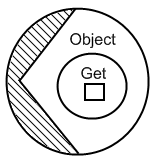
|  |
| --- |
| Circle Language Spec: System Objects |

## Connectors & Connections

### Connectors

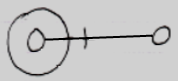
System command definitions are displayed in the system interface:



But more commonly a system command definition is displayed as a *connector*:



A connector represents a potential call to a system command. The connector looks like the result of the system call. The connector above is an Get Object connector. It represents the Get Object system command definition, and it looks like the result of an Get Object:



If a system command is not about establishing a connection between symbols, the connector looks like a system command call itself, for instance for the New command:

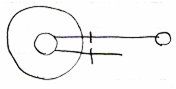


But then more like a loose end, because an unconnected line is displayed at the end of the (potential) New call. An actual New call would look like this:



So a connector looks like the result of a call or otherwise like the call.

An actual connection does not replace the connector*:* a connector is always displayed, because others can still connect to the object as well:



The line at the top is an actual connection.

The line at the bottom is the connector.

Each system command has a connector, so every system aspect has its own set of connectors. The overview below displays the different connectors.

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

### Loose Ideas about Connectors

System Objects,

2009-09-27

Not all system commands are present here yet.

Also: the connectors involved in pointer-to-pointer situations are not present here yet.

JJ

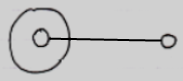
### Connections

This section covers any kind of connection between objects, that is a result of accessing system aspects. First the graphical overview of all possible system aspect connections is displayed. After that the details are covered.

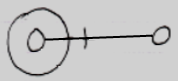
|  |  |
| --- | --- |
| Get Object | Set Object |
|  |  |
|  |  |
| Use As Class | Set Class |
|  |  |
|  |  |
| Get Value | Set Value |
|  |  |
|  |  |
| Get Clone (2) | Set Clone (2) |
|  |  |
|  |  |
| New | Annul |
|  |  |
|  |  |
| Something | Nothing |
|  |  |
|  |  |
| Execute |  |
|  |  |
|  |  |
| Add | Remove |
|  |  |

#### Outward and Inward Connections

An Set Object connection is an outward connection.

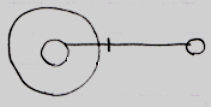


Outward connections do not have access marks. Inward connections do have access marks:



This is not only because outward connections are more common, but also because an outward connection is more passive than an inward connection.

When a symbol is inward connected:



each consult of the symbol on the right, that connects inwards, results in a call to to the Get of the target symbol. So the connection is more active in that each consult of the source object requires a Get on the target object. Inward connections are also called Get connections.

When a symbol is outward connected:



the symbol is directly connected to the target object. It stores the ID of the target object and refers to it directly, without executing a Get on each consult. The outward directed connection may have been established through acccess to a Set command, but that call is history. The connection does not have to pass through to the insides of another object. Outward connections are called Set connections. However, the Set action is history and all that matters is the result, not how it was established.

Passing through to the inside of another object requires the Get command of the target symbol to be executed.

This also conforms to the idea of automatic containment. In automatic containment, outward directed connections are automatically established: automatically drawn out in the diagram without an intervention of a programmer. Inward directed connects were actively established by the programmer. They follow a qualification before getting to the final object. They are more active and display an access symbol and result in active Get’s taking place all the time.

#### Remarks

##### Class Connections

Use As Class and Set Class are mentioned here. Class Get is not covered here, because that is only used in active assignments, not in connections.

##### Value Connections

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

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 is always an assignment.

Value connections are assignments, even though they do not have an ‘assignment notation’.

##### Data Connections

A Data connector such as the following:

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

is an abstract connector: it represents any possible Get connection to any 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 Get Data 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 aspect.

##### Existence, Execute & List

This part of the documentation displays connections that are a result of accessing system aspects. For the Object aspect that means an actual connection. But for the Existence, Execute and List aspects it means something different: more like a *call* or sometimes more like a *state*.