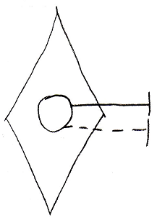
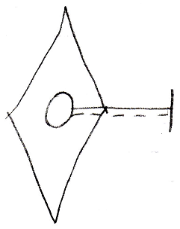
|  |
| --- |
| Circle Language Spec: Black Boxes |

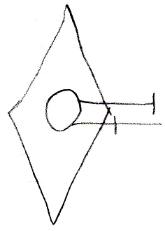
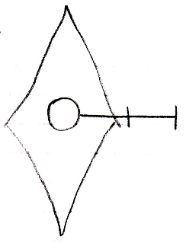
## Simplified Access Control Notation

There are only a couple of ways, in which expression of access control connectors in a diagram will be simplified.

The first method is merging the access connectors of different aspects together.

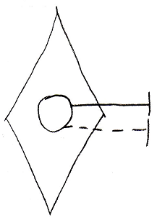
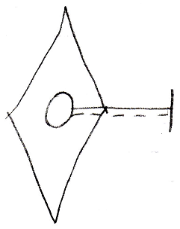
 

The second method is merging the access connectors of a Get and Set together.

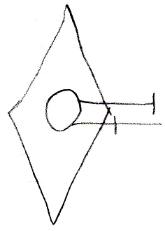
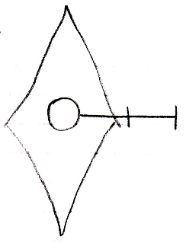
 

There will be rules about when you can apply these methods of simplification.

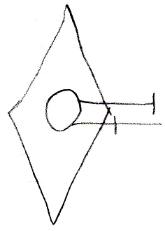
Merging access connectors of different aspects together only happens for access connectors with the same access direction and same access mark position.

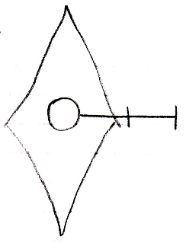
Merging Get and Set connectors happens only when they are both accessible.

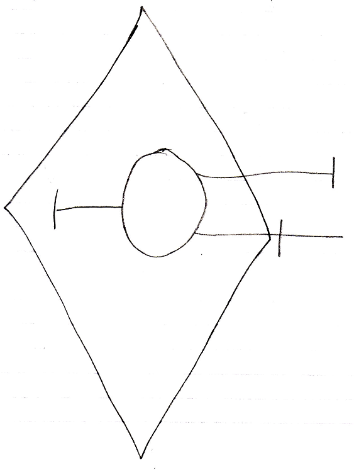
Furthermore, the merging happens in respect to the most likely chronological order of access: Set Public, Get Private, Set Private, Get Public. This means, that two access connectors will only merge if there is nothing chronologically in between them, that is accessible.



The access connectors above will merge, because nothing is chronologically in between.

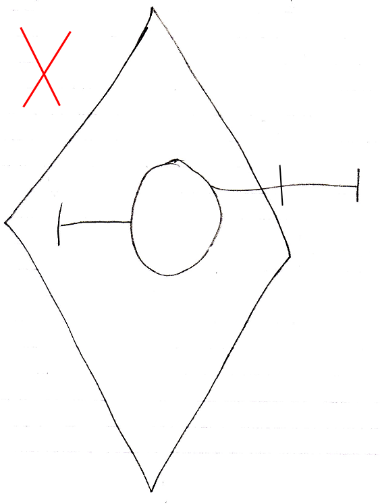


But if something were in between, for instance, a Private Set connector, then the merge would be blocked by it:



In the picture above, the Public Get and Set connectors will not be merged, because the Private Set connector is chronologically in between.

The access connectors do not need to be *positioned* in chronological order in the diagram, but merging connectors together suggests nothing can be in between them, and it is just clearer if you keep them separated when something can get chronologically in between.



Here are all the possible access connectors for parameters, now with the easier to read parameter access control literals and in a more chronological order. Actually, for now, they are not much different from the unsimplified versions, but in the future (from 2008-09-29), when the terms In, Out and Thru are more exactly defined, the terms may very well become a lot simpler.

|  |  |
| --- | --- |
| Object | Command |
|  |  |
|  |  |
| Object Set Public | Object Get Private |
|  |  |
|  |  |
| Object Set Private | Object Get Public |
|  |  |
|  |  |
| Class Set Public | Class Get Private |
|  |  |
|  |  |
| Class Set Private | Class Get Public |
|  |  |
|  |  |
| Value Set Public | Value Get Private |
|  |  |
|  |  |
| Value Set Private | Value Get Public |
|  |  |
|  |  | |
| Clone 2 Set Public | Clone 2 Get Private | |
|  |  | |
|  |  | |
| Clone 2 Set Private | Clone 2 Get Public | |
|  |  | |
|  |  |
| Data Set Public | Data Get Private |
|  |  |
|  |  |
| Data Set Private | Data Get Public |
|  |  |

|  |  |
| --- | --- |
| Execute Public | Execute Private |
|  |  |
|  |  |
| New In | New Out |
|  |  |
|  |  |
| Annul In | Annul Out |
|  |  |

### Loose Ideas

#### Simplified Access Connectors

Could also be called: Access Control Presets.

-----

< Wat wel zo is is dat de standaard access control van system procedures er voor zorgt dat je niet *zo maar* rare dingen kan gaan doen. Erg belangrijk als je effe snel iets programmeert, dat je de echt rare effecten uitsluit zonder extra moeite. >

-----

< 2009-06-29 Bedoelde ik hier oorspronkelijk mee de macro-keywords voor access control? Zoals bijvoorbeeld Public Read-Only Value ofzo? >

-----

<

I don’t have a special notation for publicity of Set procedures.

Set public or not is kind of like read only or not. Maybe I need a read only notation.

Write only is also common, though. For instance of procedure reference input parameters.

>

-----

<Also cover constants>

-----

When you give a system procedure an access controller, access to the procedure or object represented may get a special name and diagram notation.

An object symbol with an accessible State *Read* procedure, but no State *Write*, is called a Read Only Object.

An object symbol with only a Type Read, can be considered just a Type. Not an object or interface.

An object symbol with only an Interface Read, can be considered just an Interface. Not an object or type.

…

Etcetera, work it all out

-----

Standaard access controllers

* Standaard zijn Add en Remove geblokkeerd. Zo zijn er nog meer dingen standaard geblokeerd

De klasse kan zelf initial access controllers aannemen die je ook weer kunt veranderen.

-----

Something is constant if it can be set by programmers , but not by users.

-----

< By default everything is public. When studying the access configuration of a symbol you might only want to see what differs from the default >

-----

< The standard access modifiers for system procedures of procedure symbols directly inside a procedure symbol are different from procedures directly inside object symbols. For instance, Symbol Get for Reference of a clause is Private by default. Symbol Get for Reference for an object member is Public by default >

-----

A call is by default private.

-----

The access controllers of the procedure’s sub objects determine which purpose they serve. Maybe I should write a section about that in Access Control.

-----

Required and Optional parameters as well as In, Out and Thru need to be rediscussed in Access Control, because it may be so that it is required for you to Object Set a parameter, while the procedure changes the state of the target object. This is like thru too, but a totally different fashion and there are more ways of throughput.

-----

If a configuration of any sort is a common one, it might be nice to make something like a typedef so you give the configuration a name of its own, Like if you commonly use ‘Public Up 1 Module Up 2’, you might want to give it a different name.

Consider combining the concepts Presets, Macro-Keywords and just a brainstorm about nice-to-have keywords for common access control situations.

Best example: Public Read-Only Value. Read-Only Value.

You do not even have to introduce completely sound systematics, you can also just shed light on the general concept.

Those 'macro' access modifier keywords are not very important.

There may be an analogy between macro keywords and a C++ typedef.

(perhaps only for names)

(I think this issue was also temporarily put in the Parameters chapter originally.)

-----

But what about ‘making a command Private’? What do you mean by that? You can access control the Execute system aspect of commands, but that does not make other aspects of a command object private, such as the Object Set aspect (turning the command symbol into a command reference).

-----

Should there not be something implied when a connector is not there: in a friend relation this will create a lot of connectors… so no connectors should mean everything is accessible in a friend or one connector should be the placeholder for any other connector… or some common connectors and others (such as Class Get implied)

-----

In definitions, publics could also be differentiated from privates by privates not getting access connectors? Well… friends can access them, so they need access connectors…

-----

When you have to display a lot of connectors, you might summarize them, and when you float over them, they expand into options.

Maybe first you have an object connection line without a connector. You float over it, it splits up into an class, interface, object and value line, you float over one of those, it splits up into for instance Object Get and Object Set. I know that does not cover all the options, but it might be an idea to display the connector summary and fan it out when you float over it.

-----

Explicit display that something is private is optional. Explicit display, that something is public could be made mandatory, because is it not overhead in notation: it clearly indicates a connection point.

-----

If you do not see any access connectors, do you assume, that everything is accessible, or that nothing is accessible? That is a good question for the display of access connectors. But not essential to cover.

-----

Questions:

- Can you make it an option to only show befriended commands inside an object, instead of friend and non-friend commands.

I am not sure yet. It is not clear.

-----

Another fact:

One of the most important things to access control is an object’s being accessible at all or having read-only access to it. Another important access controlling capability is to only be able to retrieve (and set) the Value of an object. Perhaps, if you state that Set is public it automatically means, that Get is also public, and if you state that Get is public, it automatically implies that Set is private.

> Perhaps it is interesting when one user writes a value, but other users can not read it out, but that is sort of outside the scope: that is user access control. System access control is covered here.

-----

Perhaps that way the number of possible access controllers becomes more limited, so you do not have to display each possible access connector: Object Get implies any kind of connector.

Access controlling for just allowing a Class Get is stupid as well: who wants that? Who wants to only be able to access the reflective data of an object, but not the object itself?

-----

The notation is simply drawing the symbol a dotted line or a dashed line, which makes it only usable as an interface or class. How you should express something only being usable as a class but not as an interface is not clear to me. I do not think I really need that and I do not think I need to clear that up.

But whatever, this is for later.

-----

Also useless: allowing Set but not allowing Get… if you can already write it, why not allow reading it? Is there any point to that? The important thing is write-protecting, or not access at all… but read-protecting, while you could write to it? It seems unlogical.

That seems a whole lot of Get and Set purposes. In practice, the set up of a symbol’s system procedures is not that complicated. There’s a standard setup: the most usual system procedure setup. Only deviations from the standard are additionally denoted.