|  |
| --- |
| Circle Language Construct Drafts | System Objects |

## System Command Call Notations Construct Drafts

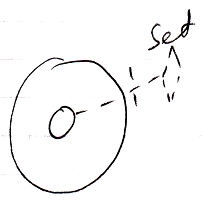
### System Interface Command Call Notation

There might be other options for displaying a call to a system command that are laid out in the articles that follow.

### Simplified System Interface Calls

System command calls might be so fundamental, that they may the most basic notation that still fits within the language, even though it can get a more complex notation when fully drawn out.

Here is an example of the simplified system command call notation:



The line type indicates the aspect.

A word like **Get**, **Set** or **Use** indicates assignment direction.

The access mark indicates the object of which the system command is called.

Leaving out the access mark may make the notation even simpler. The access mark may not be required in this particular notation, but other notations are later derived from this notation which require the access mark for disambiguation.

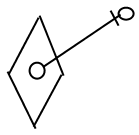
Here follow the basic notations for all the system commands.

|  |  |
| --- | --- |
| Object Get | Object Set |
|  |  |
|  |  |
|  |  |
| Use As Class | Use Reference As Class |
|  |  |
|  |  |
| Class Set |  |
|  |  |
|  |  |
| Object-Class Get | Reference-Class Get |
|  |  |
|  |  |
|  |  |
| Value Get | Value Set |
|  |  |
|  |  |
| Clone (2) Get | Clone (2) Set |
|  |  |
|  |  |
|  |  |
| New | Annul |
|  |  |
|  |  |
| Execute |  |
|  |  |
|  |  |
|  |  |
| Add | Remove |
|  |  |

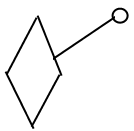
### System Interface Calls with Arguments

System commands can use a simplified argument notation.

Explicit notation of an argument of a command:



can be simplified for system commands:



The simplified notation is only used for system commands, because system commands are so fundamental that they need a notation that is as simple as possible.

The general use of this notation will be explained with **Object Get** as an example. **Object Get** will probably never use this notation, but it is easier to explain this way.

Here is the basic notation of an **Object Get** system command:

|  |
| --- |
| Object Get |
|  |

The **Object Get** command's return value is drawn out as follows, using the simplified argument notation for system commands:

< Add comparison with system interface call notation? >

|  |
| --- |
| Object Get |
|  |

|  |
| --- |
| Object Set |
|  |

|  |
| --- |
| Object Set |
|  |

< A bit more info? >

|  |  |  |
| --- | --- | --- |
| Object Get | Object Set |  |
|  |  |  |
| Object Get 🡨  (Object Set 🡪) | (Object Get 🡪)  Object Set 🡨 |  |

Yielding over **Class** aspect:

|  |  |  |
| --- | --- | --- |
| Use As Class | Use Reference As Class | Class Set |
|  |  |  |
| Use As Class 🡨 (Class Set 🡪) | Use Reference As Class 🡨  (Class Set 🡪) | (Class Get 🡪)  Class Set 🡨 |
|  |  |  |
| Object-Class Get | Reference-Class Get |  |
|  |  |  |
| Object-Class Get 🡨  (Class Set 🡪) | Reference-Class Get 🡨  (Class Set 🡪) |  |

Yielding over between **Class** aspect and **Object** aspect:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class Set | Reference-Class Get | Object-Class Get |  | |
|  |  |  |  | |
| (Use As Class 🡪)  Class Set 🡨 | Reference Class Get 🡨  (Object Set 🡪) | Object-Class Get 🡨  (Object Set 🡪) |  | |
|  |  |  |  | |
|  |  |  |  | |
| Value Get | Value Set |  | |
|  |  |  | |
| Value Get 🡨  (Value Set 🡪) | (Value Get 🡪)  Value Set 🡨 |  | |
|  |  |  | |
| Clone (2) Get | Clone (2) Set |  | |
|  |  |  | |
| Clone (2) Get 🡨  (Clone (2) Set 🡪) | (Clone (2) Get 🡪)  Clone (2) Set 🡨 |  | |
|  |  |  | |
|  |  |  | |
| New |  |  | |
|  |  |  | |
| (Use As Class 🡪)  New 🡨 |  |  | |
|  |  |  | |
|  |  |  | |
| Add |  |  | |
|  |  |  | |
| (Object Get 🡪)  Add 🡨 |  |  | |

Most of the notations above will never be used. Only the notation for the **New** and **Add** arguments are common.

#### Class Assignment Extended

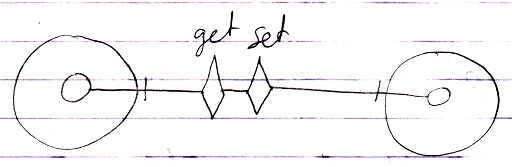
This creates the following notations of the different ways to use a class:

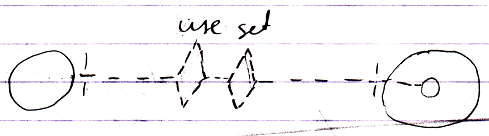
|  |
| --- |
| Class Assignment |
|  |
| Use As Class **(≈ Get Object)** 🡨  Class Set 🡪 |
|  |
| Class to Object Assignment |
|  |
| Class Get 🡪  Object Set 🡨 |
|  |
| Class Get – Class Set Assignment: |
|  |
|  |
| Class Get 🡨  (Class Set 🡪) |
|  |
|  |
| (Class Get 🡨)  Class Set 🡪 |

The **Class Get** notation and the **Class Set** notation both have the same effect, only in one case it is the **Get** that is explicitly displayed and in the other case it is the **Set** that is explicitly displayed. It is not yet clear which is the best one.

### Explicit Get & Set

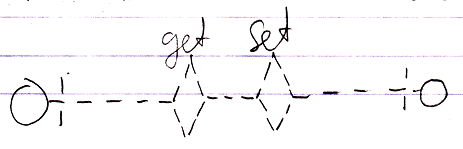
The following are examples of explicit **Get** and **Set** notations. It works for any **Get**, **Set** or **Use** that you want to combine together.





The line types of the system commands express which aspect a command is called upon. The line in between the two system commands is basically a line type that looks best in that situation: if the whole operation is typical to the use of the **Class** aspect you could make it dashed, if the operation is a more uncommon cross-aspect action, then you could just use a solid line by default.

When you look at the aspects introduced so far, the only time you may need the explicit **Get** & **Set** notation is in a **Class Get – Class Set** assignment.



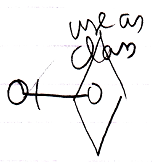
Whenever a simpler notation is available, such as an assignment notation, the simpler notation is used, however, alternative notations can *always* be used to simply express a point in *abstract* diagram notation.

### Explicit Get & Set Arguments

This article sheds light on *explicit Get & Set argument* notations. Several notations are introduced. It is not clear yet which is the best one.

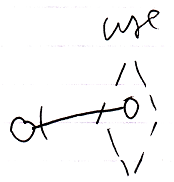
In the diagrams below, things are often implicitly notated. However, the notation of the **Get** and **Set** arguments is always *ex*plicit.

The most explicit way to draw out a call to a system command with the argument explicitly notated is as follows:



It is notated as calling the overloaded **Use As Class** command with its argument filled in.

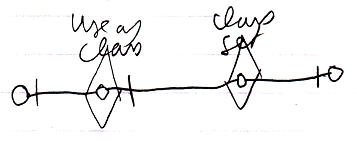
But this notation is also possible:



Then you express the aspect *implicitly* using a different line type for the diamond. You also leave out the aspect from the identifier.

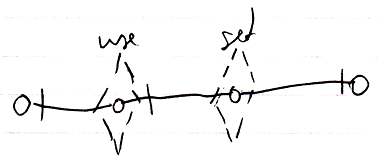
So the aspect can be expressed with a diamond line type or by explicitly expressing the aspect in the identifier.

**Get**, **Set** and **Use** commands are never used on their own. They always need another counterpart. The notation below shows two system command counterparts with explicit arguments:



The **Class Set** argument is connected to the **Use As Class** argument.

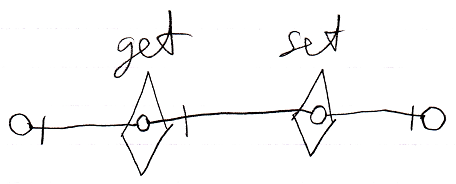
The following notation expresses the same thing:



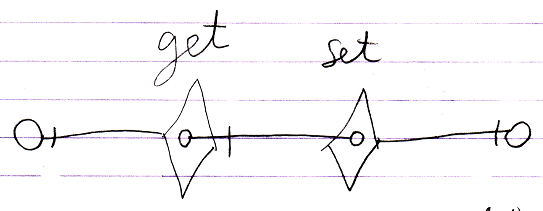
But now the aspect is implicitly expressed by the line type of the command calls.

Note that all explicit **Get** and **Set** argument notations use a solid *object* line between arguments, because that is the part that is *explicit*. Other parts of the notation are implicit sometimes.

If only the object aspect is addressed, the picture looks like this:



There is also the following weirder possibility to display the **Get** and **Set** arguments:



But the odd thing about that notation is that there is no connection between the **Get** and **Set** arguments and the real **Get** and **Set** targets. The notation looks so strange, that we will probably never go with that.

The main point is: the diagrams above show the explicit expression of the arguments of **Get** and **Set** calls.

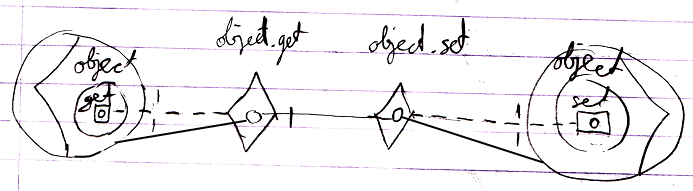
Explicit display of **Get** and **Set** arguments is hardly ever used, because there are simpler notations available, such as the assignment notation or if you must, the explicit **Get** & **Set** notation:

|  |
| --- |
| Assignment |
|  |
| Object Get 🡨  Object Set 🡪 |
|  |
| Explicit **Get** & **Set** |
|  |
| Object Get 🡨  Object Set 🡪 |
|  |
| Explicit Get & Set Arguments |
|  |
| **Object Get 🡨**  **Object Set 🡪** |

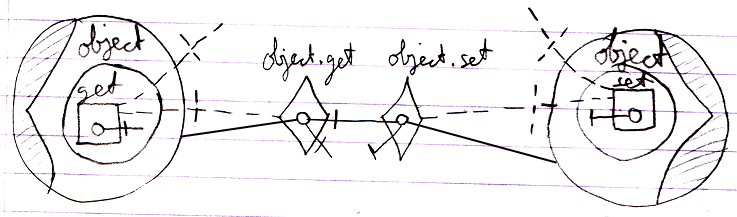
You would only use explicit display of **Get** and **Set** arguments in abstract diagram expression when you just want to make a point.

### Explicit Display of Assignment

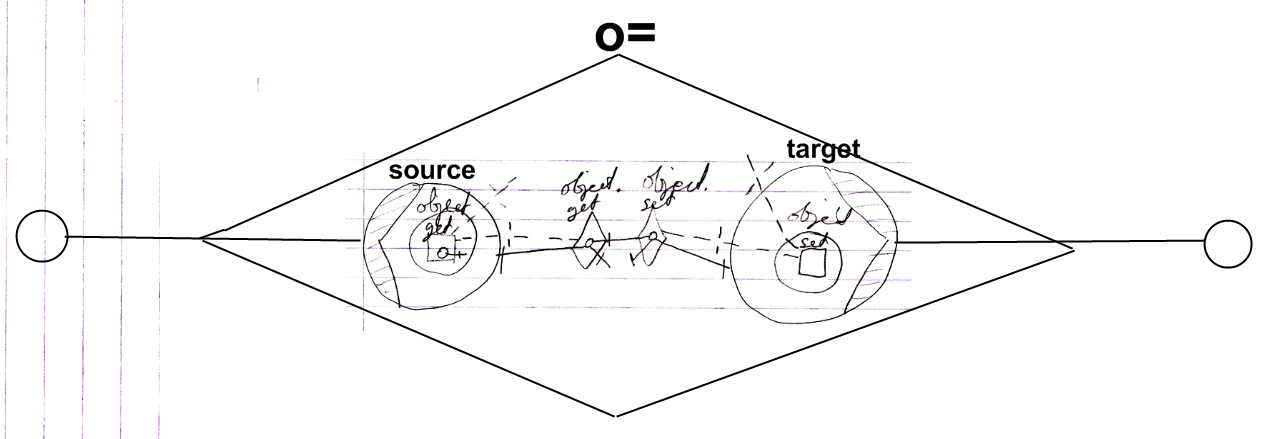
The main thing explicitly displayed here is that the *system interface* is opened up, showing explicitly what aspects and system commands are called. The following picture demonstrates this:



The picture that follows goes even further. It also displays *access connectors*:



Perhaps you could go even further in discovering more things to explicitly display.



But of course a more practical notation is:

