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

## Start & Stop

An executable object can be run, paused, stopped, recorded, put in reverse, etcetera. Those are called *execution commands*.

You can also invoke an execution command on a thread, which is a string of commands, that automatically follow each other up. A thread, however, can also be considered a single command as well.

There is a distinction between an *execution command* and an *execution state*.

An executable object has a *state* of execution, for which the same terms are used as for the *execution commands*: play, stop, record, etcetera.

The following execution states and execution commands exist:

Previous

Rollback

Faster Backwards

Slower Backwards

Reverse

Play

Slower

Faster

Go To End

Next

Pause

Wait

Timer

Not Runnable

Stop

Finished

Record

Error

Warning

Which execution commands you can invoke, depends on the execution state the command is in. When a command has finished, you can not play it again. When a command is not runnable, you can not play it at all.

The following ones can be execution *commands*:

Previous

Rollback

Faster Backwards

Slower Backwards

Reverse

Play

Slower

Faster

Go To End

Next

Pause

Wait

Timer

Stop

Record

So that excludes the following ones from being an execution command:

Not Runnable

Finished

Error

Warning

The following ones can be execution states:

Rollback

Faster Backwards

Slower Backwards

Reverse

Play

Slower

Faster

Pause

Wait

Timer

Not Runnable

Stop

Finished

Record

Error

Warning

So that excludes the following ones from being an execution state:

Previous

Go To End

Next

There is a big difference in difficulty of implementing the execution commands. This is irrelevant to their usage, but will affect their availability in early stages of development of the new computer language. There are five categories of difficulty of implementation of execution commands and execution states. Here they are going from simple to difficult:

- Play, stop, next, previous, not runnable, finished

- Foreward at arbitrary speeds

- Waiting

- Error and warning

- Record

- Going in reverse

The states Play, Stop, Next, Previous, Not Running and Finished are the most basic and more easy to implement. They are implemented first. Commands going in reverse are the worst in complexity. They include Reverse, Slower Backwards, Faster Backwards and Rollback. They are rollback and undo functionality, that is much harder to implement. So the execution commands will not all be available in one blow.

Below each execution command and execution state is explained separately.

### Previous Execution Command

The Previous execution command immediately stops the execution of the current command and positions the instruction position to the command previous in line. And if there is no previous sibbling command, there will be moved a level up, to the parent command.

The Previous execution command is only available if there is a previous command to go to.

The Previous command will cause a Stop state on the previous command for computation and might cause a Play state on the previous item for playable media, making it start to run immediately.

### Rollback Execution Command

The Rollback execution command will undo all changes of the current command and makes all data return to the state it was before the command had started. If the same data has already been changed by another source it may be overwritten by the original value it had before running the command or the change on that particular data item may not be rolled back at all, or some sort of concurrency resolution must be applied, but concurrency issues will be dealt with later.

In case of regular computation, a Rollback will be executed as fast as possible. For video, audio or presentation or any other type of running media, Rollback means immediately going to the beginning of the execution.

The Rollback execution command is only available if the command has started at all.

The Rollback command will cause a Rollback state on the current command.

### Rollback Execution State

After a Rollback has executed, the command will be in Rollback state (rolled back), which is actually a Stop state, but then after a Rollback.

### Faster Backwards Execution Command

The Faster Backwards execution command will start or continue a rollback in a fast pace or faster pace or faster, but never entirely at top speed.

The Faster Backwards execution command is only available if the command is finished or right in the middle of executing or rolling back.

The Faster Backwards execution command will cause a Faster Backwards, Reverse or Slower Backwards execution state (if it is going faster but still slow).

### Faster Backwards Execution State

A Faster Backwards (going fast backwards) execution state is the system’s ‘subjective’ opinion of whether the current speed of going backwards is faster than average. In that case the command, thread or playable medium will get Faster Backwards execution state.

### Slower Backwards Execution Command

The Slower Backwards execution command will start or continue a rollback in a slow pace, slower pace, or as slow ‘as possible’.

The Slower Backwards execution command is only available if the command is finished or right in the middle of running.

The Slower Backwards execution command will cause a Slower Backwards, Reverse or Faster Backwards execution state.

### Slower Backwards Execution State

A Slower Backwards execution state (going slower backwards) is the system’s ‘subjective’ opinion of whether the current speed of going backwards is slower than average. In that case the command, thread or playable medium will get Slower Backwards execution state.

### Reverse Execution Command

The Reverse execution command will start a rollback at normal speed. In case of regular computation, normal speed would be top speed. For video, audio or presentation or any other type of running media, normal is not as fast as possible, but the normal speed of presentation but then in reverse.

The Reverse execution command is only available if a command is finished or anywhere in the middle of running.

The Reverse command causes the execution to get the Reverse execution state.

### Reverse Execution State

A command or other executable object gets the Reverse execution state if it is rolling back at normal speed. For regular computation, normal speed is top speed. For playable media, normal reverse speed is the same speed as normal foreward speed, but then in reverse.

If the speed is slower or faster than considered average, the execution state will be set to either Faster Backwards or Slower Backwards.

### Play Execution Command

The Play execution command will start or continue execution at normal speed. In case of regular computation, normal speed would be *top* speed. For video, audio or presentation or any other type of running media, normal is not as fast as possible.

The Play execution command is only available if a command is runnable or anywhere in the middle of running.

The Play execution command will cause the Play execution state.

### Play Execution State

Play is the execution state a command gets when going at normal speed. In case of regular computation normal speed is *top* speed. In case of playable media normal speed is the normal playing speed of the medium.

### Slower Execution Command

The Slower execution command will make the current execution speed slow, or slower, but never come to a complete halt. If the command is in a rollback, it will make the command start to go foreward again.

The Slower execution command is only available if the command is runnable or in right in the middle of running.

The Slower execution command will cause a Slower, Play or Faster execution state.

### Slower Execution State

A Slower execution state is the system’s ‘subjective’ opinion of whether the current speed of going foreward is slower than average. In that case the command, thread or playable medium will get Slower execution state.

### Faster Execution Command

The Faster execution command will make the current execution speed fast, or faster, but never top speed. If the command is in a rollback, it will make the command start to go foreward again.

The Faster execution command is only available if the command is runnable or even right in the middle of executing or rolling back, which will turn around the direction.

The Faster execution command will cause a Faster, Play or Slower execution state.

### Faster Execution State

A Faster execution state is the system’s ‘subjective’ opinion of whether the current speed of going foreward is faster than average. In that case the command, thread or playable medium will get Faster execution state.

### Go To End Execution Command

The Go To End command will make the command fast-foreward to the end at top speed. For normal computation this is the same as Play, except when normal speed is set to slower than top speed. For runnable media this is not the same as play, but going to the end of the execution as fast as possible.

The Go To End execution command eventually causes the Finished exection state, once it is done going foreward as fast as possible. Until then, the execution state is Faster.

### Next Execution Command

The Next execution command immediately stops the execution of the current command and positions the instruction position to the next command in line. And if there is no next sibbling command, there will be moved a level up, to the parent command.

The Next execution command is only available if there is a next command to go to.

The Next command will cause a Stop state on the next command for computation and might cause a Play state on the next item for playable media.

### Pause Execution Command

The Pause command pauses execution at the point it is at, until you press Play or another command again.

The Pause command is available when a command is anywhere in the middle of executing.

A Pause can also be put on an executable object, that is not yet executing. In that case, the system will pause as soon as the object *is* going to execute.

You can also a Pause on a definition. In that case any call to that definition will cause the system to pause.

A Pause can also be set on a sub-command inside a definition. Then, each call to the definition will pause on that particular spot in the execution.

The Pause execution command causes a Pause execution state.

### Pause Execution State

See the section above: *Pause Execution Command*. A command will get the Pause state if a Pause was invoked on it.

### Wait Execution Command

It may be useful for an execution to be able to wait on another execution.

The Wait execution command is like Pause, but then another executable command or a thread is selected to wait on. As soon as the other executable command is done or the selected thread has finished, the execution of the waiting command or thread continues.

The Wait command is available when a command is anywhere in the middle of executing.

The Wait execution command causes the Wait execution state.

The implementation may use the fact that a single executable object might have multiple next-command references. The waiting command could register itself as a next command of another command in order to wait on it.

### Wait Execution State

An executable object will get the Wait state if a Wait execution command was invoked on it.

### Timer Execution Command

The Timer command is not really an execution command. It lets you schedule the execution of a command. You could schedule it at any selected moment in time, seconds, or month from now or repeatedly have the command run at set times.

A Timer can be set on an executable object, but then it can only be run once. A Timer can also be set on a definition, which will create a call to it at every time it is scheduled.

A Timer execution command will cause the Timer execution state. At the times it is scheduled it will turn into a Play state. At times, that it is not running, it is put in the Timer state again and if it is no longer scheduled in the future anymore, it will get the Finished state.

Timer might be a good excuse to have an inactive command class redirect to another inactive command, and time the redirector, to cause a scheduling of a command definition, that resides elsewhere.

You can also put a Timer on a Record command, instead of a Play command, to start recording something at a set time.

### Timer Execution State

An executable object gets the Timer state if the executable object is scheduled to execute in the future at a given time.

### Not Runnable Execution State

A command definition will often have the Not Runnable execution state. Command definitions are mostly inactive command objects. Inactive command object can never be run, because inactive command objects are usually command definitions and running a command definition may cause the command definition’s default values to change, while a command definition is considered to be static and unchanging. That’s why inactive command objects will get the execution state Not Runnable. Any inactive reference to a command object is also Not Runnable.

### Stop Execution Command

The Stop command will immediately stop the execution of the command or thread. This can be dangerous, because it is better to either let a command finish completely or roll it back entirely, and not leave it in some intermediate state. But in a lot of other cases it is not much of a problem.

But it should be possible to make a command ‘unstoppable’, sort of like a database transaction, so that it can either be entirely completed or completely rolled back, but never permanently stopped right in the middle.

The Stop command is only available when a command is right in the middle of running and if the command is not marked as unstoppable.

The Stop command can also be used to stop a recording. In that case it does not have the danger as described earlier.

The Stop command will cause the current command to be put in Stop state.

### Stop Execution State

The Stop execution state is not as dangerous as the Stop execution command. The Stop execution command will end a procedure right in the middle. But if an executable object has not been run yet, it also has the Stop state. After pressing Stop, a command will indeed also end up in the Stop state, but a command also has this state if it has not been run before.

### Finished Execution State

The Finished execution state is there, when the command has completed normally in its entirety. In that case the executable command can not be run again. A playable medium, though, *can* be run again after it has finished.

### Record Execution Command

The Record command can replace a command definition, with whatever action a user or a thread might perform, until recording is stopped.

The Record execution command will cause a command to be put in Record state.

### Record Execution State

An executable object will get the Record state when it is recording, after a Record execution command was invoked on it.

### Error Execution State

An error may cause an immediate rollback in the new system, but perhaps a program might also pause. Perhaps only in debugging state. In that case, when an error occurs, the execution stops on the current command and the error puts it in the Error execution state.

### Warning Execution State

When debugging and something uncommon happens, the execution might be put in Warning state, but this actually does not have to hinder execution, so one could think of a situation in which an execution is in Warning state, but still continues to run.

Any object or reference line could also be put in Warning state if something exceptional, but benign, is going on with it.

### Normal Speed

In a normal situation, normal speed for computation is top speed, and normal speed for a playable medium is the normal speed to play the medium. But you can indicate an alternate normal speed. If you are reviewing a playable medium, that you will be needing to look at in slow motion, you can make slow motion the normal speed. In that case Play means going in slow motion, and Faster means it is going faster than the normal rate of slow motion. If you are debugging, you might also set the normal speed of computation to slower. In that case you can visually follow the execution of the procedures. This normal rate of slow motion will also be the barrier for what the states Slower and Faster will consider the average speed.

### Diagram Notation

Each of the following commands or states has a symbol:

Previous

Rollback

Faster Backwards

Slower Backwards

Reverse

Play

Slower

Faster

Go To End

Next

Pause

Wait

Timer

Not Runnable

Stop

Finished

Record

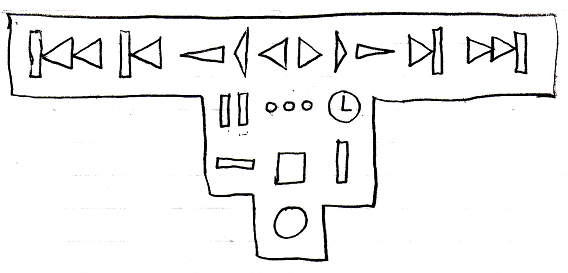
Error

Warning

The symbols are just about the same as those of media players of today:

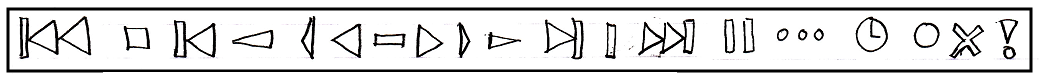
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Previous | Rollback | Faster Backwards | Slower Backwards | Reverse |
|  |  |  |  |  |
|  |  |  |  |  |
| Play | Slower | Faster | Go To End | Next |
|  |  |  |  |  |
|  |  |  |  |  |
| Pause | Wait | Timer | Not Runnable | Stop |
|  |  |  |  |  |
|  |  |  |  |  |
| Finished | Record | Error | Warning |  |

The commands can be displayed at the bottom of the screen to execute a selected command or on a thread, that you are following.



The order of the symbols may become different in practice. They will probably put in a straight line. Some of them can not even be execution *commands*, but can only be execution *states*, so those will be left out of the bottom of the screen.

Another example display of the execution commands and execution states:



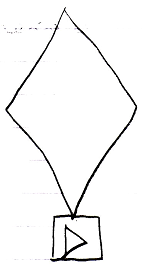
The following symbols are also possible candidates:

Those symbols could set the speed of going foreward or backward, gradually changing it from Faster Backwards, Reverse, Slower Backwards and to Slower, Play, Faster. The first symbol simply decreases the speed or makes it go faster backwards, and the second symbol simply increases the speed or makes it go slower backwards.

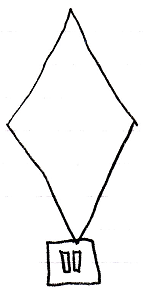
The symbols for actions, that a media player also has, were retained, because I just love symbolic representation and I do not see a need to invent any other symbolic display for this.

An execution state is also be displayed at the bottom of a command symbol, to indicate its state:

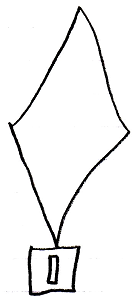


The executable command above is currently running at normal speed, since it is in Play state. Here are more examples of executable command object in different states.

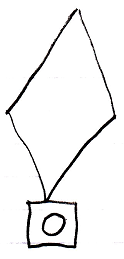
The following command is paused:



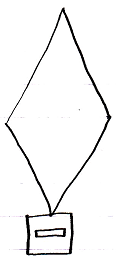
The following command is finished:



The following command is recording:



The following command is not executable at all:



Any execution state can be indicated for a command object like that.

### Implementation

Do realize, that there is a difference in difficulty of implementation of the execution commands and execution states. Do not try to implement all the execution states and commands in one blow. There are 5 categories of difficulty of implementation of execution commands and execution states. Here they are going from easy to hard:

- Play, stop, next, previous, not runnable, finished

- Foreward at arbitrary speeds

- Waiting commands

- Record command

- Commands going in reverse

The states Play, Stop, Next, Previous, Not Running and Finished are implemented first. They are the easiest. Going foreward at arbitrary speeds includes Slower, Faster and Go To End. They are nice for slowly running the steps, so you can follow better. Then there are the waiting commands, which are not that hard either. In fact it is quite important to be able to pause, so that one will be implemented sooner. The Record command is easier, than commands going in reverse, but it is much less important than the commands going in reverse. And commands going in reverse are the worst. They are rollback and undo functionality, that is much harder to implement. They include Reverse, Slower, Backwards, Faster Backwards and Rollback. Do not implement them all in one blow.