# Circle Language Spec Plan Planning + Black Box Spec Part A 2009-06 Cycle 2: Black Box Part A

*Author: JJ van Zon*

*Location: Oosterhout, The Netherlands*

*Date: June 26, 2009 – September 10, 2009*

## Purpose of This Document

This document is a checklist for software development-related work, for a good approach and to not forget anything. It is not to be read over litterly. Later it serves as a reference for looking up how exactly something was done.

## Rough Plan

- Reach goal **(3)**

- Organize ideas (29 pages)

- Work out the implementation in mind now

## Approach

- Existing material:

- The bulk of ideas inside the existing article

- The description out of the Coding Concepts redirection page

- Main issues:

- Do not focus on the already finished texts, trying to turn it into something usable

- Get the idea straight

- Make an adequate description of it

Look over the existing material there is now.

Work out the idea as you go along.

Make concise documentation.

Consider the ‘writing efficiently’ rules in the project documentation of *New Computer Language Functional Design*.

Writing-efficiently issues you have to consider when making these articles:

- Creative vs. productive: forget about the time planning while you are working out this idea

- Blunt, do not compare: in the eventual text stop comparing it to existing languages and stop defending it. Just lay it out as it is.

- Process ideas: there are a lot of ideas to process... and not sure what to do with it.

- I am also tempted to look at all the implications.  
By rule you should not do that.

Perhaps I do need to look at the implications, to see if things make sense,

but you should not keep those contemplations in the documentation.

- I think some ideas just need to be deleted, because they introduce discussion where no discussion is required. It introduces confusion.

## Steps

- Organize ideas

- Go through idea list (29 pages)

- Organize idea list

Categorization:

You have to categorize the ideas by relevance.

And then the relevant topics need exact definitions in a list of topics to cover.

Then you have a structure within which to work out the topic completely.

Look if the ideas introduce anything new and change the idea.

Possible influences of ideas:

- Matters conceptually

- Confirms the main idea

- New conceptual ideas

- Matters for notation

- Details, cover last

- Details, may not cover

- Out of scope

(not relevant in this context)

- Obsolete

- Do not use

(rules for writing efficiently)

- Approach from now on:

Funny how the ideas of the original Symbol documentation are still so relevant.

Forget about the planning, goal description and approach,

because now you need to just dig into it and focus on the creative process...

the form in which to work comes later.

I am almost sure, that during this project I will hit a wall a couple of times and have to do it completely differently.

- Get concept clear in your head

> By drawing pictures

- Make concept work

(figure out loose ends)

> You should not even read over the ideas anymore.

> Use the ideas as a cross out list.

> Do use the *categorization* of ideas, though.

> Unimportant issues are skipped. > Will be figured out later.

- Figure out loose ends:

- Comparison to traditional:

- Friend commands in object

- Containment, not mutual references

- Possible ambiguities:

- Possible ambiguities in other system aspects

- Possible ambiguities in assignment (in case of all system aspects)

- Possible ambiguities in mutual friendship  
(mutual friend declaration and private member usage)

- Write Black Box articles

- Make article subdivision

> Idea subdivision will *be* the article subdivision

- Finish up those texts

> Work on them and work on them until it gets clearer and clearer till it is done.

- Scan pictures and put them inside the documentation

(so I can see what I have already and the way it looks)

- Work out the other pictures

- Read over and reformulate:

- Read over the existing texts

- Then work out the texts to go along with the rest of the pictures.

- And the explanation order should change, to make things clearer.

~X The main issues, that remain are making 'Accessing Other Aspects'  
the same kind of overview as 'Access Controlling System Aspects in a Diagram'

- Also the issue of ‘inward active Get’, ‘outward a past Set’ was encountered

- Plan now (2009-07-20)

- Figure out Connections for New, Annul and Execute

- Finish Assignment of System Aspects

- Read over finished work

- Cross out details

- Write 'Object-Bound & Reference-Bound Aspects'

(± 6 hours, ending at 23-07-2009)

Adapt the other documentation to the distinction between object-bound & reference-bound aspects:

- Before introducing access control onto other aspects, you have to explain the difference between object-bound and reference-bound aspects.

- You also *have to* work out exactly how object-related aspects are access controlled and how friend declarations work in that case.

- The rule: ‘always imposed by parent’ is embedded into the stories, before Accessing Other System Aspects. That rule does not always apply, so you should not introduce it there either. You can introduce that in the object-bound and reference-bound story.

> It is ok this way.

- Then the access control connector diagrams have to change. The object-related aspects do not require a parent.

- I misinterpreted the Reference Class and Object Class terms when updating the connectors and connections.  
It does not have anything to do with pointers to pointers.  
You have Get Reference Class and Get Object Class,  
but not Object Set Class or Object Use As Class

- Move remarks in overviews to below the overviews.

- Wait with Misc Issues and Side-Issues

> Those are completely left out of the plan here.

- Left-over ideas will be processed later like a cross-out list later.