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

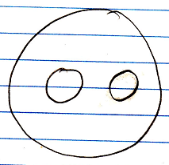
## System Objects Misc Issues

### System Command Calls by User

A user will often execute Gets and Sets and other system commands when connecting symbols together, but they will only see the connectors and the *result* of a Get and Set, but never the explicit Get and Set calls. The system commands are executed as the user builds up a diagram.

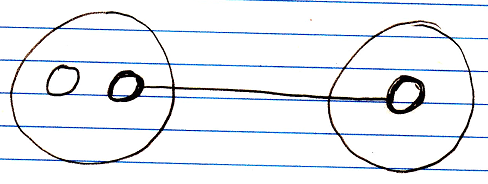
### Objects Floating Around

Objects are never directly accessed. They are always floating around somewhere you cannot touch. You are always accessing an object through an *object reference*. You are always dealing with *references* to objects, never with the object directly.



The smaller, contained circles are *references* to objects, even though the bigger circle seems to be the sole container of the objects themselves. An object does not really contain sub-objects. An object contains pointers to its sub-objects. Even when the object seems the sole container of the other objects, the other objects are really only referenced. You do not see the actual object. You are only seeing references to it.

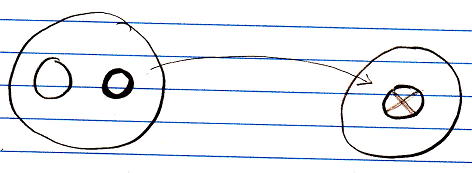
Another symbol can start referring to the same object, making the object all of a sudden not part of a unique container anymore.

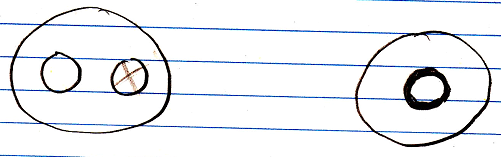


When you annul the object reference in the original parent, the second parent all of a sudden becomes the sole container of the object.



The object has moved from one parent to another.





Objects are always just floating around like that. They do not have to stay in a fixed spot.

In reality the objects do not move at all. They are physically stored in the same spot all the time, no matter where they appear to be. An object can just be freely referenced from anywhere, because objects are always accessed through references.

Even when you *create* an object, you are not directly in touch with the object. The object is immediately assigned to an object reference. Also: when you assign a *value* to an object, you do not assign the value directly to the object, but you assign it through an object reference.

Each object reference gets its own identifier, even when an object reference is Nothing. An object itself, does not have an identifier. An object can be given a Name attribute, though.

### System Command Extension

One thing that makes it important to be aware of system commands at all is *extension* of system commands.

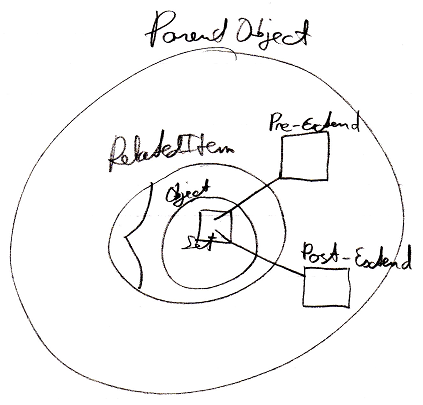
A system command, such as Value Get, can be decorated with a procedure, that determines the value that is returned. It is the new computer language’s equivalent of Getters and Setters or Property procedures. The Object Get command can be extended as well, which makes you able to further control the connection between one object and the other.

Any system command can be decorated with a Pre‑Extension, Post‑Extension or Override.

System command extensions are implemented the same way as ‘normal’ command extensions. Command extension is put under the topic of *Inheritance*, because it is an extension technique.

System command extension will get a notation, that complies to the notation of normal command extension. The code base will implement system command extension like normal command extension and the way extension is implemented inside the code base will influence how the eventual notation will look.

It might look something like this:



But the exact way in which command extension is implemented will be covered by the *Inheritance* articles.

### Parameters For Objects

**[Preliminary documentation]**

#### Concept

This is a preliminary description of the concept. The details are still to be worked out. It could be that in practice, when the new computer language is up and running, the details that have to be solved, will come to light straight away.

In other programming languages there are *getters* and *setters*, which are also called *properties*. Those are replaced by the fact, that any related object has a set of system commands, such as Object Get and Object Set, that can be extended with extra code around the Get and Set actions.

For properties it is sometimes handy to hand a *parameter* to the retrieval of a value, and then a certain value is returned. For instance to return the pressure value of a sound wave at a certain time you could have a Pressure object, that returns a value when you pass the Time to it as an argument.

Pressure ( Time )

Even through the retrieval of pressure could be made a command with a parameter, one might want to see Pressure as an *object*, rather than a *command*.

You can use a related object for that. A related object has a system interface, that allows you to let the eventual object it displays be determined by a procedure. The system interface controls what is returned as the related object. You can extend the Value Get and Value Set system commands. The new computer language must allow you to be able to add extra parameters to system commands, or add extra sub-objects to a system aspect, such as the Value aspect, creating a single parameter for both Value Get and Value Set at the same time, and the new computer language should also allow you to add sub-objects to the whole system interface, to give the retrieval and assignment of any aspect the same parameter. So it is not really the object itself, that gets a parameter, but the related object, that gets a parameter. An object *reference* gets a parameter. That is why the parameter needs to be part of the system interface.

Because you can add a parameter to the whole system interface which extends every system command with a parameter, the new computer language should supply the capability to select which system commands actually get extended with the same parameter, and whether the Time parameter is required or optional. Time is a sub-object of the related object’s system interface, so it is not really a parameter of a command. However, it does extend the system commands with a Time parameter, so a sub-object of a system interface is always called a parameter, but it is called a parameter of a *related* object instead of a parameter of a command.

Adding parameters to the system interface of a related object or extensive extension of system commands is a way to let a retrieval procedure be represented by an object instead of a command.

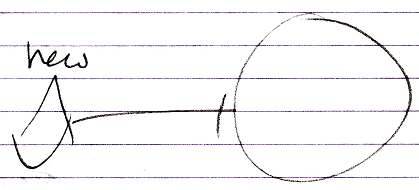
A command in the new computer language can have multiple return values, but when you convert the command into the retrieval procedure of a related object, the command will actually have a single return value. So in this case, you do have the concept of having only one single return value, unlike commands, in which you can have multiple return values.

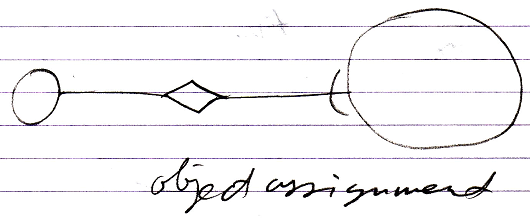
There are no plans yet to make a command, that is a retrieval procedure, and a related object with an extensive retrieval procedure, two completely equally present views on the same thing (like other flat and structured interchange concepts within the language, like exchangeability of class commands and command parameters).

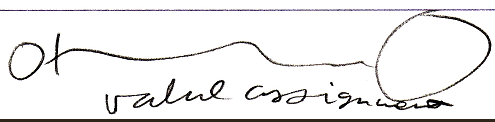
A query is also an example of a related object or related list with an extended system interface, that determines the item, list or result set eventually returned. Dependent on the parameters of the related object, the outcome is calculated.

#### Diagram Notation

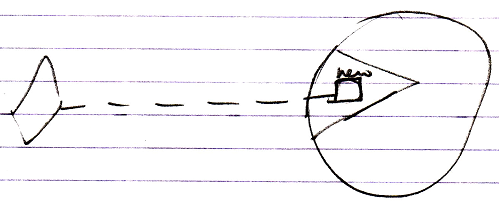
Default system commands can be called with an easy notation, that does not show the system command definition:



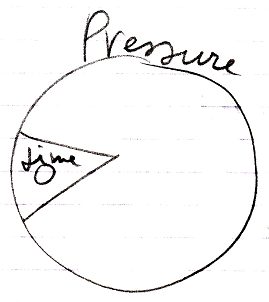




Even though you can also display the system interface and show a call to the command definition:



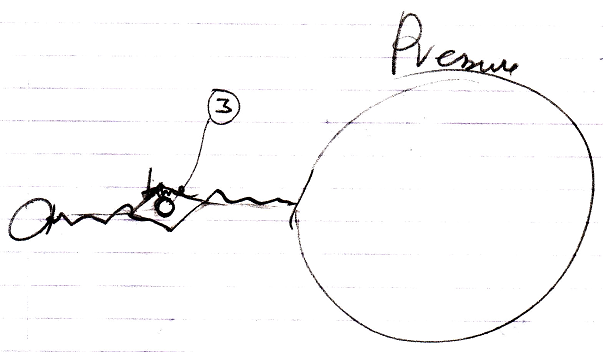
When you very much customize the system interface, you do not always have a standard notation for a consult of the system interface anymore. Giving a related object’s Object Get and Object Set a parameter, you have to display the system interface all the time.



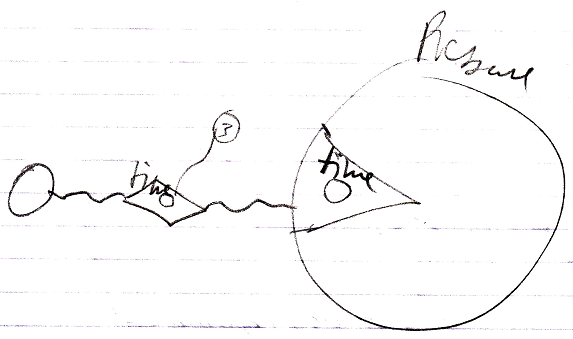
This shows, that the related Pressure object has a Time parameter.

In this case the whole system interface is extended with a parameter, because the Time parameter is not shown in a specific system command or specific system aspect, but shown inside the whole system interface. This means, that with any system command you can supply the Time parameter.

A call to the system command, such as value assignment, will show the Time parameter:



The notation above might not the best one. You may want to show the Time parameter in the related object’s system interface at all times:



This clearly depicts, that the Pressure related object has a Time parameter. You can not go around this parameter.