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

## System Interfaces

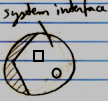
### Main Concept

#### System Interface Notation

Usually you see an object’s members:



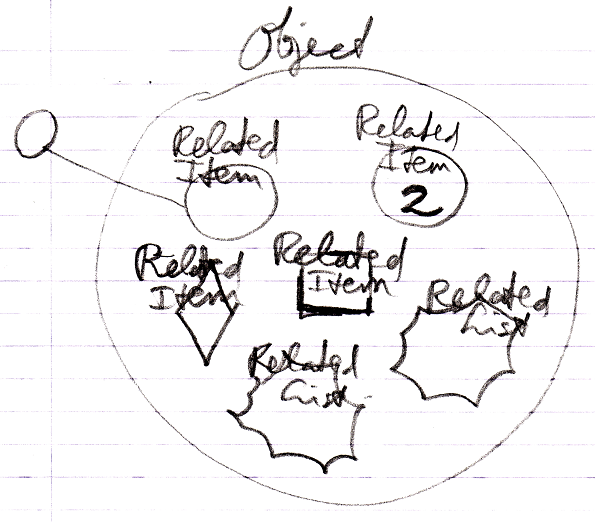
But all objects, references and lists are based on system objects. You can also show the members of the system object instead. You can do that by breaking open the inner workings of the objects and show the *system interface*:



When the system interface is shown, the normal members are not visible anymore.

#### Example: System Interface of an Object

An Object might normally look like this:



But if you open up an Object's system interface, then the related items and related lists are shown as the Related Items & Related Lists collections:



The Related Items and Related Lists are now displayed circles: normal objects, even when they are *commands* and even when they are *lists*. This only represents the exact way an Object actually internally works.

#### Aspects

Next to controlling *sub-objects*, the system interface also controls *aspects*, for instance, the Value aspect:



#### System Interface = Publics of System Objects

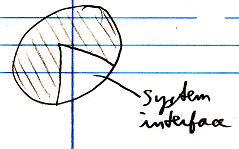
Everything about system interfaces is based on the fact that the system interface is nothing more than the public members of a system object. Even when the explanation about system interfaces is incomplete, what is missing can be derived from the fact, that the system interface will simply show all the public members of a system object. You should also realize that the *private* workings of system objects are not shown in the system interface.

#### Origin of the System Interface Notation

The notation of the *system interface* is derived from the *interface* notation. When an object has several interfaces, each interface is shown as a triangle inside the object:

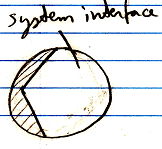


But showing the system interface is like showing the interior ‘of the symbol itself’. Therefore, the triangle of the system interface is now stuck to the border of the symbol:



That way the interface is more part of the object itself, rather than being a sub-object.

To make more room inside the system interface the following, exaggerated notation is used:



It also looks as if you have broken open the shell of the symbol to see its inner workings. It looks like you have opened up the system and can see its internal wiring. The system interface is like seeing the setup of the symbol machine.

### System Interface of an Object

The main thing you see when showing the system interface of an Object is the Related Items & Related Lists collections.



But apart from sub-objects, an object has the following aspects:

- Value

- Class

- Execute

- Clone

- Data

They are controlled through system commands. Those commands will also be visible inside the system interface.

#### The Value Aspect in the System Interface

The Value aspect is controlled through two commands:

Value Get

Value Set

The Value aspect is represented by a triangle, that wraps together the members to control the Value aspect:



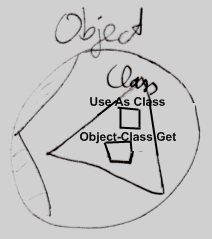
#### The Object-Class Aspect in the System Interface

The Class aspect has *five* system commands, but only *two* of them apply to Objects. The other ones apply to references. The Class aspect of an Object is controlled through the following commands:

Use As Class

Object-Class Get

The commands are placed inside a triangle, that wraps together the members of the Class aspect:

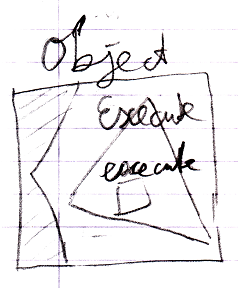


#### The Execute Aspect in the System Interface

The Execute aspect only applies to executable objects, also called commands. The Execute aspect only has one command:

Execute

The command is placed inside a triangle, that wraps together the members of the Execute aspect:



#### The Clone Aspect in the System Interface

The Clone aspect is controlled through two commands:

Clone Get

Clone Set

Both commands have a Depth parameter to indicate the cloning depth.

The commands are placed inside a triangle, that wraps together the members of the Clone aspect:



#### The Data Aspect in the System Interface

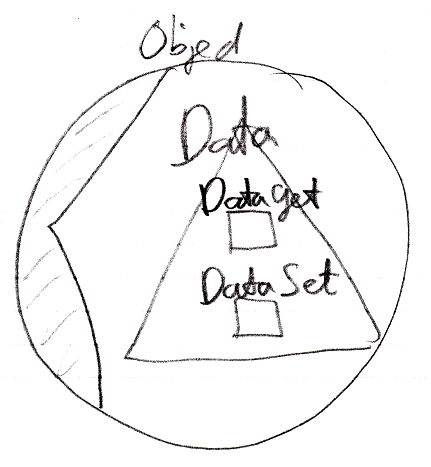
The Data aspect is controlled through two system commands:

Data Get

Data Set

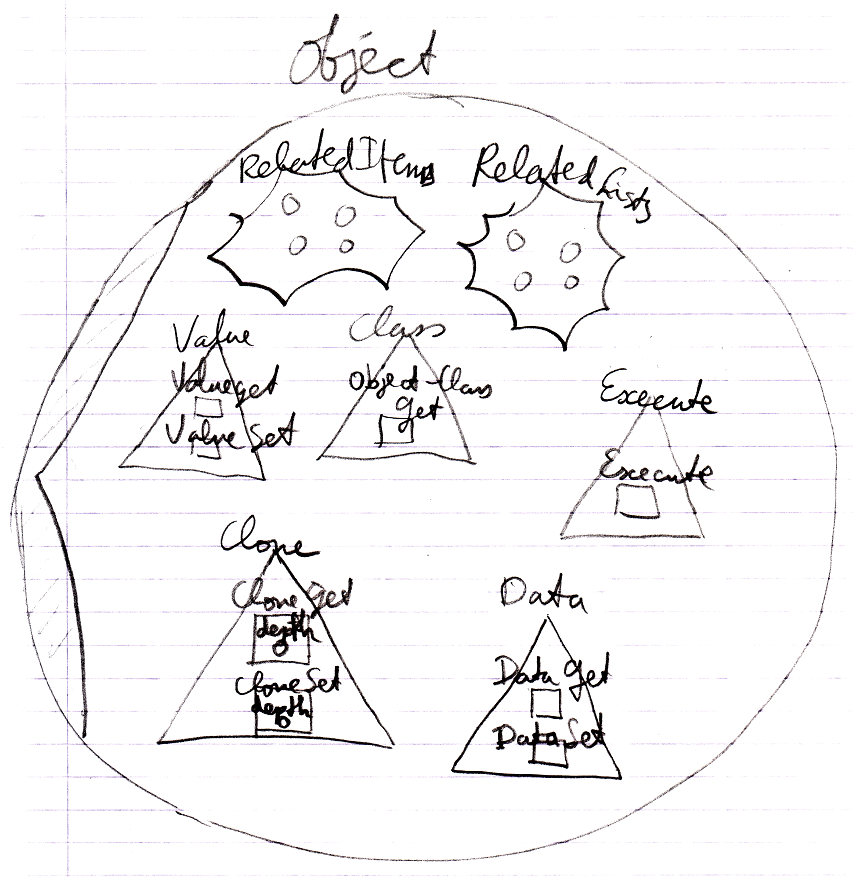
The Data Get and Data Set command can not be called; they can only be access-controlled to control read-write access to the object and all the contents of the object.

The commands are placed inside a triangle, that wraps together the members of the Data aspect:



#### The Full System Interface for Object

The full system interface of an Object looks like this:



### System Interface of a Related Item

When you open up the system interface for a Related Item, you get to see the system commands that apply to Related Items. Aspects, that apply to a Related Item are:

- Reference

- Object

- Existance

- Class

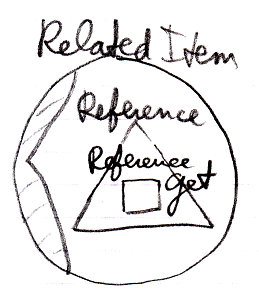
- Name

#### The Reference Aspect in the System Interface

The Reference aspect is controlled through only one command:

Reference Get

The Reference aspect is placed inside a triangle, that wraps together the members to control the Reference aspect:



#### The Object Aspect in the System Interface

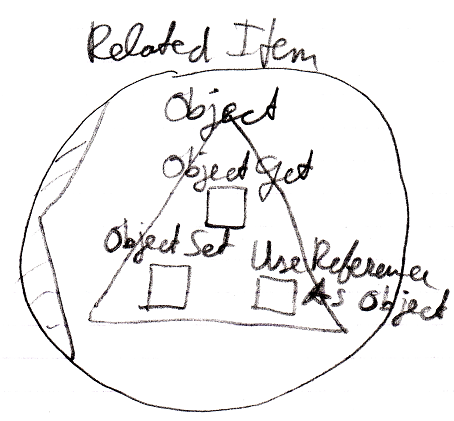
The Object aspect of a Related Item is controlled through several commands:

Object Get

Object Set

Use Reference As Object

The commands are placed inside a triangle, that wraps together the members of the Object aspect:



#### The Existence Aspect in the System Interface

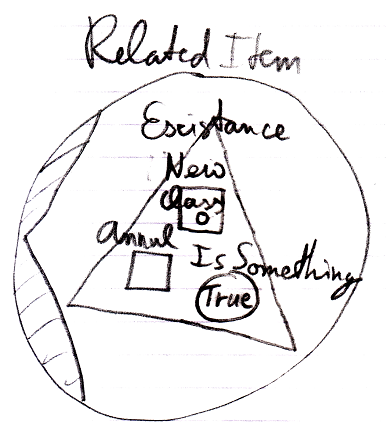
The Existence aspect is controlled through two commands and an attribute:

New

Annul

Is Something

The members are placed inside a triangle, that wraps together the members of the Existence aspect:



#### The Reference-Class Aspect in the System Interface

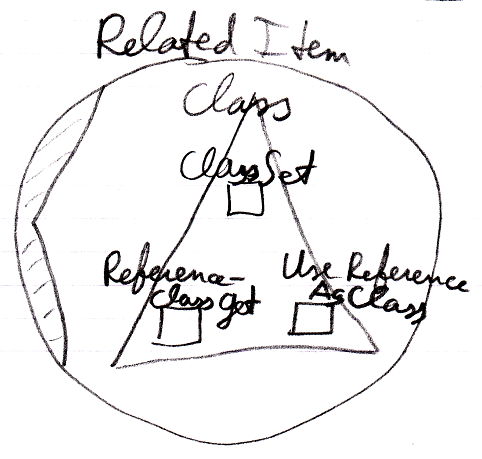
The Class aspect has *five* system commands, but only *three* of them apply to Related Items:

Class Set

Reference-Class Get

Use Reference As Class

The commands are placed inside a triangle, that wraps together the members of the Class aspect:



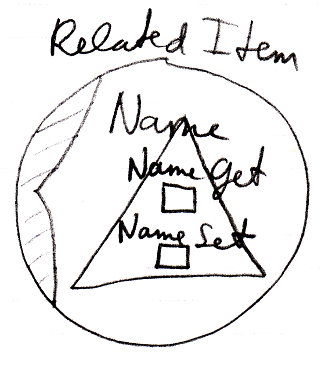
#### The Name Aspect in the System Interface

The Name aspect is controlled through two commands:

Name Get

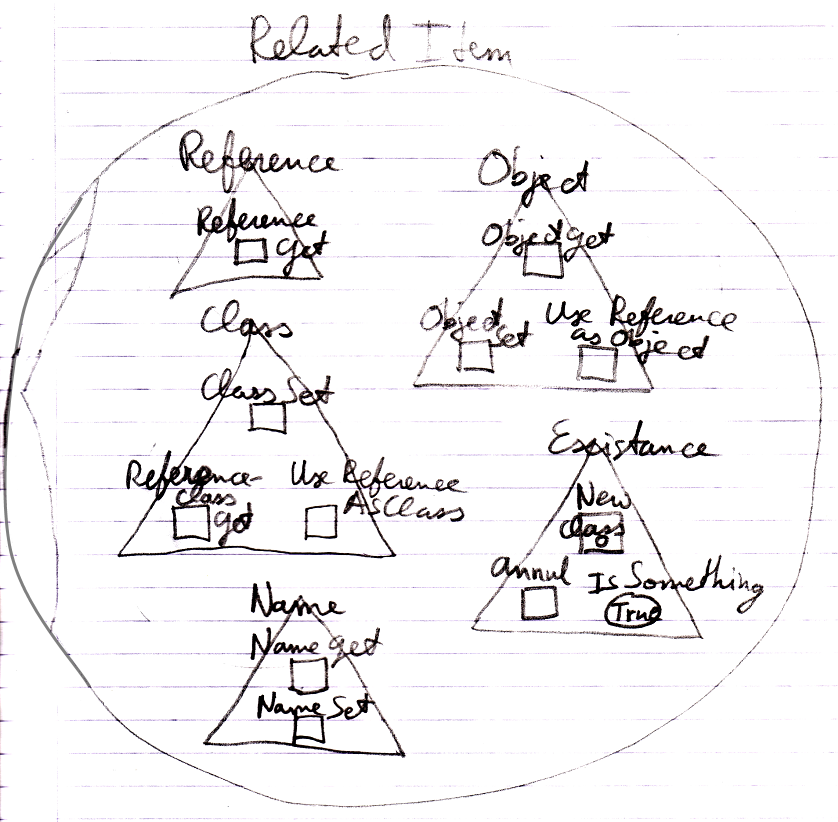
Name Set

The commands are placed inside a triangle, that wraps together the members of the Name aspect:



#### The Full System Interface for Related Item

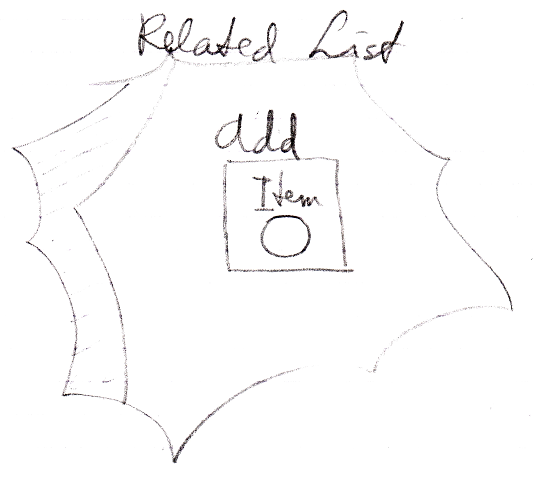
The full system interface of a Related Item looks like this:



### System Interface of a Related List

When you open up the system interface for a Related List, you get to see the system commands that apply to Related Lists. Only the List aspect applies to Related Lists. The system interface of a Related List will show the Add command. The Add command has an optional Item argument, to add an existing item to the list.

So the system interface of a Related List looks like this:



But more members may be introduced later. Specifically you would probably would want to centrally control the Class aspect of all the items in the list.

### System Interface of a Related List Item

When you open up the system interface for a Related List Item, you get to see the system commands that apply to Related List Items. Aspects, that apply to a Related List Item are:

- Reference

- Object

- Existence

- Class

- List

Those are the same as a Related Item, minus the Name aspect, plus the List aspect.

Only the List aspect for the Related List Item is explained here. The other aspects were already worked out in the article *System Interface for Related Item*.

#### Related List Item’s List Aspect in the System Interface

The List aspect of the Related List Item is represented by:

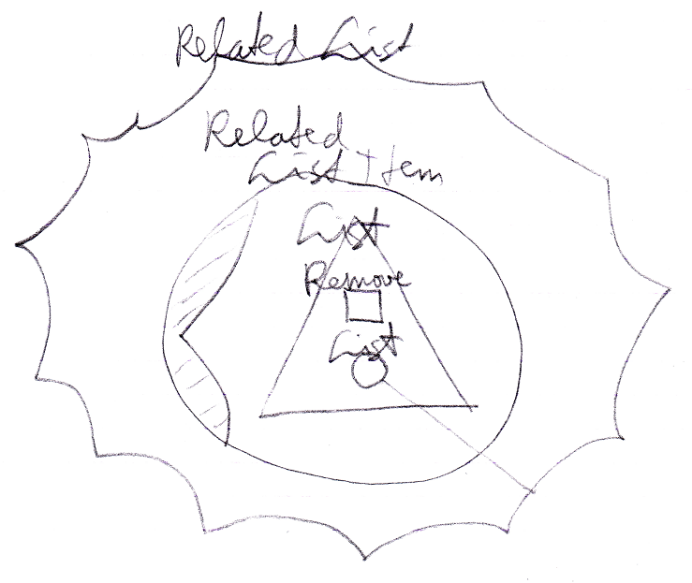
- Remove

The command, which removes the item from the list

- List

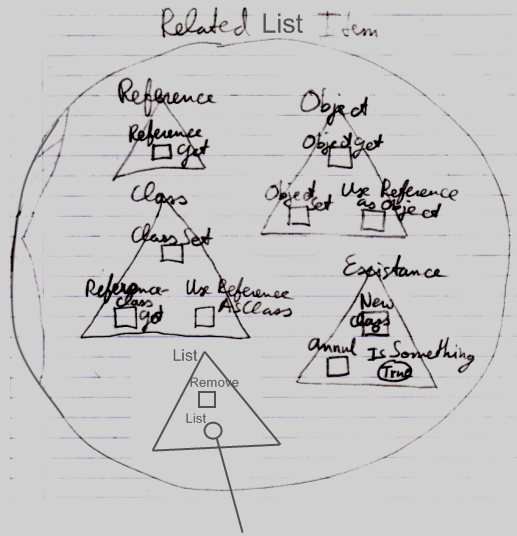
A reference to the List that the Related Item it is part of.

The members are placed inside an interface, that wraps together the members of the List aspect:



#### The Full System Interface for Related Item

A Related List Item is almost the same as a Related Item, so the full system interface of a Related List Item also shows all the other aspects:



### Use-Command Gets Another Aspect

The Use As Class command is part of the Class aspect but Gets the Object aspect.



A Use commands Gets a different aspect than what it applies to. The Use Reference As Object command is part of the Object aspect but Gets the Reference aspect. The Use Reference As Class command is part of the Class aspect, but Gets the Reference aspect.

How a Use command delegates to a Get command of the other aspect is not visible in the system interface, because that is private implementation. You only get to see the public members of the system objects, not their implementation.

### Aspect-In-A-Triangle

An aspect is represented by a triangle containing system commands:



It was a design choice to have system commands placed inside a triangle, that represents the aspect, because it does not always look best to explicitly qualify each Class command with ‘Class . ’ So you can directly call the Use As Class command and not use something like:

Class . Use As Class

Class . Use Object As

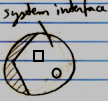
Object . Use As Class

That last one *looks* fine, but places a class-related action inside the Object aspect, which you do not want, because you want to keep all commands that apply to the Class aspect together under the Class aspect.

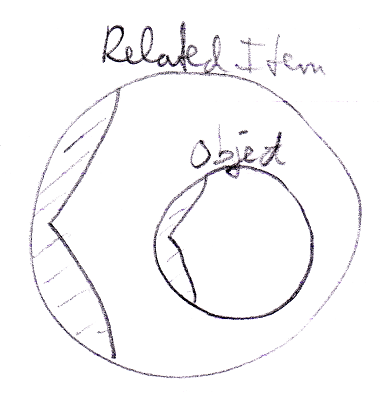
All-in-all: this is a design choice.

### System Interfaces of Objects and References

There are system commands that apply to Objects and system commands that apply to References, but when you show the system interface of a symbol, which do you show? The system interface of the Object or the system interface of the Reference?



The solution for this problem is that you always deal with *references* to objects, never with the object directly, so when you open up the system interface of a symbol, the system interface of the Reference is shown. However, inside of it you will find the referenced Object, showing its system interface.



### Preliminariness of the System Interface Notation

The basics of the system interface notation are not preliminary. But the system interface shows the exact public members of the code base objects. If the code base were to be programmed differently, the system interface members will also look different. Design choices were made in this documentation, which influenced the notation of the system interface members. Perhaps when implementing the code base in the future, different design choices are made and that will change the way the system interfaces look.