|  |
| --- |
| 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:

*Protected*

Accessibility specific to inheritance situations.

*Internal*

Accessibility solely within the module or package itself.

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

#### 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 module a Friend module.

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

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

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

>

-----

Friend for reference

Means access only anywhere within the module, but not inside embedded modules.

< 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 module.

-----

< 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 modules. >

### 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.

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.

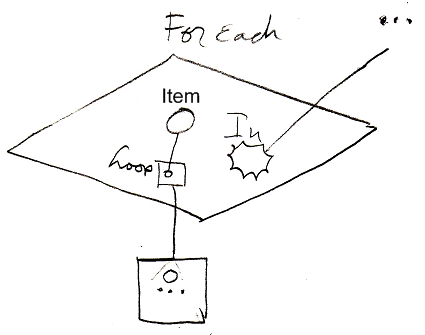
It's a detail, but it is an issue.

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 circle 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

### 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 module 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 Modules:

* Global Inaccessible
* Global Private
* Global Public

To understand what happens when procedures inside modules are given access controllers you need to view the module 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 module 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 module.

When a global procedure is Public it is however also accessible *outside* the module, as a public procedure of the module object. For a procedure to be global inside the module, but not accessible outside the module you use module level limitation: **Public Module 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 module triangularly if you want.

-----

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