
SPARK-PL: Data Layers

Alexey Solovyev

Abstract

In this tutorial the concept of data layers is explained.

Table of Contents

1. Data layers in SPARK	1
2. Data layers in SPARK-PL	1
3. Working with grid values	2
4. Global grid methods	3

1. Data layers in SPARK

A data layer is one of key components of SPARK. Roughly speaking, it defines a function on a space, that is, it assigns a numerical value to each point in a space.

2. Data layers in SPARK-PL

There is one type for working with data layers in SPARK-PL called 'grid'. Generally, all data layers in a model should be available for all agents, so it is convenient to have a global variable for each data layer. In this case, the declaration of a new data layer is very simple

```
global data-layer : grid
```

Globally defined data layer (grid) variables are treated in a special way in SPARK-PL: if there is no initialization then a default initialization will be created. This default initialization creates a grid in a space and its dimension equals to the integer dimension of the space.

Another way to create a grid is to use 'create-grid' command. This command has three arguments: name of a new grid (this name can be used to get a reference to a grid from any part of a model, for example in the case when there is no global reference to a grid), x dimension of a grid, y dimension of a grid. As always in SPARK-PL, x and y could be non-integer in which case they will be rounded toward zero.

Note: in the current implementation only grids declared as global variables can be visualized during model simulation process; if you are using 'create-grid' command to initialize a globally defined grid then the name argument should be the same as global variable's name.

Remember that whenever you define a global variable of type 'grid' and do not provide an initialization for it, then a default initialization will be always created. For example, in the code below two grids will be created

```
model Model
```

```
space StandardSpace -10 10 -10 10 true true
```

```
; A default initialization is created for this
; grid which is equivalent to the command
; create-grid "data-layer" 20 20
global data-layer : grid

to setup
  data-layer = create-grid "data" 10 10

  ; Now the global variable 'data-layer' contains
  ; a reference to a 10 by 10 grid called "data".

  ; If the name "data-layer" were used in the previous line,
  ; then an error would occur during runtime
  ; because all data layers should have unique names
  ; and there is already one grid called "data-layer"
end
```

In this example two grids will be created in the model. One of them called "data" is available through a global variable 'data-layer'. Another grid called "data-layer" can be accessed using a command

```
get-grid "data-layer"
```

This command returns a reference to a grid with the given name. Note that only one of these two grids can be visualized during runtime: "data-layer" grid because it has the same name as a global variable (this issue will be resolved in further SPARK-PL releases).

It is convenient to use 'create-grid' as an initialization command

```
global data = create-grid "data" 11 2
```

3. Working with grid values

A method 'set-value' sets the same value in all grid entries. It has one argument: a value to be set. There are several methods for working with local grid values. A method 'value-at' returns a value from a grid entry corresponding to a specific point in a space. To set value at a specific point, use 'set-value-at' method. There is also a method 'add-value-at' which adds a given value to the value at a specific point.

```
; all grid entries will be 1
data.set-value 1

; get a value at a point [2, 3, 0]
; (the third component of a vector is ignored here)
var value = data.value-at [2, 3, 0]
var new-value = value * value

; set a new value at a point [2.4, 1, 0]
data.set-value-at [2.4, 1, 0] new-value

; add a number to a value at a point [2, 3, 0]
data.add-value-at [2, 3, 0] value
```

Of course, all these methods are accurate up to the resolution of a grid. So the following commands often do the same things

```
data.set-value-at [1.02, 2.5, 0] 0.1
data.set-value-at [1.01, 2.53, 0] 0.1
```

Agents can work with data layers in a more convenient way. For agents it is often required to know a value or to set a value at the same position at which they are located. Of course, it is always possible to do so using 'value-at', 'set-value-at' methods with agent's position but there is a shorter way to do this. There are methods 'value-here', 'set-value-here', 'add-value-here' which automatically use agent's position for working with data values. Of course, only space agents can use these methods.

```
agent SomeAgent : SpaceAgent

to step
  var data-here = data.value-here
  var new-data = data-here * 4 + 5

  data.set-value-here new-data

  data.add-value-here 0.5
end
```

Instead of 'value-here' and 'set-value-here' methods, it is more convenient to use the field 'value' of a grid. You can work with this field as with a usual variable. The following example does the same thing as the previous one.

```
to step
  data.value = data.value * 4 + 5.5
end
```

4. Global grid methods

By global methods I understand methods which affect a whole grid. One global method was already introduced: 'set-value' method.

The method 'total-value' returns a sum of all values in a grid. There is a command counterpart of this method called 'sum' which takes one argument: a grid.

```
var total-data = data.total-value
var total-data2 = sum data

if total-data == total-data2
[
  print "I'm not surprised"
]
```

There is a variation of the 'total-value' method called 'total-value-in-region'. This method has four arguments which specify a region in which a data should be summed up.

```
var value-in-rectangle =  
    data.total-value-in-region x-min x-max y-min y-max
```

Methods 'max' and 'min' return maximum and minimum values stored in a grid respectively.

```
var max-value = data.max  
var min-value = data.min
```

Method 'multiply' multiplies all grid values by a given number. This method has an alias 'evaporate' which reflects one common application of this method. Also there is a command 'evaporate' with one argument (a grid) which does the same thing, and there is an operator '*=' with a grid on the left and a number on the right which is equivalent to the 'multiply' method.

```
data.multiply 2  
data.evaporate 0.6  
evaporate data 0.3  
data *= 0.99
```

Method 'diffuse' performs a diffusion operation on a grid. It has one argument: a diffusion coefficient. There is a command 'diffuse' with two argument which does the same thing.

```
data.diffuse 0.5  
diffuse data 1
```

This method works as follows. Each data layer cell gives equal shares of (coefficient * 100) percents of its value to its eight neighbors. This diffusion coefficient should be between 0 and 1 for a well-defined behavior.