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

## Commands Misc Issues

### Parent Controls Its Sub-Executions

A parent command needs full control over the execution of its sub-commands. The parent command makes the decision about the exact moment, that the sub-command runs.

To enforce this general rule, three rules are imposed to make sure a parent command never looses control over the execution of one of its sub-commands:

- Sub-commands are never referenced

See the article *Sub-Commands Are Never Referenced*

- Beware of active command references in commands

See the article *Beware of Active Command References in Commands*

- Sub-commands are never manually started

See the article *Sub-Commands Not Manually Started*

With these rules, there’s no way to point to sub-commands, and the point is made, that you have to beware when you are using active command references inside commands. And there is no way for a user to start the sub-command at any arbitrary moment. There’s no way in, and you have to be aware of the way out, so the sub-command is always an isolated command object, the execution of which is initiated by the parent command, or if it is an active command reference, you have to be aware of giving up control. An external force can not execute the sub-command it before the parent command chooses to execute it, unless you use an active command reference.

Any command object, potentially executed within another command definition, needs to be protected against this risk. The rules to protect the system against this hazard could be changed in the future. But the *reason* for this protection does not change.

For instance: the rule ‘sub-commands are never referenced’, may be changed to being able to reference sub-commands after all, but never to be able to *execute* a sub-command through a reference, even if the reference is active. But this change of rules is just an example. It might prove not to be practicle after all.

#### Sub-Commands Are Never Referenced

This is a rule for enforcement of control of a parent command over the execution of its sub-commands. Active clauses and command calls inside another command and are never referenced, because a command has to have full control over the execution of its sub-commands. If you could reference an active command inside a command, then the sub-command could be prematurely executed through that reference. Therefore sub-commands are never referenced.

It is *not* the rule, that command calls can never be referenced. It’s just that command calls *inside another command* can not be referenced. When a command call resides in an object, the command call *can* be referenced, to for instance allow a user to carry around a reference to an active command, executing on a site somewhere. So a command call inside an object can be referenced, but a command call inside another command can not be referenced.

Not being able to reference sub-commands does not mean, that you *can* use it as a class, because a class reference is also a reference. This would also put the sub-command in danger of being prematurely executed, because you could establish an active reference to the class of another command object and execute it.

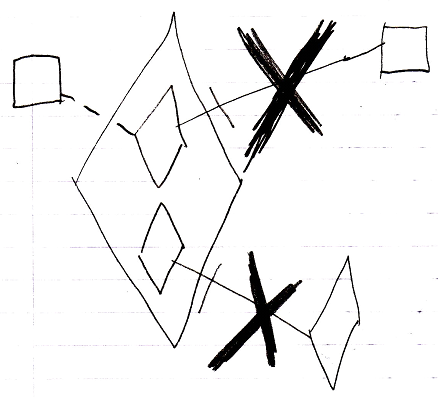
To not cause any confusion, sub-commands are made Private.

Do not change the rule to *sub-commands are always private*, because this would not fully solve the parent command’s control over its sub-commands’ execution. By just making them private, the parent command could still pass a reference to a *sub-*command, so that the parent *gives up* control over the execution of a *sub-*command. This is something, that will not be allowed. Now that you can *never reference a sub-command*, this control is restored.

##### Diagram Notation

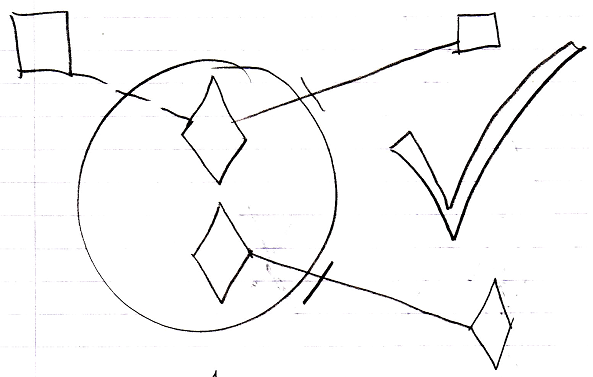
The current paragraph repeats the story, but now demonstrates the concept using diagrams.

Active clauses and command calls inside another command and are never referenced, because a command has to have full control over the execution of its sub-commands.



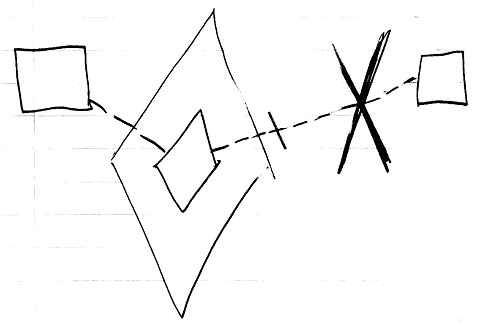
If you could reference an active command inside a command, then the sub-command could be prematurely executed through that reference. Therefore sub-commands are never referenced.

It is *not* the rule, that command calls can *never* be referenced. It’s just that command calls *inside another command* can not be referenced. When a command call resides in an object, the command call *can* be referenced, to for instance allow a user to carry around a reference to an active command, executing on a site somewhere.



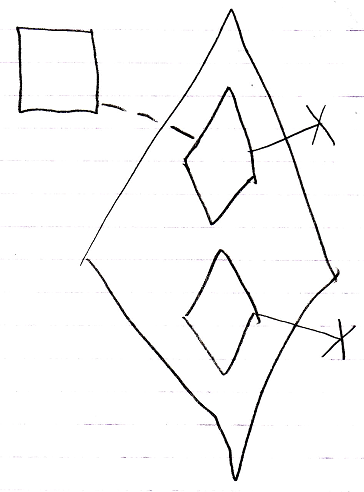
So a command call inside an object can be referenced, but a command call inside another command can not be referenced.

Not being able to reference sub-commands does not mean, that you can use it as a class, because a class reference is also a reference.

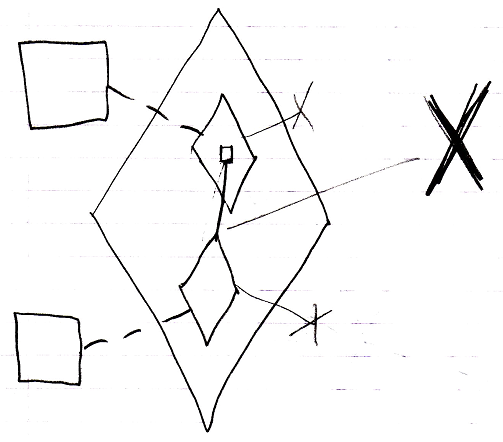


This would also put the sub-command in danger of being prematurely executed, because you could establish an active reference to the class of another command object and execute it.

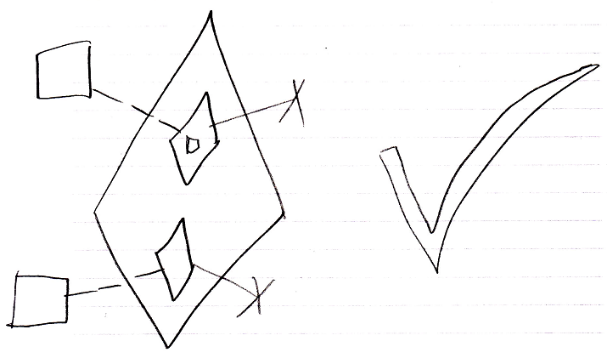
To not cause any confusion, sub-commands are made Private.



Do not change the rule to *sub-commands are always private*, because this would not fully solve the parent command’s control over its sub-commands’ execution. By just making them private, the parent command could still pass a reference to a sub-command.



That would make the parent *give up control* over the execution of a sub-command. This is something, that will not be allowed. Now that you can *never reference a sub-command*, this control is restored.



#### Beware of Active Command References in Commands

This is a rule for enforcement of control of a parent command over the execution of its sub-commands. It does not impose any restriction, but just advises you to beware of what you are doing, when you are using an active command reference inside a command.

When you are using an active command reference inside a command, then you are giving up the full control over this sub-command’s execution.

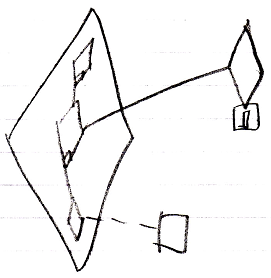
You have to be aware, that when you use an active command reference, the active command reference may already have been executed, or that you may be *waiting* on an external execution to finished. If the external execution won’t finish, then the referring procedure is stuck. If the external command object is an inactive command object, it will never execute, and the command will truly be stuck. A warning should be generated then.

This also has consequences for setting parameters for the active command reference. If it is already executing or has already executed, then you can not overwrite the parameters. This may mean, that it will simply not be allowed to set parameters for an active command reference at all.

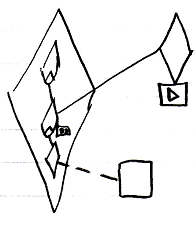
It is *not* the rule, to beware of *any* active command reference. It is the rule to just always beware of them in a *parent command*. An *object* can contain an active command reference, in order to for instance allow a user to carry around a reference to an active command, executing on a site somewhere.

##### Diagram Notation

You have to be aware, that when you use an active command reference, the active command reference may already have been executed.

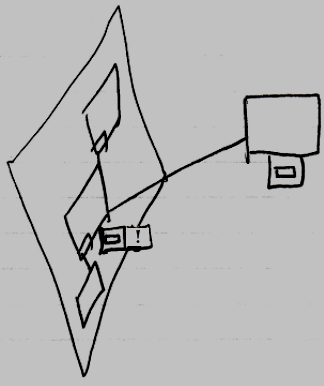


Or that you may be *waiting* on an external execution to finished.

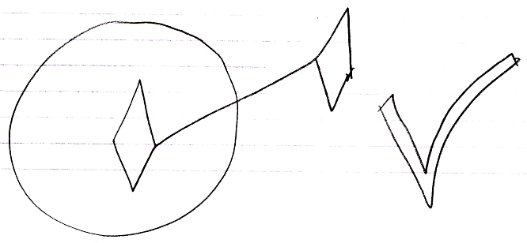


If the external execution won’t finish, then the referring procedure is stuck.

If the external command object is an inactive command object, it will never execute, and the command will truly be stuck. A warning should be generated then.



It is *not* the rule, to beware of any active command reference. It is the rule to just always beware of them in a *parent command*. An *object* can contain an active command reference, in order to for instance allow a user to carry around a reference to an active command, executing on a site somewhere.



#### Sub-Commands Not Manually Started

This is a rule for enforcement of control of a parent command over the execution of its sub-commands.

Sub-commands are never manually started by a user. Then there is no way for a user to run the sub-command at any arbitrary moment, and the parent command will keep control over the execution of its sub-commands.

### Command Referrers

**[Preliminary documentation]**

#### Command Object Referrers

The *Referrers* article explained how an object can be made aware of its referrers. The concept of referrers applies direction to command objects as well.

If a command and its references both support the Referrers concept, then every command reference will register itself in the Referrers list of the target command.

Refer to the article *Referrers* for an explanation on how the Referrers concept works.

#### Command Definition Referrers

The *Referrers* article explained how an object can be made aware of its referrers. A command is an object as well and the Referrers concept already provides a command with *command definition referrers* functionality. An inactive command is the only type of command symbol that can be referenced. Am inactive command symbol can implement the Referrers concept to register every call or reference to it.

When a site hosts a command definition, that is widely used all over the world, you might not want the command definition to register its referrers, because it would be a very long list to maintain. You can turn off the Referrers concept for any command definition.

If another site uses this widely used command definition, the using site could add a command reference to the command definition on the other site. A command reference has its own list of referrers. The using site could then redirect calls and references to its own command reference. Then the using site has a registration of anything on its site that uses the external command definition.

##### In a Diagram

< The expression of referrers in a diagram needs to be redone, because the referrers list refers to the parents of the references, which is not neccesarily the way to go. I’m not sure yet. I might want to register the related items and related lists items that are the references to the command definition, instead of registering their parents, and an ID, that the reference has inside the parent. >

### This

**[Preliminary documentation]**

#### Class . HasThis

This Boolean expresses if the Item Class will have a This property. For more information see the *This* section.

#### Class . This

The **This** member returns the object itself.

#### This

If you set the **Class.HasThis** property to **True**, then the class will get a **This** property, which will return the object itself. This is particularly handy in **With** blocks in which you wish to set another variable to the **With** block object. **Class.HasThis** is **True** by default.

* Using class procedures with the THIS argument

Don’t know if I should cover here:

* Using type procedures with the THIS argument (need a good name for the section)