|  |
| --- |
| Circle Language Spec: System Objects |

## Assignment

Calls to systemcommands (like **Get Object**, **Set Value** or **Use As Class**) might not be commonly seen. They might be more likely to be called indirectly. An assignment command might indirectly carry out a **Get** on one object and a **Set** on another object, trying to yield over something from one symbol to another.

### Distinguishing Object-Bound and Reference-Bound Aspects

To demonstrate the difference between object-bound aspects and reference-bound aspects this article tries to employ the following notation. When a reference-bound aspect might be **Get** or **Set** then a reference might be displayed with a parent around it:



When an object-bound aspect might be **Get** or **Set** then a targeted object may be displayed without a parent around it:



### Basic Assignment Types

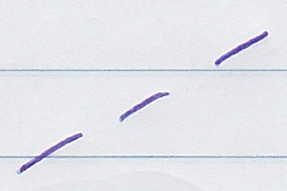
Different system aspects (such as a value, object or class) might correspond with different types of assignment. Below an attempt to list some of the more common types of assignment.

The overview also tries to illustrate, which **Get**, **Set** and **Use** commands might be called while performing the assignment.

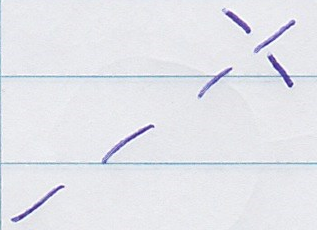
Because assignments might occur commonly, an implicit, simplified notation is suggested here.

|  |
| --- |
| **Value Assignment** |
| **Get Value 🡪**  **Set Value 🡨**  *Aims to copy a value from one object to another.*  *The result might be:* |
| Object Assignment |
| **Get Object 🡪**  **Set Object 🡨**  *Tries to let one symbol point*  *to the object indicated by the other.*  *So attempts to yield over the object aspect.*  *The result might be:* |
| Class Assignment |
|  |
| **Use As Class (≈ Get Object) 🡪**  **Set Class 🡨** |
| *Tries to use one object as the class for another.*  *The result might be:* |

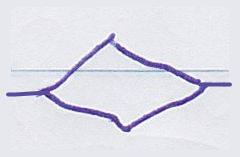
In this assignment notation the line type (dashed, wavy) might indicate which aspect would be yielded over.



The access mark (the smaller line crossing the longer line) might indicate direction.



The diamond shape in the middle would symbolize a call to an assignment command.



### Cross-Aspect Assignments

Perhaps a more common way to use the **Class** aspect in an assignment might be to **Get** the **Object** aspect from one reference and assign it to the **Class** aspect of another reference. But it might also be done the other way around: attempting to get the **Class** aspect from one reference and assign it as the **Object** of another reference. That might be called *Class-to-Object* assignment or a *Class-Get* *assignment* perhaps. Less conventional ways of yielding over aspects like that, might be called *cross-aspect* assignment and they might be used with the notation below.

Also noteworthy might be, that there could be two ways to get a **Class** aspect: **Get** the **Class** aspect of an *object* or **Get** the **Class** aspect of a *reference*, since those seem to be distinct.

|  |
| --- |
| **Reference-Bound Class to Object Assignment** |
| **Get Reference-Bound Class 🡪**  **Set Object 🡨**  *The result might be:*    *The object reference on the right may now point to the class of the object reference on the left.* |
| **Object-Bound Class to Object Assignment** |
| **Get Object-Bound Class 🡪**  **Set Object 🡨**  *The result might be:*    *The object reference on the right may now point to the class of the object on the left.* |

### Ideas about Assignment

There were some ideas let go of in the main description. An attempt is made here to describe them, in case they are still used somewhere or may be reconsidered later.

#### Opposite Direction Indicators?

There used to be an idea to flip the indication of direction in assignment notation.

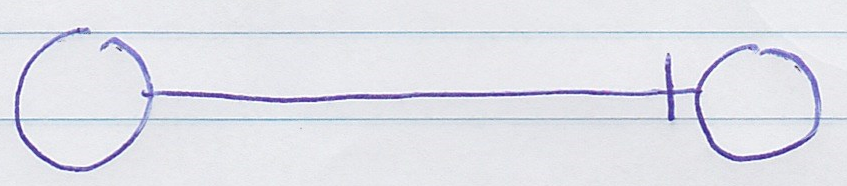
**

**Get Object 🡪**

**Set Object 🡨**

(opposite direction)

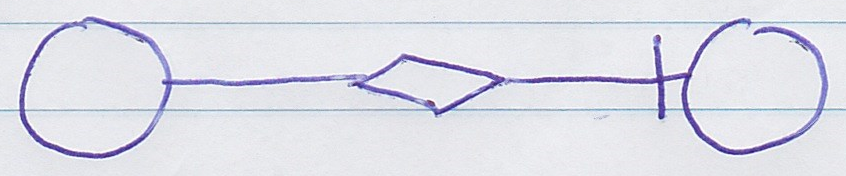
Access marks seem to indicate direction in this language. It may make sense that the access mark might indicate the assignment direction and that the assignment direction means from source to destination. But what might make less sense is when the result of the assignment would seem to be in the opposite direction of the assignment direction:



*Result of the assignment*

Perhaps it can be imagined seeing live changes in a diagram, assignment calls be executed live, seeing their effect, it would perhaps not be so intuitive when lines would seem to flip direction so many times.

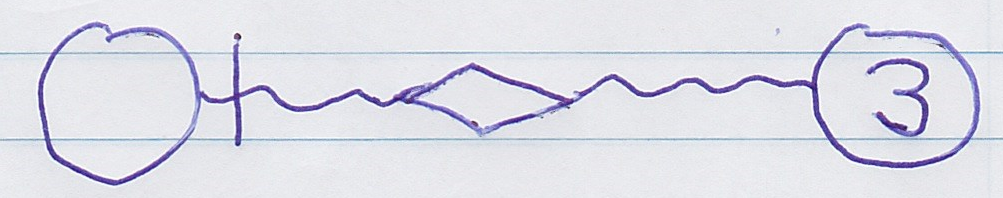
That is why the suggestion was made to make the assignment look more like the direction of the result:



**Get Object 🡪**

**Set Object 🡨**

The idea to put the mark at the opposite end, might have some intuitivity to it, when looking at value assignments.

**

**Get Value 🡪**

**Set Value 🡨**

*(opposite direction)*

Since this would sort of indicate the direction of the flow of the data. But the price of inconsistency might not be worth that intuitivity, and might actually be less intuitive.



**Get Value 🡪**

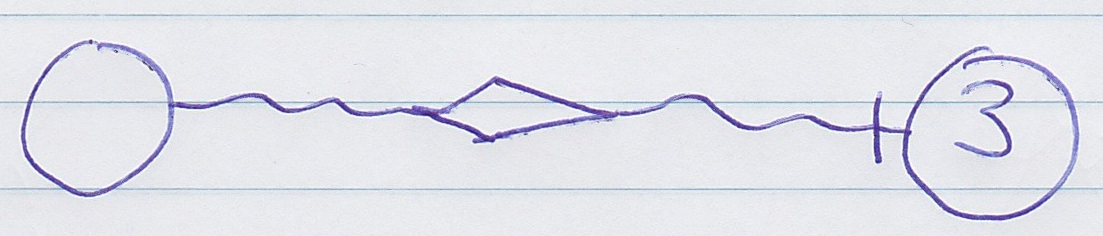
**Set Value 🡨**

In the main idea the access mark sort of points in the direction of what the value is going to be, rather than where the value will go.

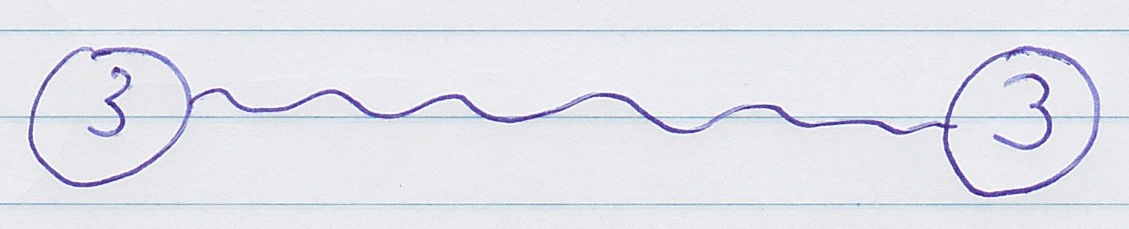
That said, there is a little bit of doubt. That is why this alternative idea is still described here.

#### Leaving Out Diamonds for Value Assignment?

Currently the idea might be to express value *assignment* and value *correspondence differently:*

**

*Value assignment*



*Value correspondence*

In the past there was the idea, that value lines might always mean assignment. So that assignment might not need a diamond shape:



*Value assignment?*

But this might conflict with wanting to express value assignment and value correspondence distinctly and be consistent with the rest of the language.

Then again: if value correspondence would have a direction (indicated with an access mark), what else could it mean but a value assignment?



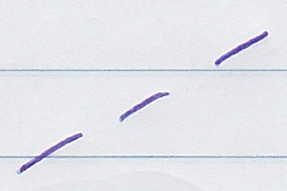
*Value assignment?*

So there still seems to be some doubt around the idea. A choice was made to make a notation leading with less implicit rules and more equal ways of symbolizing things across the language.

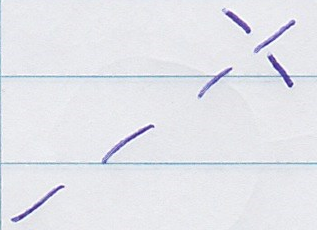
A line would mean correspondence:



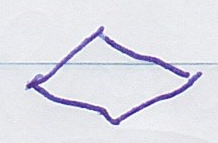
Line type could indicate the system aspect, like the value, object or class:



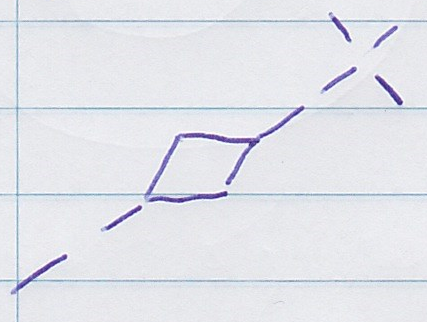
Access mark may mean direction:



And a diamond shape might mean a call:



An assignment call in this case:



The only implicitness in the assignment notation that might be left, may be that the assignment call was simplified to just a diamond shape, instead of for instance showing parameters inside it.