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

## System Command Call Notations

### Introduction

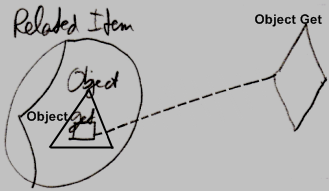
There are several notations for calling a system command. They will be covered in the articles that follow.

You will usually not see any direct calls to **Get**, **Set** and **Use** commands. Those system commands are usually called indirectly by *assignment* commands.

But knowing how to use **Get** and **Set** commands, instead of a single assignment call, gives you a low-level understanding of assignment. This low-level understanding is important in order to create alternative forms of assignment, that are not covered by any of the assignment notations.

### System Interface Command Call Notation

One of the notations for system command calls is calling upon the command definition in the system interface:



This however requires the system interface to be shown. Then remains the question whether to show the other members of the system interface as well, or only to show the members of the system interface, that are actually called.

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

### Simplified System Command Call Notation

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

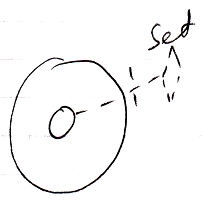
There are system commands that apply to object references and to the object itself. System commands, that apply to an object reference are displayed here with a parent around the targeted reference:



System commands, that apply to the object itself are displayed without a parent around the targeted object.



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 |
|  |  |

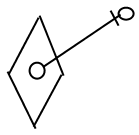
Usually a **Get** command or **Use** command is immediately followed by a **Set** command. But in the overview above you see isolated **Get**, **Use** and **Set** calls. You will probably not see this happen in actual programs, but the notation above is used later as a base for other notations.

### System Command Call with Argument Notation

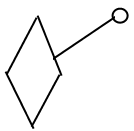
*Basic System Command Calls* showed the basic system command call notation, but most system command calls are incomplete without an argument.

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 has a return value: the object. It is an *output* argument. It is not displayed in the picture above, but it should be there to transfer the **Object** aspect to another symbol. This is drawn out as follows, using the simplified argument notation for system commands:

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

In the picture above the output argument of the **Object Get** call is filled in.

**Object Get** is called upon the smaller circle contained by the larger circle.

But next, an **Object Set** is *implicitly* called upon the smaller circle in the corner.

The diagram above implictly causes an **Object Set** to be executed upon the smaller circle in the corner.

Something similar happens in case of an **Object Set** command. Here is the basic notation of an **Object Set** command call:

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

And here the argument of the **Object Set** command is filled in:

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

This causes an implicit **Object Get** on the smaller circle in the top-right corner, followed by the explicit **Object Set**, targeting the smaller circle contained by the larger circle.

So the basic display of a basic system command call with an argument filled in, implictly causes another system command to be called as well.

Below you will find an overview of all the system commands with their argument filled in using the simplified argument notation for system commands. Which system commands are implicitly called, is also made clear in the overview.

|  |  |  |
| --- | --- | --- |
| 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.

System command calls to **Get**, **Set** or **Use** are often not visible, because they are usually implied by assignment calls. But for alternative types of assignment, that do not have a standard assignment notation, you can explicitly display **Get**, **Set** and **Use** calls. Only when a simpler notation does not suffice, a **Get**, **Use** or **Set** command are explicitly displayed. This basically only happens in unusual cross-aspect assignments. In abstract diagram expression though, you can always use the explicit notations to simply make a point. When you look at the aspects introduced so far, the only time you might need the ‘system command with argument’ notation for **Gets** and **Sets**, is in a **Class Get – Class Set** assignment. This creates the following notations of the different ways to use a class:

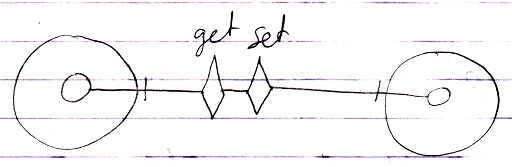
|  |
| --- |
| Class Assignment |
|  |
| Use As Class 🡨  Class Set 🡪 |
|  |
| Class to Object Assignment |
|  |
| Class Get 🡨  Object Set 🡪 |
|  |
| Class Get – Class Set Assignment: |
|  |
|  |
| Class Get 🡨  (Class Set 🡪) |
|  |
|  |
| (Class Get 🡨)  Class Set 🡪 |

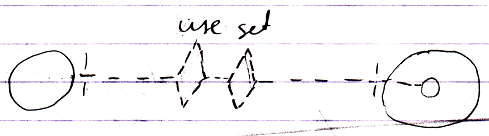
The **Class Get** notation and the **Class Set** notation both have the same effect, only in one case it is the **Get** that is explicitly displayed and in the other case it is the **Set** that is explicitly displayed. It is not yet clear which is the best one.

### Explicit Get & Set Notation

Assignment has a special, simplified notation covered in the *Assignment* articles. But you can also display the **Get** and **Set** of the assignment explicitly. This notation is only required when there is no simplified notation available for what you want. However, explicit notation of **Get** and **Set** can always be used in *abstract* diagram expression just to make a point.

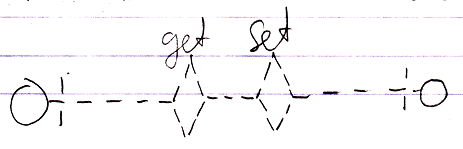
The following are examples of explicit **Get** and **Set** notations. It works for any **Get**, **Set** or **Use** that you want to combine together.





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.



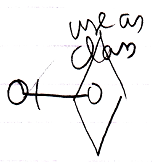
Whenever a simpler notation is available, such as an assignment notation, the simpler notation is used, however, alternative notations can *always* be used to simply express a point in *abstract* diagram notation.

### Explicit Get & Set Arguments Notation

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

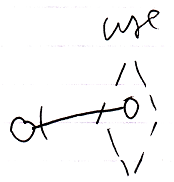
In the diagrams below things are often implicitly notated. However, the notation of the **Get** and **Set** arguments is always *ex*plicit.

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.

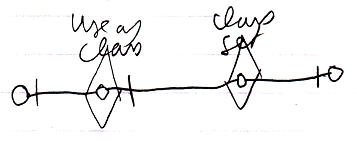
But this notation is also possible:



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

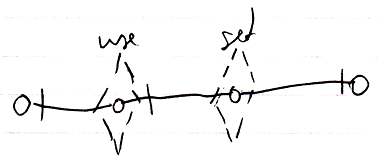
So the aspect can be expressed with a diamond line type or by explicitly expressing the aspect in the identifier.

**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.

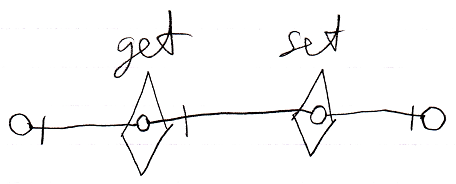
The following notation expresses the same thing:



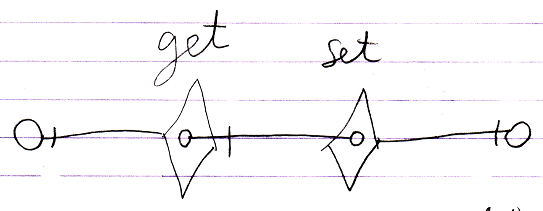
But now the aspect is implicitly expressed by the line type of the command calls.

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 adressed, 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.

### Completely Explicit Display of Assignment

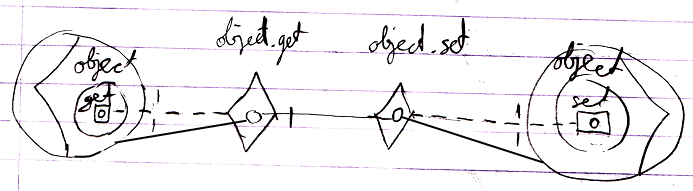
An assignment has a simple notation:



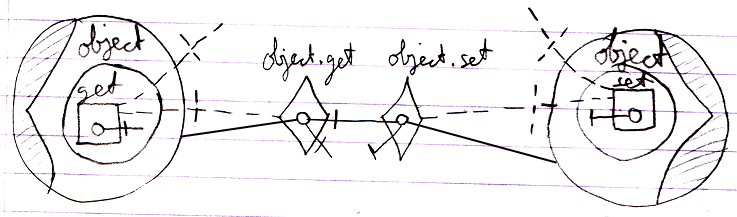
But it is an implicit notation. Assignment needs such a simple notation, because it is one of the most basic kinds of commands in computer technology. Other articles have demonstrated how assignment can be displayed in a more explicit way. The current article goes as far as to display assignment completely implicitly.

The point of this article is to demonstrate how assignment works at system-level and also to demonstrate how impractical the explicit notation is.

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:

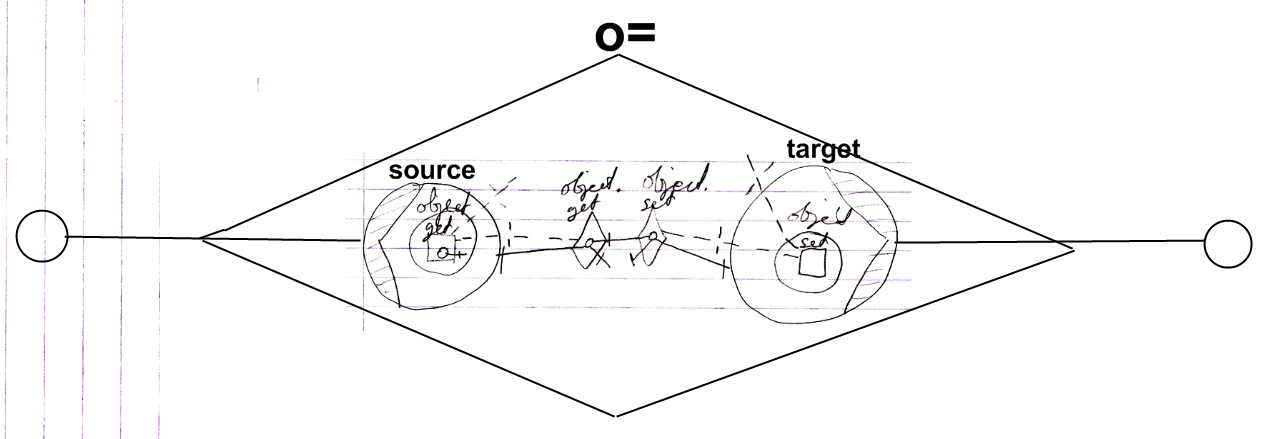


The picture that follows goes even further. It also displays *access connectors*:



Perhaps you could go even further in discovering more things to explicitly display, but the main point is: you have now seen how system operations work internally and you can also see that it is highly unpractical to express all the assignments explicitly like that.

You could also explicitly display the assignment call, which basically puts the whole thing inside a command:



But ofcourse a more practical notation is:

