|  |
| --- |
| Circle Language Spec: Commands |

## Creation Behavior Of Calls

A call can be present inside an object or inside another command. When a call is created, it is not immediately run, so that you get a chance to set its parameters. Before a command call is run, the command call’s public contents are there: its parameters. Those parameters are copied out of the call’s definition. The private contents are not there yet. A command’s private contents include private objects, clauses and command calls. The private contents will be copied out of the command defininition, only just before the command call runs. The public contents of a command call are added when the command object is created. The private contents of a command call are only added when its about to run. So the general rule is: creation of private contents of a command call is delayed until just before a command call is run.

The reasons for the delay of creation of private contents are explained later. First, the steps of a command call’s creation are laid out one by one.

Before a command is running, it is in the current state:

- Command is created

- Parameters / public contents are present

- Private contents are *not* created  
(private objects, clauses and command calls)

- Reference and Object Out parameters are already assigned

- Value parameter assignments refer to their sources and targets

Right before a command executes:

- Value In parameter assignments are executed

- Private contents are created  
(private objects, clauses and command calls)  
(copied out of the command’s definition)

- *Sub*-commands’ parameters / public contents are created  
(copied out of the sub-command’s definition)

- Mind, that the sub-command’s private contents are not created, which disincludes its private objects, command calls and clauses.

- *Sub*-commands’ Reference and Object Out parameters are assigned  
(targets copied from the parent command’s definition)

- Sub-commands’ Value parameter assignments refer to their sources and targets  
(copied out of the parent command’s definition)

Then the command runs, which means it runs all its sub-commands one by one, following the same procedure of call creation.

After a command call has finished:

- Value Out parameter assignments are executed

- Private contents are released  
(private objects, clauses and command calls)

### Calls In A Parent Command

A parent command, that executes, automatically executes its sub-command-calls one by one. This means, that when a parent command is about to run, the sub-command-calls are created with *public* contents only. A sub-command-call’s *private* contents are created just before a sub-command-call is run. After a sub-command-call has run, the private contents are released, but its public contents remain. Sub-command-calls are the parent command’s private contents, so after the parent command is done, the sub-command-calls with their public contents are released. Because the parent command has a sole reference to its sub-command-call, releasing the sub-command-call means, that the sub-command-call is destroyed. But the parameter objects of the sub-command could still keep existing if they are still referenced in other places.

### Original Problem & Solution

This was the original problem:

When a sub-command would only be created just before it is run, there is a problem: if a sub-command is created only just before it is run, when can a parent command fill in the parameters of the *not yet created* sub-command? Parameter passings could not reference the sub-command’s parameters. That’s why before running the parent command, the sub-commands need to be created.

But when creating the sub-command even before it is run, there is another problem: if sub-commands need to be created when their parent is created, the sub-commands of sub-commands would also need to be created, and their sub-commands and so on. So the whole call structure would have to be created before the parent command could even run. This could even have resulted in circularities, that would make the system hang.

The solution was, to create a call before it is run, but only the public contents, so that a parent can set the parameters. But the private contents are only created just before the command is run, which creates its sub-commands, but again only its parameters. This prevents recursive creation of the call structure, and gives command creation a neat and steady pulse. So this all makes it doable. Otherwise there could have been an endless recursive creation procedure, before any command could ever run.

### Problems Solved By Delayed Creation

The creation behavior of command calls also solves *when* and *when not* to display a command’s private contents.

The problems solved by delayed creation of a command’s private contents are covered in separate articles:

- *No Overhead of Command Creation*

- *No Circular Command Creation*

- *No Private Contents in Calls in Definitions*

- *Calls in Calls Show Privates When Running*

### Delayed Creation Of Private Contents Only Counts For Command Calls

Delayed creation of private contents only counts for command calls. Command calls redirect their definition.

But if a command object does not have a definition, then it defines its own definition. For command objects that define their own definition, private contents have to be created all the time, because nothing else defines its private contents but the object itself. This counts for command definitions. This also counts for *active* command definitions, which are executable object that define their own definition. But it also counts for clauses. See the article *Creation Behavior of Clauses*.

### Compared To CPU-Like Call

There was a lot of brainstorming about how this relates to the way it goes in other programming languages, which have a command call creation scheme totally compliant to the CPU’s way of handling command calls, which is not in the spirit of *commands as an object*. But that brainstorming was postponed, because it did not have anything to do with a goal, but more with optimization of the new computer language. The contemplations are still there in the article *Comparison to CPU-Like Call Instantiation*, as an unfinished brainstorm, that may be later worked out, to better take advantage of CPU power.