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

## System Commands

System *commands* are the commands of system objects, through which aspects of objects are controlled.

Most aspects have a Get and a Set command, but each aspect is controlled in a different way.

### System Commands for the Reference Aspect

The only system command for the Reference aspect is:

Reference Get

A reference can be a Related Item or a Related List Item. This creates two overloads for Reference Get:

Reference Get 🡪 Related Item Get

Reference Get 🡪 Related List Item Get

The reference aspect is used in pointer operations.

### System Commands for the Object Aspect

The main two system commands for the Object aspect are:

Object Get

Object Set

Object Get retrieves the targeted object of a reference.

Object Set changes an object reference target.

Those cover all the standard situations, but pointer-to-pointer situations make things more complex.

#### Pointer-to-Pointer

In a standard situation Object Get and Object Set control references to objects. However, a reference can also point to yet again another reference: to a related item contained by another parent object. This makes the other parent object decide which object is eventually pointed at.

As such, pointer-to-pointer functionality introduces extra commands.

#### Set Object to Reference

To be able to set the object aspect to another related item, Object Set has two overloads:

Object Set 🡪 Set Object to Other Related Item

Object Set 🡪 Set Object to Other Related List Item

If you want a single name to express both situations, you could call it Set Object to Reference.

#### Get Object which is a Reference

Because the object aspect can be another related item, the Object Get command gets two overloads:

Object Get 🡪 Get Object which is Another Related Item

Object Get 🡪 Get Object which is Another Related List Item

If you want a single name to express both situations, you could also call it Get Object which is a Reference. During execution these system commands call Reference Get on the other related item.

#### Use Reference As Object

The reference aspect can be access-controlled for the different ways you can use it. Pointer-to-pointer situations require you to be able to use a reference as an *object*. To be able to access control the different purposes for which you can use a reference, the Reference Get command gets the second implementation:

Use Reference As Object

which delegates directly to the Reference Get command.

#### The Overloads Recapitulated

Do not wreck your brain over all this delegation and overloading. It is just for pointer-to-pointer situations to have the same command names as standard situations, and also to be able to separately access control the specific *uses* of a reference. You will not usually see the pointer-related commands, because they will be implicitly delegated to by the main system commands. This leaves us with the following commands:

Object Get

Object Set

Use Reference As Object

Detail: For that last command you might want to overload Object Get. But that does not work. You can not overload it, because they will both take a pointer to an object as an argument. To disambiguate, they have to have a different name and you have to point to a *specific* command.

### System Commands for the Class Aspect

The main system commands to control the Class aspect are:

Use As Class

Class Set

Reference-Class Get

Object-Class Get

Use As Class is like Object Get, but then for the purpose of using that object as the class of another object. This is common usage of the class aspect. It also makes you able to separately access control whether an object can be used as a class.

The Class Set command is executed on an object reference. The reference can then only point to objects of that class. Class Set applies only to references, and not to objects, because the class of an object is only set upon creation.

The commands Reference-Class Get and Object-Class Get get the class object that is associated with a reference or object. Those commands are actually less commonly used.

#### Class is both Object-Bound and Reference-Bound

The Class aspect applies to both objects and references, but differently. An object has a certain class, which is fixed upon creation of the object and throughout its lifetime. A reference also has a class, defining which class of object you can assign to the reference. The class of an *object* can never change. The class of a reference can be changed, but only while the reference is Nothing. You can not change the class of a reference when an object is assigned to it, except that you can set it to Nothing or to the same class as the object again.

Even though the Object-Class and Reference-Class are the same or one or the other is not filled in, objects and references still each need to separately store which class they have.

#### Pointer-to-Pointer Situations

In a standard situation the Use As Class, Class Set, Reference-Class Get and Object‑Class Get commands are about making an object function as another object’s class. However, you can also make something’s class be yet again another reference. That means that another parent object determines the eventual class.

(However, this might create difficulty for the system to maintain a constant class. You might want another parent to determine the initial class, but the class of an object should not change during its lifetime.)

#### Set Class to Reference

To be able to set the Class aspect to another related item, Class Set has two overloads:

Class Set 🡪 Set Class to Other Related Item

Class Set 🡪 Set Class to Other Related List Item

If you want a single name to express both situations, you could call it Set Class to Reference.

#### Get Class which is a Reference

Because the Class aspect can be set to another related item, the Class Get command gets extra overloads. Next to that, there are different overloads for the two types of Class Get: Reference-Class Get and Object-Class Get. This creates the following overloads:

Reference-Class Get 🡪 Get Reference-Class which is Another Related Item

Reference-Class Get 🡪 Get Reference-Class which is Another Related List Item

Object-Class Get 🡪 Get Object-Class which is Another Related Item

Object-Class Get 🡪 Get Object-Class which is Another Related List Item

You could also call them Get Class which is a Reference.

#### Use Reference As Class

The Reference aspect can be access-controlled for different ways you can use it. Pointer-to-pointer situations require you to be able to use a reference as a *class*. To be able to access control the different purposes for which you can use a reference, the Reference Get command gets the secondary implementation:

Use Reference As Class

which delegates directly to the Reference Get command.

#### The Overloads Recapitulated

Do not wreck your brain over all this delegation and overloading. It is just for pointer-to-pointer situations to have the same command names as standard situations, and also to be able to separately access-control the specific *uses* of references or objects. You will not usually see the pointer-related commands, because they will be implicitly delegated to by the main commands. This leaves us with the following commands:

Use As Class

Class Set

Reference-Class Get

Object-Class Get

Use Reference As Class

Detail: For that last command you might want to overload Object Get. But that does not work. You can not overload it, because they will both take a pointer to an object as an argument. To disambiguate, they have to have a different name and you have to point to a *specific* command.

### The Extra Commands & Overloads

The system commands for the Reference, Object and Class aspects introduce accessory commands and overloads. They seem to be making the explanations more complicated, but they actually make things easier to work with. There are three reasons for the introduction of the extra commands and overloads:

- Access-control usage

- Common commands for related items and related list items

- Common commands for direct pointers and pointers-to-pointers

Here follows an overview of which reason applies to which command or overload:

Access-control usage:

Use Reference As Object

Use Reference As Class

Use As Class (= Use Object As Class)

Common commands for related items and related list items:

Reference Get 🡪 Related Item Ge*t*

Reference Get 🡪 Related List Item Get

Both:

*- Common commands for related items and related list items and*

*- Common commands for direct pointers and pointers-to-pointers*

Object Set 🡪 Set Object to Other Related Item

Object Set 🡪 Set Object to Other Related List Item

Object Get 🡪 Get Object which is Another Related Item

Object Get 🡪 Get Object which is Another Related List Item

Class Set 🡪 Set Class to Other Related Item

Class Set 🡪 Set Class to Other Related List Item

Reference-Class Get 🡪 Get Reference-Class which is Another Related Item

Reference-Class Get 🡪 Get Reference-Class which is Another Related List Item

Object-Class Get 🡪 Get Object-Class which is Another Related Item

Object-Class Get 🡪 Get Object-Class which is Another Related List Item

Again: the reasons for extra commands, overloads and delegation are:

- Access-control usage

- Common commands for related items and related list items

- Common commands for direct pointers and pointers-to-pointers

### System Commands for the Value Aspect

The Value aspect is controlled through two system commands:

Value Get

Value Set

Value Get gets the value of an object.

Value Set sets the value of an object.

### System Commands for the Clone Aspect

The Clone aspect is controlled through two system commands:

Clone Get

Clone Set

The Clone aspect is relate to the Value aspect, but will also copy the sub-objects’ values. Clone has a parameter, that defines the cloning depth: the depth at which to clone sub-objects. When the cloning depth is 2, then this is also expressed as:

Clone (2) Get

Clone (2) Set

Clone Get copies the object and sub-objects. Clone Set assigns the cloned values to another object. This can also be a new object.

### System Commands for the Name Aspect

The Name aspect is controlled through two system commands:

Name Get

Name Set

Name Get gets the name of an object, reference or list.

Name Set sets the name of an object, reference or list.

### System Commands for the Data Aspect

The Data aspect is controlled through two system commands:

Data Get

Data Set

Data Get and Data Set can not be called. They can only be access-controlled. By access‑controlling them you are access controlling being able to read or write to any sub-object or deeper object. The Data Get command abstractly represents any *read*, in other words: any possible Get call to any sub-object. The Data Set command abstractly represents any *write* to any sub-object.

### System Commands for the Execute Aspect

The Execute aspect is controlled through one system command:

Execute

You can access control being able to execute a command. Some commands are never meant to be executed, because they are a definition.

### System Commands for the Existence Aspect

The Existence aspect is controlled through the following system commands:

New

Annul

The New command has an optional Class argument, with which you can indicate the class to create a new Object from.

There is also the system attribute Is Something which returns True or False, also represented by the terms Something or Nothing.

### System Commands for the List Aspect

The List aspect is controlled through the following system commands:

Add

Remove

The Add command is part of a List object. The Add command adds an item to the list. This may be an existing item, passed through the optional argument Item. You can also create a new item for it, possibly of a new class, using the New command, which is part of the Existance aspect.

The Remove command is part of a Related List Item. It removes that item from the list.