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

## Assignment

You will usually not see any direct calls to **Get**, **Set** and **Use** commands. Those system commands are called indirectly by *assignment* commands. An assignment command executes a **Get** on one object and a **Set** on another object, thus yielding over a system aspect from one object to another.

Different aspects have different types of assignment. Below is an overview of the most common types of assignments.

It is also made clear in the overview, which **Get**, **Set** and **Use** commands are called to perform the assignment.

Object-bound aspects and reference-bound aspects are displayed differently. When a reference-bound aspect is **Get** or **Set** then the reference is displayed with a parent around it:



When an object-bound aspect is **Get** or **Set** then the targeted object is displayed without a parent around it:



### Conventional Assignment Types

|  |
| --- |
| **Value Assignment** |
|  |
| Value Get 🡨  Value Set 🡪 |
| *Copies the value of one object*  *to another.* |
|  |
| Object Assignment |
|  |
| Object Get 🡨  Object Set 🡪 |
| *Makes the target point to*  *the same object as the source.*  *So yields over the object aspect.* |
|  |
| Class Assignment |
|  |
| Use As Class 🡨 (~= Object Get)  Class Set 🡪 |
| *Turns the source into*  *the class of the target.* |

In the assignment notation the line type indicates which aspect is yielded over. The access mark indicates the direction of the assignment.

**Value** assignment does not require an assignment call symbol, because a **Value** connection is always an assignment.

### Pointer Assignments

Next to assigning one object reference’s object to another object reference, you could also assign the object reference itself to another object reference. In that case the second object reference will become a *reference to an object reference*, instead of a reference to an *object*. This requires another type of assignment: a pointer assignment.

Pointer assignments establish a pointer-to-pointer. Instead of assigning a target object to the reference, you assign a reference to the reference. This creates a *pointer-to-pointer*, instead of a direct reference to an *object*. This allows another object reference to decide which object is eventually pointed at.

A pointer assignment always has a **Reference** as a source, not its **Object**, not its **Class**, but the **Reference** itself.

A pointer assignment is displayed with an arrow inside the diamond.

|  |
| --- |
| **Object Pointer Assignment:** |
|  |
|  |
| Reference Get 🡨  Object Set 🡪 (~= Set Object to Other Related Item) |
|  |
|  |
| Reference Get 🡨  Object Set 🡪 (~= Set Object to Other Related List Item) |
|  |
|  |
| Reference Get 🡨  Object Set 🡪 (~= Set Object to Other Related List Item) |

Pointer assignment also works for class assignment. You can use a reference as a class, instead using an object itself as the class:

|  |
| --- |
| **Class Pointer Assignment:** |
|  |
|  |
| Use Reference As Class 🡨 (~= Reference Get)  Class Set 🡪 (~= Set Class to Other Related Item) |
|  |
|  |
| Use Reference As Class 🡨 (~= Reference Get)  Class Set 🡪 (~= Set Class to Other Related Item) |
|  |
|  |
| Use Reference As Class 🡪 (~= Reference Get)  Class Set 🡪 (~= Set Class to Other Related List Item) |

### Assignment With Pointer Source

If something is already a pointer-to-pointer and it is the source of a conventional assignment, the target also becomes a pointer-to-pointer. Pointer assignments *establish* pointers to pointers, but in this case a pointer-to-pointer is already there.

So a conventional object assignment can also have the following implementations:

|  |
| --- |
| **Object Assignment:** |
|  |
|  |
| Object Get 🡨 (~= Use Reference As Object ~= Reference Get)  Object Set 🡪 (~= Set Object to Other Related Item) |
|  |
|  |
| Object Get 🡨 (~= Use Reference As Object ~= Reference Get)  Object Set 🡪 (~= Set Object to Other Related List Item) |

Assignment when source is pointer to pointer also works for the **Class** aspect:

|  |
| --- |
| **Class Assignment:** |
|  |
|  |
| Use As Class 🡨 (~= Use Reference As Class)  Class Set 🡪 (~= Set Class to Other Related Item) |
|  |
|  |
| Use As Class 🡨 (~= Use Reference As Class)  Class Set 🡪 (~= Set Class to Other Related List Item) |

### Cross-Aspect Assignments

The standard way to use the **Class** aspect in an assignment is to get the **Object** aspect from one reference and assign it to the **Class** aspect of another reference. But you can also do it the other way around: get the **Class** aspect from one reference and assign it as the **Object** aspect of another reference. You can call it *Class-to-Object* assignment. It can also be called a *Class-Get* *assignment*. Less conventional ways of yielding over aspects like that, is also called a *cross-aspect* assignment.

Also note here, that there are two ways to get the **Class** aspect: **Get** the **Class** aspect of the *object* or **Get** the **Class** aspect of the *reference*.

|  |
| --- |
| **Reference-Class to Object Assignment** |
|  |
| Reference-Class Get 🡨  Object Set 🡪 |
| *Result:* |
| *The object reference on the right now points to the class of the object reference on the left.* |
|  |
| **Object-Class to Object Assignment** |
|  |
| Object-Class Get 🡨  Object Set 🡪 |
| *Result:* |
| *The object reference on the right now points to the class of the object on the left.* |

If the source of the assignment is a pointer-to-pointer, then the target also becomes a pointer-to-pointer. So this also gives **Reference Class to Object** assignment the following implementations:

|  |
| --- |
| **Reference-Class to Object Assignment** |
|  |
| Reference-Class Get 🡨 (~= Other Related Item Class Get)  Object Set 🡪 (~= Other Related Item Set) |
| *Result:* |
|  |
|  |
| **Reference-Class to Object Assignment** |
|  |
| **Reference-Class Get 🡨 (~= Other Related List Item Class Get)**  **Object Set 🡪 (~= Other Related List Item Set)** |
| *Result:* |
|  |

### Cross-Aspect Pointer Assignments

*Pointer* assignments do not have a cross-aspect variation. Pointer assignments use an the reference aspect as the source of an assignment: not a particular aspect of the object reference, but the reference itself. It does not apply to cross-aspect assignments, because on one end of the assignment no aspect at all is involved.