : grid, grid3d
```

Returns a total sum of all data values stored in a grid.

4.78. this

```
this : $Object
```

Returns a reference to the current object.

4.79. to-string

```
to-string arg : string
n : number, boolean, $integer, vector, $Object
```

Converts the argument into a string.

Note: the argument should be not null.

4.80. truncate

```
truncate v min max : vector
v : vector
min : number
max : number
```

Truncates all components of a given vector.

4.81. vector-in-direction

```
vector-in-direction length angle : vector
length : number
angle : number
```

Returns a vector of length 'length' in the direction specified by 'angle'.

Note: the angle should be measured in degrees.

Note: a 2-d vector is returned.

4.82. while

```
while condition [commands]
```

Commands in the block are executed while the condition is true.

5. string

Methods and fields of the 'string' class.

5.1. split

```
split regexp : List<string>
regexp : string
```

Splits the string object into a list of strings based on the given regular expression (Java regular expression syntax).

Example

```
var str = "first,second"
var list = str.split ","
; Prints "first"
print (get list 0)
; Prints "second"
```

```
print (get list 1)
```

5.2. trim

```
trim : string
```

Removes leading and ending spaces in the string object.

Example

```
var str = " first, second "
var list = str.split ","
; Prints "first"
print (get list 0)
; Prints "second"
print (get list 1)
```

5.3. to-num

```
to-num : number
```

Converts the string object into a number.

Example

```
var str = "12.4"
var x = str.to-num + 1
; Prints 13.4
print x
```

5.4. to-int

```
to-int : $integer
```

Converts the string object into an integer number.

5.5. starts-with

```
starts-with str : boolean
str : string
```

Returns true if the string object starts with the given substring.

Example

```
var str = "first,second"
; Prints 'true'
```

```
print str.starts-with "first"
```

5.6. substring

```
substring begin end : string
begin : number
end : number
```

Returns a substring of the string object with the first symbol specified by the 'begin' index, and the last symbol specified by the 'end' index (indices start from 0).

Note: the 'end' index is exclusive, that is, the substring symobls have 'begin', 'begin' + 1,..., 'end' - 1 indices in the original string.

Example

```
var str = "first,second"
; Prints "st,s"
print str.substring 3 7
```

5.7. substring-end

```
substring-end begin : string
begin : number
```

Returns a substring of the string object which starts at the 'begin' index.

Example

```
var str = "first,second"
; Prints "econd"
print str.substring-end 7
```

5.8. char-at

```
char-at i : number
i : number
```

Returns the code of a symbol at the given position in the string object.

6. List

Methods and fields of list objects.

6.1. add

```
add object
```

```
object : $Object (the type should be compatible with the list subtype)
```

Adds an object to the list object.

Example

```
var list = create-list string
list.add "first"
```

6.2. add-all

```
add-all list
list : List<type> (the type should be compatible with the list subtype)
```

Adds all objects from the argument to the list object.

Example

```
var list = create-list string
var list2 = create-list string
list.add "first"
list2.add "second"
list2.add "third"

list.add-all list2
```

6.3. clear

```
clear
```

Removes all elements from the list object.

6.4. set

```
set i val
i : number
val : $Object (the type should be compatible with the list subtype)
```

Sets a new value at the given index.

Note: the index should be less than the length of the list.

Example

```
var list = create-list string

ask list
[
 add "first"
 set 0 "new first"
]
```

6.5. remove-at

```
remove-at i
i : number
```

Removes an element at the given index.

Example

```
var list = create-list string
list.add "first"
list.add "second"
list.remove-at 0
; Now the list has only 1 element: "second"
```

6.6. remove

```
remove object
object : $Object (the type should be compatible with the list subtype)
```

Removes the given object from the list. If the object is not inside the list, then the list is not changed.

Example

```
var list = create-list string
list.add "first"
list.add "second"

list.remove "second"
list.remove "third";

; Now the list has 1 element: "first"
```

6.7. remove-all

```
remove-all list-arg
list-arg : List<type> (the type should be compatible with the list subtype)
```

Removes all elements of the argument from the list object.

This command is equivalent to the following code

```
ask list-arg
[
 list.remove self
]
```

6.8. contains

```
contains object : boolean
object : $Object (the type should be compatible with the list subtype)
```

Returns true if the list contains the given object.

7. vector

Methods and fields of the 'vector' class.

7.1. x

```
x : number
```

The first component of a vector.

7.2. y

```
y : number
```

The second component of a vector.

7.3. z

```
z : number
```

The third component of a vector.

7.4. length

```
length : number
```

Returns the length of a vector.

7.5. length-squared

```
length-squared : number
```

Returns the squared length of a vector.

7.6. normalize

```
normalize : vector
```

Normalizes a vector and returns a reference to itself.

7.7. truncate-length

```
truncate-length max-length : vector
max-length : number
```

If the length of a vector is higher than 'max-length' then its length is truncated to 'max-length'. Otherwise, the vector is unchanged. A reference to itself is returned.

7.8. copy

```
copy : vector
```

Returns a copy of a vector.

7.9. dot

```
dot v : number
v : vector
```

Computes the dot product with a given vector.

7.10. cross

```
cross v : vector
v : vector
```

Computes the cross product with a given vector. The result overrides the current vector's value.

7.11. parallel-component

```
parallel-component unit-vector : vector
unit-vector : vector
```

Returns a component of a vector parallel to a given unit vector.

Note: the argument vector should be of a unit length for this method to yield a correct result.

This method computes: (u,v) * u, where u is the argument, and v is the vector object.

7.12. perpendicular-component

```
perpendicular-component unit-vector : vector
```

```
unit-vector : vector
```

Returns a component of a vector perpendicular to a given unit vector.

Note: the argument vector should be of a unit length for this method to yield a correct result.

This method computes: v - (u,v) * u

8. complex

Methods and fields of a class for complex numbers.

8.1. re

```
re : number
```

Returns the real part of a complex number.

8.2. im

```
im : number
```

Returns the imaginary part of a complex number.

8.3. abs

```
abs : number
```

Returns the absolute value of a complex number.

8.4. arg

```
arg : number
```

Returns the polar argument of a complex number.

8.5. reciprocal

```
reciprocal : complex
```

Computes the inverse of a complex number.

8.6. conjugate

```
conjugate : complex
```

Returns the conjugate of a complex number.

9. Reader

A class for reading text files.

9.1. close

```
close
```

Closes an open file.

9.2. read-line

```
read-line : string
```

Returns the next line from an open file. Returns null if no new lines are available.

Example

```
ask create-reader "test.txt"
[
 ; Read all lines
 while true
 [
  var line = read-line
  if line == null
   [ break ]

  ; Process the line
 ]

 ; Close the file
 close
]
```

9.3. read-num

```
read-num : number
```

Reads a number from a file.

9.4. read-bool

```
read-bool : boolean
```

Reads a boolean value from a file.

9.5. read-int

```
read-int : $integer
```

Reads an integer from a file.

9.6. read-vector

```
read-vector : vector
```

Reads a vector from a file.

10. Writer

A class for writing text files.

10.1. close

```
close
```

Closes an open file.

10.2. write-line

```
write-line line
line : string
```

Writes a string into a file.

10.3. write-num

```
write-num n
n : number
```

Writes a number into a file.

10.4. write-bool

```
write-bool b
b : boolean
```

Writes a boolean value into a file.

10.5. write-vector

```
write-vector v
v : vector
```

Writes a vector into a file.

10.6. write-object

```
write-object obj
obj : $Object
```

Converts the given object into a string and writes this string into a file. If the object is null, then "null" is written into a file.

11. KeyEvent

A class for keyboard events.

11.1. pressed

```
pressed : boolean
```

Returns the status of the key for this event.

11.2. name

```
name : string
```

Returns the name of the key for this event.

11.3. space-name

```
space-name : string
```

Returns the space name for which the event was created.

12. MouseEvent

A class for mouse events.

12.1. event-type

```
event-type : string
```

Returns the type of the event. Possible types are: "LBUTTON_DOWN", "LBUTTON_UP", "RBUTTON_DOWN", "RBUTTON_UP", "MBUTTON_DOWN", "MBUTTON_UP", "MOUSE_MOVE", "MOUSE_WHEEL".

12.2. position

```
position : vector
```

Returns the mouse position for the event.

12.3. buttons

```
buttons : $integer
```

Returns the mouse button states for the event.

12.4. wheel

```
wheel : $integer
```

Returns the mouse wheel state for the event.

12.5. space-name

```
space-name : string
```

Returns the space name for which the event was created.

13. Space

13.1. create

```
create type number : $Array<type>
type : $type (derived from SpaceAgent)
number : number
```

Creates 'number' of agents of the type 'type' in a specific space.

Example:

```
; Create 1000 GoodAgent in the space "Space2".
ask get-space "Space2"
[
 create GoodAgent 1000
]
```

13.2. create-one

```
create-one type : Type of this command equals to ‘type’
```

```
type : $type (derived from SpaceAgent)
```

Creates one agent of a specific type in a given space. The main difference form the previous command with number = 1 is that the type of this command is different.

Example

```
var space1 = add-standard-space (-10) 10 (-20) 20 true false
(space1.create-one GoodAgent).color = red
; it is not possible to write the following
; (space1.create GoodAgent 1).color = red
; because the command in parentheses has type $Array
```

13.3. x-min

```
x-min : number
```

Returns the minimum x coordinate of a space.

13.4. x-max

```
x-max : number
```

Returns the maximum x coordinate of a space.

13.5. y-min

```
y-min : number
```

Returns the minimum y coordinate of a space.

13.6. y-max

```
y-max : number
```

Returns the maximum y coordinate of a space.

13.7. x-size

```
x-size : number
```

Returns the size of a space along the x-axis. That is, x-max – x-min.

13.8. y-size

```
y-size : number
```

Returns the size of a space along the y-axis. That is, y-max – y-min.

13.9. agents-at

```
agents-at type point radius : List<type>
type : $type
point : vector
radius : number
```

Returns all agents in a given space of a specific type at the specific circular region (given by its center ‘point’ and radius ‘radius’). Radius cannot be negative. Zero radius means that the agents at a point are requested.

13.10. agents-at-as

```
agents-at-as type point radius : List<type>
type : $type
point : vector
radius : number
```

Returns all agents in a given space derived from a specific type at the specific circular region (given by its center ‘point’ and radius ‘radius’). Radius cannot be negative. Zero radius means that the agents at a point are requested.

13.11. all-agents-at

```
all-agents-at point radius : List<SpaceAgent>
point : vector
radius : number
```

Returns all agents in a given space at a given circular region. Type of all returned agents is SpaceAgent.

13.12. agents-here

```
agents-here type : List<type>
type : $type
```

Returns all agents of a specific type intersecting with the current active agent.

Note: only agents can call this function.

Note: agents in a given space are returned.

Note: the calling agent may also be returned if its type is the same as requested one.

13.13. agents-here-as

```
agents-here-as type : List<type>
type : $type
```

Returns all agents derived from a specific type intersecting with the current active agent.

Note: only agents can call this function.

Note: agents in a given space are returned.

Note: the calling agent may also be returned if its type is derived from requested one.

13.14. all-agents-here

```
all-agents-here : List<SpaceAgent>
```

Returns all agents at the position of a calling agent.

Note: agents in a given space are returned.

14. Agent

14.1. die

```
die
```

Destroys an agent.

14.2. is-dead

```
is-dead : boolean
```

Returns true if the calling agent is dead.

14.3. all-links

```
all-links : List<AbstractLink>
```

Returns all links of this agent.

14.4. link

```
link agent : Link
agent : Agent
```

'link a' returns a link if the active agent is connected to 'a', or 'null' if there is no link between the active agent and 'a'.

14.5. link-of-type

```
link-of-type link-type
link-type : Link
```

Return a link of the specific type connected to the current agent.

14.6. links-of-type

```
links-of-type link-type
link-type : List<link-type>
```

Returns all links of the given type connected to the current agent.

15. AbstractLink

A base class for all links.

15.1. connect

```
connect end1 end2
end1 : Agent
end2 : Agent
```

Connects two agents with the link.

16. Link : AbstractLink

A link class for space agents.

16.1. color

```
color : vector
```

Link's color.

16.2. width

```
width : number
```

Link's width (used for visualization only).

16.3. end1

```
end1 : SpaceAgent
```

Returns the first end of the link.

16.4. end2

```
end2 : SpaceAgent
```

Returns the second end of the link.

16.5. distance

```
distance : vector
```

Returns a vector distance between agents connected by the link.

17. SpaceAgent : Agent

17.1. move

```
move v
v : vector
```

Adds 'v' to the current position and moves the space agent to that new position.

17.2. get-agent-space

```
get-agent-space : Space
```

Returns the space in which the space agent is located.

17.3. move-to

```
move-to pos
pos : vector
```

Moves the space agent to a new position.

17.4. move-to-space

```
move-to-space new-space new-position
new-space : Space
new-position : vector
```

Moves an agent to a given space at a specific position inside that space.

17.5. set-random-position

```
set-random-position
```

Moves an agent to a random position inside a space.

17.6. hatch

```
hatch type number : Array<type>
type : $type (derived from SpaceAgent)
number : number
```

Creates ‘number’ of agents of type ‘type’ in the same space as a calling agent and at the same position as a calling agent.

17.7. hatch-one

```
hatch-one type : Type of this command equals to ‘type’
type : $type (derived from SpaceAgent)
```

Creates one agent of a specific type in the same space as a calling agent and at the same position as a calling agent. The main difference form the previous command with number = 1 is that the type of this command is different.

17.8. position

```
position : vector
```

Field 'position' is the position of an agent inside a space. This is a read only field. To assign a value to the position use 'move-to' or 'move' methods.

17.9. color

```
color : vector
```

Field 'color' is the color of a space agent.

17.10. radius

```
radius : number
```

Field 'radius' is the size of a space agent.

17.11. label

```
label : string
```

Field 'label' of the space agent. Labels of space agents could be visualized during a simulation process.

17.12. alpha

```
alpha : number
```

Field 'alpha' determines the transparency of the space agent. If alpha = 0, then the agent is completely transparent. If alpha = 1, then the agent is completely opaque.

17.13. rotation

```
rotation : number
```

Rotation of the agent in the xy-plane (measured in degrees).

17.14. fixed-rotation

```
fixed-rotation : boolean (set only)
```

If true, then the physical simulation engine will not attempt to rotate the agent. Works only for physical agents.

17.15. restitution

```
restitution : number
```

Field 'restitution'. Works only for physical agents.

17.16. shape

```
shape : ShapeInfo (set only)
```

Field 'shape'. Allows to change agent's shape dynamically. Works only for physical agents. See create-circle and create-rect commands for examples.

17.17. update-proxy

```
update-proxy
```

A special command for physical agents. In general, should be called when a physical agent is moved to a new position with 'move' or 'move-to' methods.

17.18. set-collision-category

```
set-collision-category category
category : number
```

Sets a collision category of a physical agent. A collision category is an integer number from 0 to 15.

17.19. add-collision-category

```
add-collision-category category
category : number
```

Adds a collision category to a physical agent. A collision category is an integer number from 0 to 15.

17.20. add-collision-mask

```
add-collision-mask mask
mask : number
```

Adds a collision mask to a physical agent. A collision mask is an integer number from 0 to 15.

17.21. remove-collision-category

```
remove-collision-category category
category : number
```

Removes a collision category from a physical agent. A collision category is an integer number from 0 to 15.

17.22. remove-collision-mask

```
remove-collision-mask
mask : number
```

Removes a collision mask from a physical agent. A collision mask is an integer number from 0 to 15.

17.23. apply-force

```
apply-force force
force : vector
```

Applies the given force to a physical agent.

17.24. apply-impulse

```
apply-impulse impulse
impulse : vector
```

Applies the given impulse to a physical agent.

17.25. get-velocity

```
get-velocity : vector
```

Returns a velocity vector of a physical agent.

18. grid

18.1. max

```
max : number
```

Returns the maximum value stored in a grid.

18.2. min

```
min : number
```

Returns the minimum value stored in a grid.

18.3. total-value

```
total-value : number
```

Returns the sum of all values stored in a grid.

18.4. total-value-in-region

```
total-value-in-region x-min x-max y-min y-max : number
x-min : number
x-max : number
y-min : number
y-max : number
```

Returns the sum of all values stored in a grid in a specific region.

18.5. multiply

```
multiple coefficient
coefficient : number
```

Multiplies all values in a grid by a given coefficient.

18.6. evaporate

```
evaporate coefficient
coefficient : number
```

Multiplies all values in a grid by a given coefficient (this is an alias for the 'multiply' method).

18.7. diffuse

```
diffuse coefficient
coefficient : number
```

Performs a diffusion operation on a grid with a given diffusion coefficient.

The diffusion operation is performed as follows. Each data layer cell shares (coefficient * 100) percents of its value with its eight neighbors. The coefficient should be between 0 and 1 for a well-defined behavior.

18.8. set-value

```
set-value value
value : number
```

Sets all values in a grid to a given value.

18.9. data-at

```
data-at i j : number
i : number
j : number
```

Returns a value in a cell (i, j).

18.10. set-data-at

```
set-data-at i j new-value
i : number
j : number
new-value : number
```

Sets the value of a cell (i, j) to a new value.

18.11. get-xsize

```
get-xsize : number
```

Returns the number of cells in a grid along x-axis.

18.12. get-ysize

```
get-xsize : number
```

Returns the number of cells in a grid along y-axis.

18.13. get-smooth-gradient

```
get-smooth-gradient point : vector
point : vector
```

Returns a gradient at the given point.

18.14. value-at

```
value-at pos : number
pos : vector
```

Returns a value corresponding to a given position.

18.15. set-value-at

```
set-value-at pos new-value
pos : vector
new-value : number
```

Sets the value of a cell corresponding to a given position.

18.16. add-value-at

```
add-value-at pos num
pos : vector
num : number
```

Adds a given number to the value of a cell corresponding to a given position.

18.17. value-here

```
value-here : number
```

Returns a value corresponding to the position of a calling space agent.

18.18. set-value-here

```
set-value-here new-value
new-value : number
```

Sets the value of a cell corresponding to the position of a calling space agent.

18.19. add-value-here

```
add-value-here num
num : number
```

Adds a given number to the value of a cell corresponding to the position of a calling space agent.

18.20. value

```
value : number
```

The field 'value' is the value of a cell corresponding to the position of a calling space agent.