|  |
| --- |
| Circle Language Spec: Black Boxes |

## Black Boxes Unfinished

### Misc Issues

In previous projects the main articles about black boxing were finished, but other subjects were left unfinished. This article contains the unfinished material of some of these topics.

This article contains texts and loose ideas about the following topics:

*Deeper Exclusion*

A parent object’s restricting access to specific members of their children.

*Protected*

Accessibility specific to inheritance situations.

*Internal*

Accessibility solely within the module or package itself.

*Objects Take Over Class Access Control*

Objects take over the access control properties of their class.

As said, the texts about these subjects are not finished yet.

#### Deeper Exclusion

Apart from controlling access to the type’s own procedures, a type can also control publicity of procedures *of its chilren*. That way you can publish a child, but make some of its public procedures inaccessible outside the parent. This way you can also exclude members of a base type. A parent can’t make a child’s procedure inaccessible to the child itself; it’s the child’s own procedures.

You can also exclude members of ancestors even further down the hierarchy.

Forms of exclusion other than Private are just called exclusion or privatising.

You can only make a procedure of a child *less* accesible, never *more* accessible. The parent can’t publish a child’s procedure that the child defines as Private. A child can’t put restrictions on how its members are privatised by the parent, so a parent has full control over the exclusion of the procedures its ancestors.

Exclusion access controllers are also called *access* *modifiers*, because they modify formerly set access controllers.

Inheritance, Exclusion,

2008-10-15

This is also a blackboxing thing...

... but I put it in inheritance.

JJ

Inheritance, Exclusion,

2008-10-15

Because it is complexity hiding,

it is blackboxing.

JJ

Exclusion,

20008-10-16

But it is just access control,

so it is blackboxing, not inheritance.

You can apply access control combined with inheritance, to further specialize a class, object or system.

JJ

Exclusion... so a grandparent imposing extra access control onto its grandchildren...

Another advanced topic to cover at the end.

Gee, Public and Private are already a form of exclusion,

because the parent object is imposing restrictions to access

to its sub-objects ( the references it contains. )

When you make it private, it means that it is only accessible to friends...

hmmm...

Deeper exclusion protects members of sub-objects,

but how much can those sub-object members be protected, if they might

also be referenced elsewhere. Somehow they have to be protected from

being referenced elsewhere... fixed logical residence?

What if you reference the object from elsewhere?

Somehow it needs to be controlled, that an object is always referenced

THROUGH the object that protects it...

There is a link with user access control here, but I won't go there yet.

Yeah, there is a discrepancy between controlling access and the fact that

containment is now a volatile thing. Fixed containment needs to be

imposed in order to establish access control...

Multiple objects can impose another access modifier upon the same object,

so where are all these access modifiers stored?

At first it will only be access modification of a direct reference,

which is already under control of the parent object.

But for deeper objects it is different...

so I am not sure yet... are those deeper objects supposed to

be exclusively contained by the grandparent object?

Or can they also be referenced elsewhere,

so that the grandparent has to remember the grandchild access modification

settings and not just delegate that to the child, that holds the grandchild?

I'm not sure.

Perhaps details like that should be worked out when actually turning

this language into an actual programming environment...

and we should keep the focus on the general idea?

Private, Public is exclusion

< Public and Private are an application of exclusion. So then the base of the explanations can be only accessible and inaccessible. >

Exclusion is the primary method for access control. Friends is the method to make exceptions to the privatisation imposed by exclusion. A procedure or type can make any other procedure or type they can reach a Friend, meaning that that procedure or that type has access to all members privately accessible in the befriended.

> And cover deeper access as well.

> Also cover how a fixed logical residence allows parents to impose a permanent, non-volatile access control. Qualified access to an object without it being a fixed logical residence does not guarantee access control imposed by the qualifiers. But if it is a fixed logical residence, the qualifier is mandatory and you can not go around the access control, that it imposes.

#### Protected

There is another basic access controller in addition to Accessible and Inaccessible: *Protected*.



Protected (By the way: it’s not an arrow.)

When a procedure is *Protected*, it means that it’s inaccessible if contained in a circle, but accessible if contained in a triangle.

|  |  |  |
| --- | --- | --- |
|  | ≈ |  |
|  | ≈ |  |

A circular descendant automatically privatises a Protected member:

|  |  |  |
| --- | --- | --- |
|  | ≈ |  |

The member isn’t accessible to any higher symbols than the first circular descendant.

Formally the choice of making an extra Protected access controller seems totally arbitrary. In practice there are situations in which it’s useful to work with members that can be accessible to the interface holder, but not accessible to the outside.

~~Protected is used in exclusion only. It can not be used for ‘outcommenting’ code.~~

A triangular mark:



Is used to indicate that a symbol that owns the line is Protected, meaning, that it is only accessible when the symbol that owns the line is a triangle. A triangular mark is not an arrow.

And then there is Protected. A funny one there. It should be an extra issue, explained after everything else is explained, because it works the same way, except that the befriended object should be referenced with a triangle / should be the friend object’s base object.

Private & Public,

2008-06-10

Protected:

Something needs to be done with protected.

JJ

Being a friend grants full access to the private and protected members of the befriended.

Interfaces,

2009-06-28

do not forget that the protected access modifier can not only

be used for object access, but also for class access and interface access.

(Class Get, Interface Get, etc.)

JJ

#### Internal

< Also called Friend in VB6 >

< better term: internal >

<< I have to note the application of the Visual Basic Friend access modifier. The basic access controller Friend isn’t required (Public Friend and Protected Friend). Friend Basic Access Controller means that you make a procedure’s own globality a Friend globality.

Yes, but for the effect of the Visual Basic Friend access modifier you also need to deprive global members from being accessed outside the globality. And that is done with globality level restriction.

So the effect of the Visual Basic Friend access controller is replaced by • making the globality a friend of the procedure and level restricting global procedures.

NO! It’s just making it Globality Up 1! Nothing Else! Do not use a Friend Globality for specific procedures!

>>

Friend for reference

Means access only anywhere within the globality, but not inside embedded globalities.

< 2008-10-10 That was the VB meaning of Friend. >

A procedure can contain variable objects. The public ones are parameters. The friend ones are parameters within the current globality.

< Old VB meaning of Friend >

< What place the VB6 keyword Friend has in black boxing, I do not know. this is a detail even less prominent, than Protected, which is also a detail later discussed. VB6 keyword friend has to do with access from within globalities. >

#### Objects Take Over Class Access Control

< And references take over object-access control. >

… is objects taking over friend relations from classes as straightforeward as you’d think?

When it is classes, the friendship counts for all the objects derived from that class.

Usually, access to procedures is controlled in a *class* and each of its objects adopts those same access settings.

### Side-Issues

In previous projects the main articles about black boxing were finished, but other subjects were left unfinished. This article contains the unfinished material of some of these topics.

This article contains texts and loose ideas about the following topics:

*Private Names*

You can access control the Name aspect to make the identifier of an object invisible. Assignment commands and execution control commands use this feature to simplify the way they are displayed.

*Inclusion*

This topic may be cut.

*Black Boxing and User Access Control*

Covers how simple black boxing could be made parallel to user access control.

*Programmers and Users*

Adresses mainly the issue why when something is private, programmers get to change everything anyway, no matter how private or inaccessible things are. And why do users not get that privilege and what stops them from getting that privilege.

As said, the texts about these subjects are not finished yet.

#### Private Names

Private Names,

A name being private, even though the reference is not.

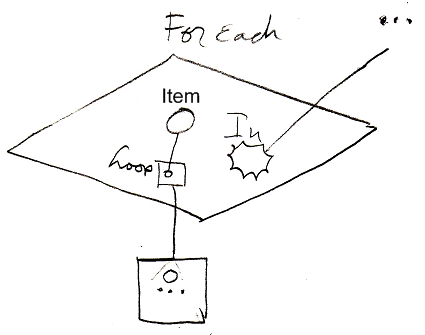
It's a detail, but it is an issue.

It is important to separate the interface from the implementation.

The implementation is only visible in the target definition.

> 2009-06-26: And inside references of friends.

For execution control commands, you have to be able to hide the name of a symbol when you call it. So while the symbol itself can be part of the interface, the name could be part of the implementation, which is not visible unless you’re editing the definition of something.



The cirlce inside the loop parameter is not named. It is an unnamed parameter, that might be referenced through the name of the variable it is pointing to. But I’m not sure about that.

JJ

#### Inclusion

Apart from a *decisive* access modifier, you can also give a procedure a *suggested* access modifier. This means that when you put an object in a type, a procedure of the object will get its *suggested* access modifier, for instance Private, but if its *decisive* access controller is Public, then you can still upgrade it to Public. This is called *inclusion*. The suggested access modifier is always set to less accessible than the decisive one. A type can also give suggested access modifiers to procedures of deeper objects.

> 2009-07-06: Inclusion might only be useful for programmers. But programmers get automatic friend access, so real, exclusive access control is already the suggestion and everything can still be included, so it might be an unuseful concept.

#### Black Boxing and User Access Control

Access control will be fully managed

by the access control module, even for

access controlling an object's members,

making the members only accessible

to the direct container of the member in

order to hide complexity from the outside,

or to to protect the object's data.

But that form of access control is so

important in programming software,

that it is introduced earlier

as the concept of Public & Private,

before access control is worked out

as a complete solution to security.

Maybe access controllers are a step into the direction of security. Maybe security should be a kind of advanced access control.

Interesting idea for user access control:

a constant in a program is a variable that can

be changed by a programmer, but not by a user

and also not by the program itself.

Public & Private,

± 2008-09

This project should be the working out

of the concept of Public & Private, even

before it is integrated into the complete

Access Control concept.

Public & Private need a preliminary notation,

and preliminary explanation, so that the concept

can be used in explanations of

coding concepts whereever required.

JJ

Private & Public,

2008-06-10

Access Controlling System Aspects:

Save for user access control, because this is about different roles:

You can have friends for reading, writing

or changing (add and delete symbols.)

JJ

#### Programmers and Users

… = user access control

privates visible in friends, and in definitions when you have user access to it, so you are a programmer, not a user.

Something like that…

Private & Public,

2008-06-10

Editing definition:

> 2009-06-30 Some of the ideas here are wrong, but the general idea should be thought of about editing the definition.

Friend access is different from access to private contents

only when editing the definition.

> 2009-06-26: No. this is user access control. Some users are authors of a definition. Some are not. Do not confuse it with private/public concepts.

It's the difference between friend for reference,

and friend for change.

> 2009-06-26: Bull, see last comments.

If you are a friend for change (not for reference) for private contents,

you will only see the private contents if you are at logical target the definition.

JJ

Set and Copy are used to set up lines. Lines are set up at design time, but can also be changed at run time. Sometimes you want to set a line at design time without being able to change it at run time. Therefore, Set and Copy must be able to have different access controllers at design time and run time. Just like the Get purposes can have different access controllers, it’s possible to have different access controllers for Design Time Set and Run Time Set and for Design Time Copy and Run Time Copy. Something is a constant if it can be Design Time Set, but not Run Time Set.

Object symbol system procedures and purposes:

Symbol

Get : Copy (Run Time or Design Time)

Object

Get : Copy (Run Time or Design Time) or Redirection

Set : Run Time or Design Time

Type

Get : Copy (Run Time or Design Time) or Redirection

Set : Run Time or Design Time

Interface

Get : Copy (Run Time or Design Time) or Redirection

Set : Run Time or Design Time

State

Get : Copy (Run Time or Design Time)

Set : Run Time or Design Time

Target Object

Get : Copy (Run Time or Design Time) or Access

Target Type

Get : Copy (Run Time or Design Time) or Access

Target Interface

Get : Copy (Run Time or Design Time)

Procedure symbol system procedures and purposes:

Symbol

Get : Copy (Run Time or Design Time)

Call

Get : Copy (Run Time or Design Time)

Set : Run Time or Design Time

Reference

Get : Copy (Run Time or Design Time) or Redirect

Set : Run Time or Design Time

Target Procedure:

Get : Copy (Run Time or Design Time) or Access

### May Cover Last

In previous projects the main articles about black boxing were finished, but other subjects were left unfinished. This article contains some unfinished material. The topics are considered of lesser importance, compared to even other topics, that were not finished.

Some of the topics might be deleted and others might not.

#### Access Controlling Globals

> This issue is more important than level limitation and globality level limitation, but those two concepts are used in the implementation as laid out below (the implementation will probably change in the future.

Access modifiers in Globalities:

* Global Inaccessible
* Global Private
* Global Public

To understand what happens when procedures inside globalities are given access controllers you need to view the globality as an object, that is referenced from any of its ancestor objects:

|  |  |  |
| --- | --- | --- |
|  | ≈ |  |

If a global procedure is Inaccessible, it’s just not accessible at all (*‘outcommented’*), so not really global either. If a global procedure is Private, it’s not really global either, only accessible by the globality object itself, just like something private of a circle object. If a global procedure is Public, then it *is* global. It is then accessible from any ancestor within the globality.

When a global procedure is Public it is however also accessible *outside* the globality, as a public procedure of the globality object. For a procedure to be global inside the globality, but not accessible outside the globality you use globality level limitation: **Public Globality Up 1**.

> 2009-06-27: And here I am attempting to differentiate between VB6 Friend on a global level and VB6 Public on a global level. I am trying too hard to make a match between access modifiers I know from VB6 and C++ and match them with a new language. I was too afraid to reinvent each atypical concept, and insisted to put those concepts inside a single systematic. That is why I went to such an extent with this level limitation stuff. Ok, I can respect it now. It is an attempt to put those access modifier situations into context, but I think this should only be an intermediate implementation of it and in the near future an easier approach needs to be come up with. This stuff here should be seen as just a brainstorm.

It is not a permitted to make a global procedure Protected. A pentagon is not exchangable with a triangle, so Global Protected wouldn’t have much meaning.

> 2009-06-29 That, by the way, is not true anymore. You can reference a globality triangularly if you want.

Access Modifiers in Globalities needs to be further thought through. It gets tricky as you put it in diagram code.

#### Simplified Access Connectors

Could also be called: Access Control Presets.

< Wat wel zo is is dat de standaard access control van system procedures er voor zorgt dat je niet *zo maar* rare dingen kan gaan doen. Erg belangrijk als je effe snel iets programmeerd, dat je de echt rare effecten uitsluit zonder extra moeite. >

< 2009-06-29 Bedoelde ik hier oorspronkelijk mee de macro-keywords voor access control? Zoals bijvoorbeeld Public Read-Only Value ofzo? >

<

I don’t have a special notation for publicity of Set procedures.

Set public or not is kind of like read only or not. Maybe I need a read only notation.

Write only is also common, though. For instance of procedure reference input parameters.

>

<Also cover constants>

When you give a system procedure an access controller, access to the procedure or object represented may get a special name and diagram notation.

An object symbol with an accessible State *Read* procedure, but no State *Write*, is called a Read Only Object.

An object symbol with only a Type Read, can be considered just a Type. Not an object or interface.

An object symbol with only an Interface Read, can be considered just an Interface. Not an object or type.

…

Etcetera, work it all out

Standaard access controllers

* Standaard zijn Add en Remove geblokkeerd. Zo zijn er nog meer dingen standaard geblokeerd

De klasse kan zelf initial access controllers aannemen die je ook weer kunt veranderen.

Something is constant if it can be set by programmers , but not by users.

< By default everything is public. When studying the access configuration of a symbol you might only want to see what differs from the default >

<The standard access modifiers for system procedures of procedure symbols directly inside a procedure symbol are different from procedures directly inside object symbols. For instance, Symbol Get for Reference of a clause is Private by default. Symbol Get for Reference for an object member is Public by default>

A call is by default private.

The access controllers of the procedure’s sub objects determine which purpose they serve. Maybe I should write a section about that in Access Control.

Required and Optional parameters as well as In, Out and Thru need to be rediscussed in Access Control, because it may be so that it is required for you to Object Set a parameter, while the procedure changes the state of the target

object. This is like thru too, but a totally different fashion and there are more ways of throughput.

If a configuration of any sort is a common one, it might be nice to make something like a typedef so you give the configuration a name of its own, Like if you commonly use ‘Public Up 1 Globality Up 2’, you might want to give it a different name.

Consider combining the concepts Presets, Macro-Keywords and just a brainstorm about nice-to-have keywords for common access control situations.

Best example: Public Read-Only Value. Read-Only Value.

You do not even have to introduce completely sound systematics, you can also just shed light on the general concept.

Those 'macro' access modifier keywords are not very important.

There may be an analogy between macro keywords and a C++ typedef.

(perhaps only for names)

( I think this issue was also temporarily put in the Parameters chapter originally.)

But what about ‘making a command Private’? What do you mean by that? You can access control the Execute system aspect of commands, but that does not make other aspects of a command object private, such as the Object Set aspect (turning the command symbol into a command reference).

Should there not be something implied when a connector is not there: in a friend relation this will create a lot of connectors… so no connectors should mean everything is accessible in a friend or one connector should be the placeholder for any other connector… or some common connectors and others (such as Class Get implied )

In definitions, publics could also be differentiated from privates by privates not getting access connectors? Well… friends can access them, so they need access connectors…

When you have to display a lot of connectors, you might summarize them, and when you float over them, they expand into options.

Maybe first you have an object connection line without a connector. You float over it, it splits up into an class, interface, object and value line, you float over one of those, it splits up into for instance Object Get and Object Set. I know that does not cover all the options, but it might be an idea to display the connector summary and fan it out when you float over it.

Explicit display that something is private

is optional.

Explicit display, that something is public

could be made mandatory, because

is it not overhead in notation: it clearly

indicates a connection point.

If you do not see any access connectors, do you assume, that everything is accessible,

or that nothing is accessible?

That is a good question for the display of access connectors.

But not essential to cover.

Questions:

- Can you make it an option to only show befriended commands inside an object, instead of friend and non-friend commands.

I am not sure yet. It is not clear.

Another fact:

One of the most important things to access control is an object’s being accessible at all or having read-only access to it. Another important access controlling capability is to only be able to retrieve (and set) the Value of an object. Perhaps, if you state that Set is public it automatically means, that Get is also public, and if you state that Get is public, it automatically imples that Set is private.

> Perhaps it is interesting when one user writes a value, but other users can not read it out, but that is sort of outside the scope: that is user access control. System access control is covered here.

Perhaps that way the number of possible access controllers becomes more limited, so you do not have to display each possible access connector: Object Get implies any kind of connector.

Access controlling for just allowing a Class Get is stupid as well: who wants that? Who wants to only be able to access the reflective data of an object, but not the object itself?

The notation is simply drawing the symbol a dotted line or a dashed line,

which makes it only usable as an interface or class.

How you should express something only being usable

as a class but not as an interface is not clear to me.

I do not think I really need that and I do not think I

need to clear that up.

But whatever, this is for later.

Also useless: allowing Set but not allowing Get… if you can already write it, why not allow reading it? Is there any point to that? The important thing is write-protecting, or not access at all… but read-protecting, while you could write to it? It seems unlogical.

That seems a whole lot of Get and Set purposes. In practice, the set up of a symbol’s system procedures is not that complicated. There’s a standard setup: the most usual system procedure setup. Only deviations from the standard are additionally denoted.

### Might Not Cover

These topics are considered of such less importance, that they eventually might be deleted. This is because they might introduce discussion where no discussion is required. And that introduces confusion.