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

## Public & Private Connectors, Connections & Assignments

### Public & Private Connectors

The following articles display the various access connectors an object can have.

#### Object Connectors

|  |  |  |
| --- | --- | --- |
| Public Object Get | Private Object Get | Friend Object Get |
|  |  |  |
|  |  |  |
| Public Object Set | Private Object Set | Friend Object Set |
|  |  |  |

#### Class Connectors

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Public Use As Class | | Private Use As Class | | Friend Use As Class | |
|  | |  | |  | |
|  | |  | |  | |
| Public Class Set | | Private Class Set | | Friend Class Set | |
|  | |  | |  | |
|  | |  | |  | |
| Public Reference Class Get | | Private Reference Class Get | | Friend Reference  Class Get | |
|  | |  | |  | |
|  | |  | |  | |
| Public Object Class Get | Private Object Class Get | | Friend Object Class Get | |
|  |  | |  | |

#### Value Connectors

|  |  |  |
| --- | --- | --- |
| Public Value Get | Private Value Get | Friend Value Get |
|  |  |  |
|  |  |  |
| Public Value Set | Private Value Set | Friend Value Set |
|  |  |  |

#### Data Connectors

|  |  |  |
| --- | --- | --- |
| Public Data Get | Private Data Get | Friend Data Get |
|  |  |  |
|  |  |  |
| Public Data Set | Private Data Set | Friend Data Set |
|  |  |  |

#### Clone Connectors

|  |  |  |
| --- | --- | --- |
| Public Clone (2) Get | Private Clone (2) Get | Friend Clone (2) Get |
|  |  |  |
|  |  |  |
| Public Clone (2) Set | Private Clone (2) Set | Friend Clone (2) Set |
|  |  |  |

#### Existance Connectors

|  |  |  |
| --- | --- | --- |
| Public New | Private New | Friend New |
|  |  |  |
|  |  |  |
| Public Annul | Private Annul | Friend Annul |
|  |  |  |

#### Execute Connectors

|  |  |  |
| --- | --- | --- |
| Public Execute | Private Execute | Friend Execute |
|  |  |  |

#### Add And Remove Connectors …

…

#### Remarks

##### Object-Bound & Reference Bound Connectors

In the overviews, object-bound connectors are displayed with a separate shape without a parent, while reference-bound connectors are displayed with the target inside a parent.

##### Object Class Connectors

There is no Object Class Set connector, because the object class can not be freely set; it can only be set on object creation.

##### Class Get and Use As Class Connectors

For the Class aspect, there is a distinction between Use As Class and Class Get. They are different. Use As Class makes you use one symbol as the class for another symbol. That is how classes are most commonly used. That would be the result of an Object Get on the source object, followed by a Class Set on the target object. No Class Get is involved in a Use As Class action, which you might expect if you draw a parallel to the Object aspect.

Class Get gets a reference to the class object of an object. Traditionally this would be considered getting an object’s reflective data. You can use the reference to the class to your own discretion. Class Get is far less common. Use As Class gets a fundamental notation, whereas Class Get has to make dues with the ‘system command’ notation.

This is due to the behavior of the Class aspect. This is different behavior compared to the Object aspect. You just have to remember, that each aspect has different behavior, because it has a different purpose.

The funny thing about access controlling Use As Class, is that you are actually access controlling the Object Get for specific use in a class assignment.

##### Value Connectors

Note that for the Value aspect the directions of Get and Set are switched compared to the Object aspect

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

This is because for the Object aspect, the access mark indicates the direction of the *resultant line*, while for the Value aspect, the access mark indicates the direction of the *assignment*. You have to remember, that each aspect has its own unique behavior.

##### Data Connectors

You can access control sub-objects, to apply some sort of write-protection to the sub-object. But that does not write-protect the sub-objects of the sub-objects. To collectively write-protect any sub-object, you can access control the object’s Data aspect.



A preliminary notation for Data access is a *triple wavy line* (‘triple’ expressing multitude, ‘wavy’ expressing values or data).

##### Clone Connectors

The Value aspect is about copying the direct value of the object. But you can also copy the values of sub-objects up to a certain cloning depth. The could be any number of levels up to which the values of sub-objects are cloned. The diagrams in the overview show access connectors for cloning at a depth of 2. That is just an example. Any cloning depth other than two can also be separately access controlled.

Cloning has the exact same notation as the Value aspect, but then with a number near the access mark, to indicate the cloning depth.

|  |  |
| --- | --- |
| Value Get | Clone (2) Get |
|  |  |

Clone is simply thatclosely related to the Value aspect.

##### Interface Connectors & Other Connectors

There are also interface connectors, but *interfaces* as a whole will be cought up with in the *Interfaces* article group. Interface connectors work roughly the same as class connectors. In fact, any other aspect that will be introduced in the future will extend the set of possible connectors.

##### Preliminary System Command Notation

The access connectors using the system command notation have a preliminary notation. The exact notation system commands will get an update in the future.

### Public & Private Connections

This section covers any kind of connection between objects, that are a *result* of accessing system aspects.

#### Object Connections

|  |  |
| --- | --- |
| Public Object Get Connection | Friend Object Get Connection |
|  |  |
|  |  |
| Public Object Set Connection | Friend Object Set Connection |
|  |  |

#### Class Connections

|  |  |
| --- | --- |
| Public Use As Class Connection | Friend Use As Class Connection |
|  |  |
|  |  |
| Public Class Set Connection | Friend Class Set  Connection |
|  |  |

#### Value Connections

|  |  |
| --- | --- |
| Public Value Get Connection | Friend Value Get Connection |
|  |  |
|  |  |
| Public Value Set Connection | Friend Value Set Connection |
|  |  |

#### Clone Connections

|  |  |
| --- | --- |
| Public Clone (2) Get Connection | Friend Clone (2) Get Connection |
|  |  |
|  |  |
| Public Clone (2) Set Connection | Friend Clone (2) Set Connection |
|  |  |

#### New & Annul Calls

New and Annul connections are *calls* to the New and Annul system commands.

There are the following possible New and Annul calls:

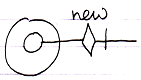
|  |  |
| --- | --- |
| Public New Call | Friend New Call |
|  |  |
|  |  |
| Public Annul Call | Friend Annul Call |
|  |  |

But the *result* of accessing New and Annul are the reference’s being Nothing or Something, which look as follows:

|  |  |
| --- | --- |
| Nothing | Something |
|  |  |

(As with the Set connections of the Object and Class aspects, actually *accessing* the symbol is history and it does not matter anymore if it was Public of Friend access that caused the result, so you do not need to see it in the diagram of Nothing and Something whether it was established through Public access or Friend access.)

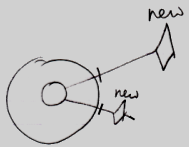
The access *connector* notation of New and Annul were preliminary:



And here it becomes apparent why. The following diagrams show New and Annul calls and also the access connectors displayed right in the picture:

|  |  |
| --- | --- |
|  |  |

An access connector stands for a potential connection and should look like the potential connection. In the pictures above, the connector does not look like the actual connection, so the notation may need to change in the future. A potential connection is usually expressed as a loose end, so the potential system command call should also be expressed as a loose end somehow, so perhaps the following notation for access controlling a system command would be better:



For now the notation of system command connectors is open to discussion.

#### Execute Connections

< TO DO: You have to address the details in *Using Command Symbols* and the preliminariness of the system command notation must eventually be removed. And then you have to display just the actual connections (executable references to the command) that are already visible inside the more complex pictures now shown below. >

Access controlling the Execute system command for command objects may seem like access controlling the ability to call a command. But that is not true. You can access control a command’s Use As Class connector to make a command callable or not. You do not need to access control the Execute system command for that.

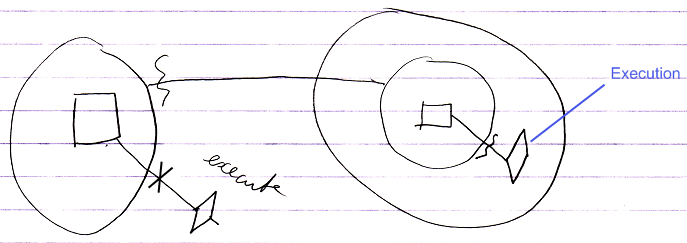
The essence of it is, that each system aspect can be separately access controlled. They all get their own access connector. The uses and implications of access controlling the Execute command are less important for now. Access controlling commands is covered later separately.

The Execute access connector was introduced as follows:



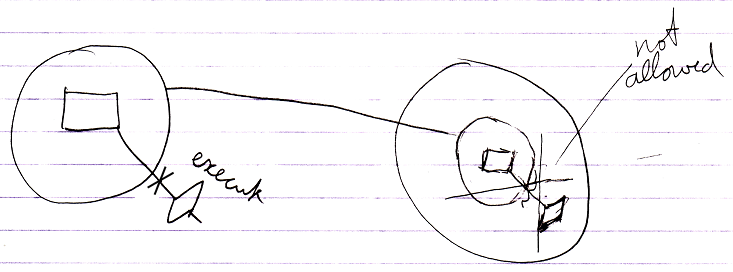
The result of access controlling the Execute command is that the command can only be executed by Friends. The command can not be referenced by a diamond shape (an *execution*) unless it is from a Friend object. Access controlling the Execute aspect *disallows* execution of the command.

So this is allowed (see **Execution** in the diagram):

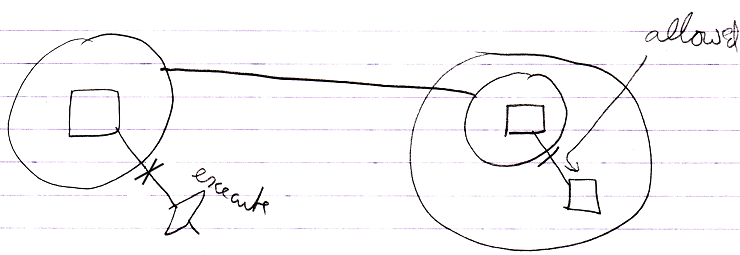


It is allowed, because of the friend declaration on the left.

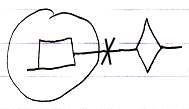
When you take away the friend declaration, execution is no longer allowed:



But *non-executing* references to the command *are* allowed:



The notation for access controlling system commands is not final yet. But because an access connector should be a depiction of a potential connection, the access connector for the Execute aspect might eventually have to look as follows:



With not even the word ‘execute’ in it anymore.

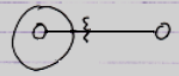
#### < Add And Remove Connections … >

…

#### Remarks

##### Friend and Private Connections

Because there is not distinction between a Friend connection and a Private *connection*, Friend connections:



are called Private connections as well.

##### Object Set Connections

The Public and Friend Object Set connections look the same.

|  |  |
| --- | --- |
| Public Object Set Connection | Friend Object Set Connection |
|  |  |

Public and Friend does not matter for outward connections. Public or Friend access only mattered when *establishing* the outward connection. Outward connections are passive connections: direct connections to the target object. *Inward* connections, though, cause a Get command on each consult of the source symbol, so in that case an actual *access* takes place, and the distinction between Friend or Public access does matter.

##### Class Connections

Use As Class and Class Set are covered here. Class Get is not covered here, because that is only used in an assignment, and assignments are addressed in other section: *Assignment of System Aspects*.

##### Value Connections

Note that the indication of *direction* for Get and Set is switched for the Value aspect, compared to the Object aspect: Object Get is inward, while Value Get is *outward*.

Value connections are shown here in the section *Connections to System Aspects*, and not in the section *Assignment of System Aspects*, but Value connections are assignments, but they do not have an ‘assignment notation’.

What you can also notice is that outward Value connections do have access marks, while outward connections for the Object and Class aspects do *not* show access marks. This is because *outward* Object and Class connections are passive connections and outward Value connections are *active* connections, because an Value connection always requires an active assignment.

##### Data Connections

A Data connector such as the following:

|  |
| --- |
| Public Data Get |
|  |

is an abstract connector: it represents any possible Get connection to a sub-sub-object.

Even though there is a Data *connector*, there are no Data *connections*. Any Get connection to a sub-object is be considered a Data Get connection.

##### Clone Connections

The diagrams above show access connections for cloning at a depth of 2. That is just an example. Any cloning depth other than 2 could be used.

Cloning has the exact same notation as the Value aspect, but then with a number near the access mark. Cloning is that closely related to the Value concept.

##### Existance & Execute Connections

The remaining aspects are Existance and Execute.

The essence of it is, that all system commands are separately access controlled. They all get their own access connector. The uses and implications are less important.

This part of the documentation displays connections that are a result of accessing system aspects. But for the Value aspect that means the display of an assignment.

And for Existance and Execute that also means something different.

Here you see again, that all system aspects behave totally differently.

### Public & Private Assignment

This section demonstrates how accessing system aspects looks in assignment notations.

#### Object Assignment

|  |  |
| --- | --- |
| Public Object Get  in Assignment | Friend Object Get in Assignment |
|  |  |
|  |  |
| Public Object Set in Assignment | Friend Object Set in Assignment |
|  |  |

#### Class Assignment

|  |  |
| --- | --- |
| Public Use As Class in Assignment | Friend Use As Class in Assignment |
|  |  |
|  |  |
| Public Class Set  in Assignment | Friend Class Set  in Assignment |
|  |  |

#### Class Get Connection

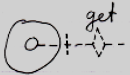
At first glance you would guess, that Class Get in assignment looks something like this:



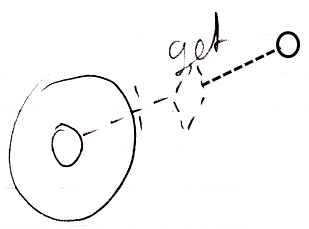
But that is Use As Class. The smaller circle on the left becomes the class of the circle on the right, requiring an Object Get, followed by a Class Set. Class Get is far less common: it retrieves a reference to the class of an object, and then you can assign the class to another object.

Using Class Get always requires the system command notation, because the more fundamental notation above is used by Use As Class.

The system command notation is not yet final, though. The Class Get access connector may look something like this:



Yielding over the class from one object to another may look something like this:



Even though the eventual notation does not look like assignment, it still involves an assigment to the object on the right, whose aspect is changed.

Yielding over Class aspect:

|  |  |
| --- | --- |
| Public Reference Class Get Connection | Friend Reference Class Get Connection |
|  |  |
|  |  |
| Public Object Class Get Connection | Friend Object Class Get Connection |
|  |  |
|  |  |

An alternative: assigning Class object to Object aspect of other symbol:

< Change notation. >

|  |  |
| --- | --- |
| Public Reference Class Get Connection | Friend Reference Class Get Connection |
| *(source is inward)* | *(source is inward)* |
|  |  |
| Public Object Class Get Connection | Friend Object Class Get Connection |
| *(source is outward)* | *(source is outward)* |

#### Remarks

##### Only Object and Class Assignments

Only Object and Class assignments are covered here.

Value and Clone assignments were covered in ***Connections*** *to System Aspects*.

The Execute aspect does not involve assignment.

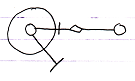
The Data aspect does not work with assignment either.

The Existance aspect may involve assignment but that is not covered here.

So it comes down to Object and Class assignments.

##### Connectors & Assignments

The following diagram shows Object Set in an assignment, and the Object Set access connector:



The access mark in the assignment and the access mark in the connector have opposite directions.

This is because for the assignment notation it was decided, that an the access mark expresses the direction of the assignment, and the connector expresses the direction of the *potential connection*, so the direction of the result. It was decided, that it would be clearer to express the direction of the *assignment* inside the diagram notation, instead of letting the assignment call express the direction of its *result*.