SPARK-PL Dictionary

Alexey Solovyev

Abstract

A short description of all SPARK-PL commands.

Table of Contents

1. Constants

1.1. pi

Mathematical pi constant, 3.1415926…

1.2. true

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

1.3. false

Logical false value.

1.4. circle

A circle shape.

1.5. square

A square shape.

1.6. 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.

1.7. Colors

black white grey red green blue cyan magenta yellow 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).

1.8. agents-number

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

Returns the number of all agents of a specific type.

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 parenthesis to change the precedence.

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

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 ->
```

Multiplication, division. Works with numbers and with a vector one side and a 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, string->string->str
```

Addition, subtraction: work both for numbers and vectors. Also + can be used for string concatenation.

2.6. <, > , >=, <=

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

Comparison operations. Work for numbers only.

2.7. ==, !=

```
==, != : double -> double -> boolean, vector -> vector -> Boolean
```

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 but they are always not equal.

2.8. - (unary)

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

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

2.9. and, or

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

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

2.10. not

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

Logical negation.

Example:

```
If not true == false [ print “OK” ]
```

2.11. =, +=, *=, -=, /=

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

Usual assignment operations from such programming languages as 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.

2.12. is

```
variable is type : boolean

variable : $Object
type : $type
```

Operation that 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.

2.13. 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 a 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
```

Returns the absolute value of a given number.

3.2. acos

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

Returns the arc cosine of an angle.

3.3. asin

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

Returns the arc sine of an angle.

3.4. atan

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

Returns the arc tangent of an angle.

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
value : number
```

Returns the cosine of an angle.

3.8. exp

```
exp value : number
value : number
```

Returns the exponent of an argument.

3.9. floor

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

Rounds number towards zero.

3.10. log

```
log value : number
value : number
```

Returns the natural logarithm of an argument.

3.11. round

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

Rounds a number.

3.12. sin

```
sin value : number
value : number
```

Returns the sine of an angle.

3.13. sqrt

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

Returns the square root of a number.

3.14. tan

```
tan number : number
number : number
```

Returns the tangent root of a number.

4. Commands

Note: command descriptions are in the following format: Command name followed by arguments separated by whitespaces. If the command has a return value then after a 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 $ sign.

List of special types:

$type	A name of any type (except number/double).
$ArrayList<type>	list of objects of type ‘type’
$Array<type>	array of objects of type ‘type’, array and list are different types

4.1. add-netlogo-space

```
add-netlogo-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 NetLogo space into a model. Call it in the model setup method. 'name' is a 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 default space and is always used when no explicit reference to a space is given. Also, that default space has name 'space' so it is prohibited to use name 'space' for additional spaces.

Example: see 'add-standard-space'

4.2. 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 a 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 default space and is always used when no explicit reference to a space is given. Also, that default space has name 'space' so it is prohibited to use name 'space' for additional spaces.

Example:

```
; Default space
space NetLogoSpace -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.

4.3. agents

```
agents type : $ArrayList<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
```

4.4. agents-at

```
agents-at type point radius : $ArrayList<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’). 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
```

4.5. agents-at-as

```
agents-at-as type point radius : $ArrayList<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’). Radius cannot be negative. Zero radius means that the agents at a point are requested.

4.6. agents-here

```
agents-here type : $ArrayList<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 a calling agent are returned.

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

4.7. agents-here-as

```
agents-here-as type : $ArrayList<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 a calling agent are returned.

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

4.8. agents-number

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

Returns the number of agents of a specific type.

4.9. agents-number-as

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

Returns the number of agents derived from a specific type.

4.10. agents-as

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

Returns all agents which are derived from a given type.

4.11. all-agents-at

```
all-agents-at point radius : $ArrayList<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.

4.12. all-agents-here

```
all-agents-here : $ArrayList<SpaceAgent>
```

Returns all agents at the position of a calling agent.

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

4.13. ask

```
ask list [commands]
list : $ArrayList (or $Array, or a single object)
```

Iterates through all objects in the list and performs commands for each object. Commands are given in brackets. ‘self’ inside command block refers to the currently processed object from the list. ‘myself’ inside 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 equals (in some sense)
  ; to 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 argument.

4.14. 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 like 'while', 'for', or 'repeat' 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
]
```

4.15. 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
]
```

4.16. count

```
count list : number
list : $ArrayList (or $Array)
```

Returns the number of elements in a list.

4.17. create

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

Creates ‘number’ of objects of type ‘type’. Cannot be used with type ‘number’. 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
]
```

4.18. 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 folder from which SPARK was started.

Example

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

4.19. 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 previous 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
```

4.20. 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 default space. This command can be used during grid declaration inside model class.

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

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

Example:

```
model Model
global data = create-grid “data” 10 10
```

4.21. create-grid-in-space

```
create-grid space name x-size y-size : grid
space : Space (or 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 during grid declaration inside model class.

Note: space should be initialized before this command.

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

Example:

```
model Model

; Default space
space NetLogoSpace -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 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 default space with default size
global data3 : grid
```

4.22. create-vector

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

```
y : number
z : number
```

Creates a new vector with the given components

4.23. 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
```

4.24. diffuse

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

Performs a diffusion operation on a given grid. ‘value’ is a diffusion coefficient.

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

4.25. evaporate

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

Performs evaporation on a given grid. ‘value’ is an evaporation coefficient.

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

4.26. exit

```
exit
```

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

4.27. 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 line of text will be appended to the end of the 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
```

4.28. get

```
get list index
list : $ArrayList (or $Array)
index : number
```

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

4.29. get-space

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

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

4.30. if

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

Standard if control statement. If condition is true, then commands are executed.

4.31. ifelse

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

Standard if-else control statement. If condition is true, then commands from the first block are executed, otherwise commands from the second block are performed.

4.32. interpolate

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

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

4.33. kill

```
kill agent-list
agent-list : $ArrayList (or $Array)
```

Kills all agents in the list, that is, calls ‘die’ method for each agent in the list. This command is a shortcut for

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

4.34. max

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

Returns the maximum of two numbers.

4.35. min

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

Returns the minimum of two numbers.

4.36. myself

```
myself : $Object
```

Returns a reference to the calling agent.

4.37. null

```
null
```

A null object. Can be used for testing results of some operations.

Example

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

4.38. parent

```
parent : $Object
```

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

4.39. print

```
print str
str : string
```

Prints out a string

4.40. random-in-interval

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

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

4.41. random

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

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

4.42. random-vector

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

```
b : number
```

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

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

4.43. random-vector-of-length

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

Returns a random vector of the given length.

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

4.44. repeat

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

Repeats commands the given number of times.

4.45. return

```
return value
```

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

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

4.46. self

```
self : $Object
```

Returns a reference to the active object (agent).

4.47. space-xmin

```
space-xmin : number
```

Returns the minimum x coordinate of space.

4.48. space-xmax

```
space-xmax : number
```

Returns the maximum x coordinate of the default space.

4.49. space-ymin

```
space-ymin : number
```

Returns the minimum y coordinate of the default space.

4.50. space-ymax

```
space-ymax : number
```

Returns the maximum y coordinate of the default space.

4.51. space-xsize

```
space-xsize : number
```

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

4.52. space-ysize

```
space-ysize : number
```

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

4.53. sum

```
sum list : number
list : grid
```

Returns a sum of all data stored in a grid.

4.54. this

```
this : $Object
```

Returns a reference to the current object.

4.55. truncate

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

Truncates all components of a given vector.

4.56. 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: angle should be in degrees.

Note: a 2-d vector is returned.

4.57. while

```
while condition [commands]
```

Commands in the block are executed while condition is true.

5. vector

Methods and fields of the 'vector' are described in this section.

5.1. x

```
x : number
```

Field 'x' is the first component of a vector.

5.2. y

```
y : number
```

Field 'y' is the second component of a vector.

5.3. z

```
z : number
```

Field 'z' is the third component of a vector.

5.4. length

```
length : number
```

Returns an absolute value of a vector, its length.

5.5. length-squared

```
length-squared : number
```

Returns a squared absolute value of a vector.

5.6. normalize

```
normalize : vector
```

Normalizes a vector and returns a reference to itself.

5.7. 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, no changes. A reference to itself is returned.

5.8. copy

```
copy : vector
```

Returns a copy of a vector.

5.9. dot

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

Computes the dot product with a given vector.

5.10. cross

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

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

5.11. parallel-component

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

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

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

5.12. perpendicular-component

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

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

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

6. Space

6.1. create

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

Creates ‘number’ of agents of type ‘type’ in a specific space.

Example:

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

6.2. 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
```

6.3. x-min

```
x-min : number
```

Returns the minimum x coordinate of a space.

6.4. x-max

```
x-max : number
```

Returns the maximum x coordinate of a space.

6.5. y-min

```
y-min : number
```

Returns the minimum y coordinate of a space.

6.6. y-max

```
y-max : number
```

Returns the maximum y coordinate of a space.

6.7. x-size

```
x-size : number
```

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

6.8. y-size

```
y-size : number
```

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

6.9. agents-at

```
agents-at type point radius : $ArrayList<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.

6.10. agents-at-as

```
agents-at-as type point radius : $ArrayList<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.

6.11. all-agents-at

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

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

6.12. agents-here

```
agents-here type : $ArrayList<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: calling agent may also be returned if its type is the same as requested one.

6.13. agents-here-as

```
agents-here-as type : $ArrayList<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: calling agent may also be returned if its type is derived from requested one.

6.14. all-agents-here

```
all-agents-here : $ArrayList<SpaceAgent>
```

Returns all agents at the position of a calling agent.

Note: agents in a given space are returned.

7. Agent

7.1. die

```
die
```

Destroys an agent.

7.2. 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'.

7.3. link-of-type

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

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

7.4. links-of-type

```
links-of-type link-type
link-type : $Array<link-type>
```

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

8. SpaceAgent : Agent

8.1. move

```
move v
v : vector
```

Adds to the current position 'v' and a space agent is moved to that new position.

8.2. get-agent-space

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

Returns the space in which a space agent is located.

8.3. move-to

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

Moves an agent to a new position.

8.4. 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.

8.5. set-random-position

```
set-random-position
```

Moves an agent to a random position inside a space.

8.6. 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.

8.7. 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.

8.8. 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.

8.9. color

```
color : vector
```

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

8.10. radius

```
radius : number
```

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

9. Link

9.1. connect

```
connect end1 end2
end1 : SpaceAgent
end2 : SpaceAgent
```

Connects two agents by the link.

9.2. color

```
color : vector
```

Field 'color' is the color of a link.

9.3. distance

```
distance : vector
```

Computes a vector-distance between two ends of the link (between two agents which are connected by the link)

9.4. end1

```
end1 : SpaceAgent
```

Returns an agent associated with the first end of the link

9.5. end2

```
end2 : SpaceAgent
```

Returns an agent associated with the second end of the link

9.6. width

```
width : number
```

Field 'radius' is the width of a link.

9.7. width

```
width : number
```

Field 'radius' is the width of a link.

10. grid

10.1. max

```
max : number
```

Returns the maximum value stored in a grid.

10.2. min

```
min : number
```

Returns the minimum value stored in a grid.

10.3. total-value

```
total-value : number
```

Returns the sum of all values stored in a grid.

10.4. 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.

10.5. multiply

```
multiple coefficient
coefficient : number
```

Multiplies all values in a grid by a coefficient.

10.6. evaporate

```
evaporate coefficient
coefficient : number
```

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

10.7. 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 gives equal shares of (coefficient * 100) percent of its value to its eight neighbors. Coefficient should be between 0 and 1 for well-defined behavior.

10.8. set-value

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

Sets all values in a grid to a given value.

10.9. data-at

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

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

10.10. 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.

10.11. value-at

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

Returns a value corresponding to a given position.

10.12. 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.

10.13. 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.

10.14. value-here

```
value-here : number
```

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

10.15. 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.

10.16. 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.

10.17. value

```
value : number
```

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