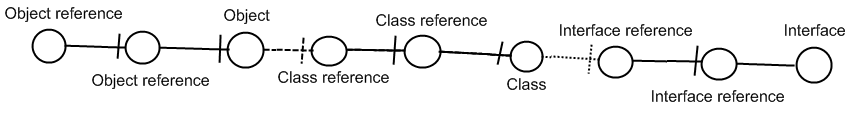
|  |
| --- |
| Circle Language Spec: Interfaces |

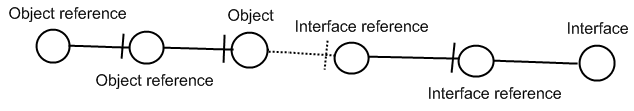
## Target Interface

The *target interface* is found by following a number of redirections to find the actual interface object that the publics of an object are based on.

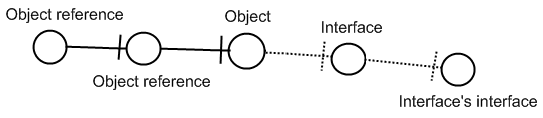
To find the target interface, you first follow *all* the object redirections, then a maximum of *one* class redirection, then *all* the object redirections again, then a maximum of *one* interface redirection and then *all* object redirections again, and there it ends.



Here is an example where there are no class lines involved:

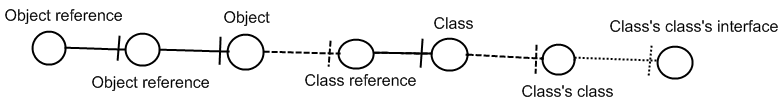


If the interface has an interface as well, this does not redirect the original object’s interface, because the second interface is *another* interface object, that the first interface is just *based* on. An object redirection is just a much tighter bond, than a class or interface redirection.



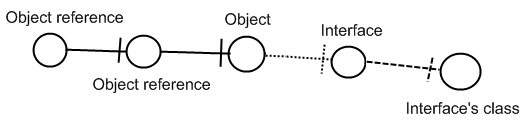
The target interface of the object reference is the symbol Interface, not the symbol Interface’s interface. So you should not follow more than *one* interface redirection to get to the interface object.

You should also not follow more than *one* class redirection to find the target interface. If the class has again a class which has an interface, the first class does not have an interface defined at all:



The last symbol Class’s class’s interface is only the interface of Class’s class, not the interface of Class. So Class is where the interface redirections end.

Also, you should follow one class redirection and one interface redirection *in that order*. Here is a less logical example, in case of which you do not follow the last class redirection:



The last symbol may eventually *define* the interface, but it is still Interface which is the Object reference’s interface object.

## Ideas

*The texts below are loose ideas yet to be turned into good documentation.*

Targets,

2010-05

> Now my mind says: follow all redirections, including multiple interface redirections… but in the Target Class story I stopped doing that. Maybe it is just what you want the term Target Interface to define.

> Perhaps there should be a distinction between interface definition and target interface. I do feel that both the ‘follow only one class or interface step’ version is a concept to be aware of, but the target interface concept would actually be following all redirections to find the object that actually determines the publics.

> Yes. What is now called Target Interface should probably be called the *Interface Object* and the *Target Interface* is the object after following all types of redirections in any order.

JJ

### Out of the original Symbol documentation

#### Tracing Object Aspects

…

The same way, type lines can redirect the interface. Object lines can redirect the type and the type can redirect the interface, so an object line can also direct the interface.

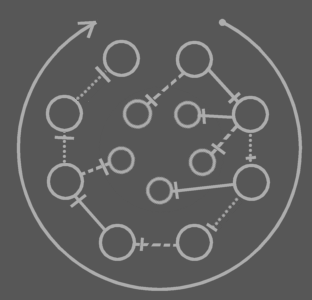
##### Interface Trace

You’ll use all three line types to trace the interface. Follow the interface line if it exists, else follow the object line, else the type line. When you run into a symbol with no lines, then that’s the interface.

A special thing about an interface line is that you can select a single interface of a type and not use the whole type.

When there’s no interface line and the type line is followed, then the type’s *whole* interface is used.

The reason that the object line is preferred over the type line is that if there is no interface line, the *object* can determine *which* interface of the typeis used.



This picture shows any combination of lines an object symbol can have and which of the lines to follow in the interface trace.

In other words: follow interface line, otherwise object line, otherwise type line. The final symbol altogether is the interface.

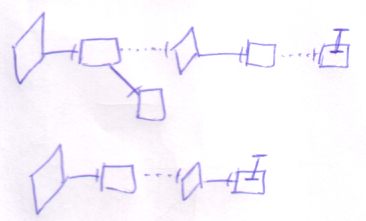
##### Object-Type-Interface Trace

It happens frequently that you need a symbol’s target object, type and interface in one blow. That requires two traces: an object-type trace and an interface trace. The full operation is an *object-type-interface trace*. It deserves a separate name for it’s a very common operation.

#### Tracing Procedure Aspects

##### Procedure Interface Trace

In the *procedure interface trace* you follow both line types, but you prefer following an interface line over a reference line.



The last symbol of the trace is the interface.

Following any of the two line types will eventually lead to the same symbol. In that sense it doesn’t matter what line type you prefer to follow. The interface line is preferred, though, both because it’s more logical to follow, and because following them you’ll probably hit the interface sooner: probably there will be less interface redirections than reference redirections.

##### Execution-Definition-Interface Trace

It happens frequently that you need target execution, definition and interface in one blow. That requires two traces: an execution-definition trace and an interface trace. The full operation is called an e*xecution-definition-interface trace*. It deserves a separate name for it’s a very common operation.

##### Type Line Being called Interface Line

The term *interface target* can mean the target of the *interface line*, but there’s also a second meaning. When the interface line is missing, the interface can also be redirected by an object line or type line. The term *interface target* can also denote the symbol to which the interface is redirected, *be it* with an object line or type line. Sometimes the word *interface line* is even used where I really talk about the line that *redirects* the interface.

The same thing applies to the term *type target*. *Type target* can mean the target of the type line, but it can also mean the symbol to which the type is redirected, be it with an object line, be it with a type line. Sometimes the word *type line* is even used where I really talk about the line that *redirects* the type.

It’s the same thing with the redirection of other system aspects.