Animated Timer Manual - Version: 240804 - AT Version: 1.0.0

Table of content

Animated Timer Manual - Version: 240804 - AT Version: 1.0.0

General : Introduction

Animated Timer (AT) data pack adds an in-game timer via the actionbar field of Minecraft's HUD. A wide range of functions ensures a simple yet versatile configuration, allowing you to customize your timer according to your likings and needs.

General : Features

In its original and unmodified state, features include:

- Count-up & count-down
- Multiple render (display) types
- Both static (never changing) and animated (always changing) color configuration
- General style configuration (bold & italic text)
- Pre-built animations
- Profiles (ability to save current configuration & load at a later time)
- Trigger scoreboards for complete configuration & interaction even if cheats are disabled
- Advanced features for map makers (facilitated embedding into external projects)

Basics : Controlling the timer

AT can be controlled the following way:

Description	Cheats enabled	Without cheats
Continue / resume the timer in the mode in which it was last active OR start it initially	`/function at:continue`	`/trigger continue`
Pause the timer	`/function at:pause`	`/trigger pause`
If counting up: Set time to zero and start counting up immediately If counting down: Restore last manual time setting and start counting down from there	`/function at:reset`	`/trigger reset`
Set time to zero and stay paused – next continue will always start counting upwards	`/function at:reboot`	`/trigger reboot`

Animated Timer Manual - Version: 240804 - AT Version: 1.0.0

Basics : Manually setting time

In some situations, you may want to alter the current time manually. This can be done either with a function in the at-time namespace or with */trigger*:

Description	Cheats enabled	Without cheats

Configure seconds	`/function at-time:add/second`
	`/function at-time:remove/second`
	`/function at-time:set/seconds/10`
		`/trigger seconds set 10`

Configure minutes	`/function at-time:add/minute`
	`/function at-time:remove/minute`
	`/function at-time:set/minutes/15`
		`/trigger minutes set 15`

Configure hours	`/function at-time:add/hour`
	`/function at-time:remove/hour`
	`/function at-time:set/hours/3`
		`/trigger hours set 3`

Configure days	`/function at-time:add/day`
	`/function at-time:remove/day`
	`/function at-time:set/days/7`
		`/trigger days set 7`

Note: The above functions that include *at-time:set* are just examples. In fact, a multitude of *set* functions are available for use.

Basics : Timer modes

AT is counting up by default but count-down is also available. You can toggle between both count modes with a single command:

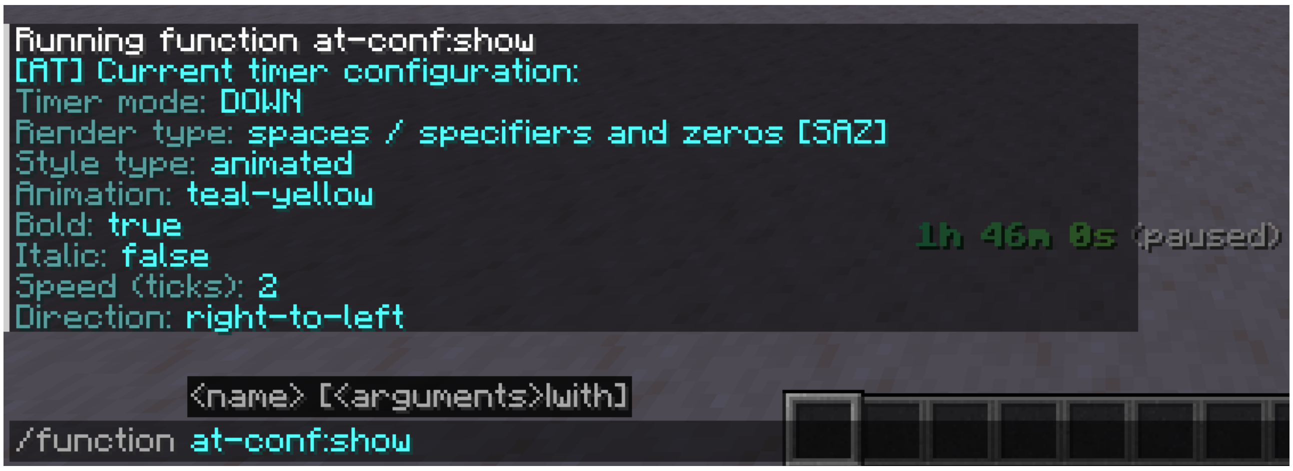
Description	Cheats enabled	Without cheats
Switch between timer modes	`/function at-conf:mode`	`/trigger mode`

Advanced : Configuration

Part of what makes AT so unique is its versatile configuration, which you can print at any time.

Description	Cheats enabled	Without cheats
View current configuration	`/function at-conf:show`	`/trigger show`

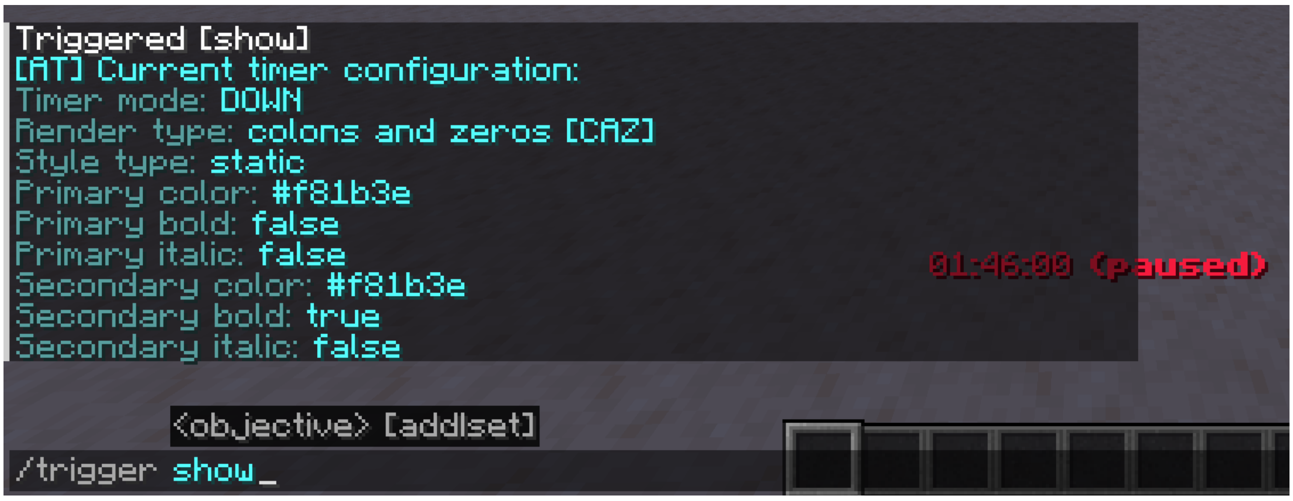
The output will look something like this, depending on what settings your currently use:



```
Running function at-conf:show
[AT] Current timer configuration:
Timer mode: DOWN
Render type: spaces / specifiers and zeros [SAZ]
Style type: animated
Animation: teal-yellow
Bold: true
Italic: false
Speed (ticks): 2
Direction: right-to-left
```

```
/function at-conf:show
```

Here is another example, this time using the static style type:



```
Triggered [show]
[AT] Current timer configuration:
Timer mode: DOWN
Render type: colons and zeros [CAZ]
Style type: static
Primary color: #f81b3e
Primary bold: false
Primary italic: false
Secondary color: #f81b3e
Secondary bold: true
Secondary italic: false
```

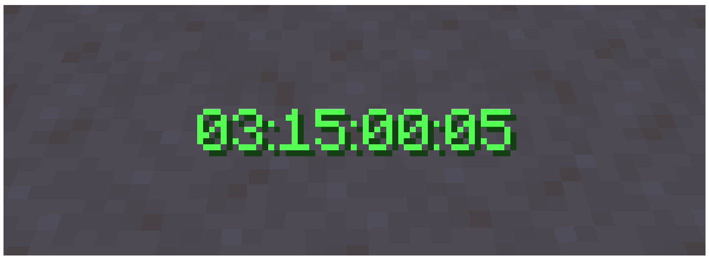
```
/trigger show_
```

Advanced : Configuration : Render types

The render type defines the shape of your timer. There are three render types in total:

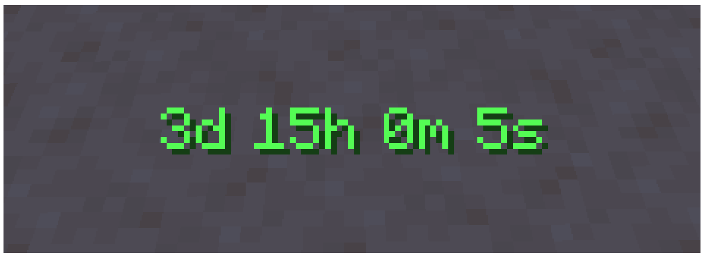
Colons and zeros [CAZ]

- Looks like this: DD:HH:MM:SS
- Every timer unit is always padded to two characters
- If an unit matches zero, it is shown



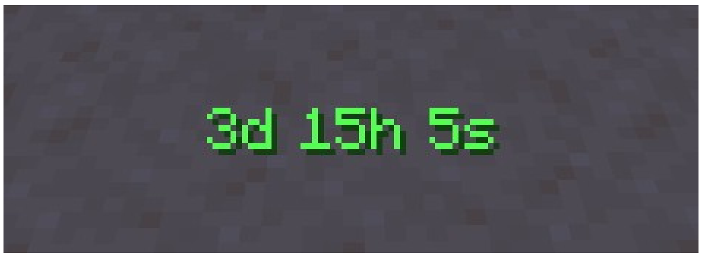
Spaces / specifiers and zeros [SAZ]

- Looks like this: DDd HHh MMm SSs
- No padding
- If an unit matches zero, it is shown



Spaces / specifiers no zeros [SNZ]

- Looks like this: DDd HHh MMm SSs
- No padding
- If an unit matches zero, it is hidden



To toggle between them, use one of these commands:

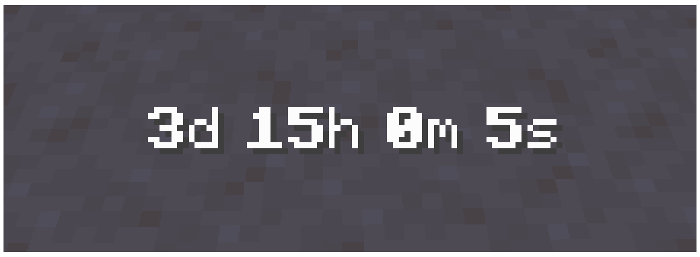
Description	Cheats enabled	Without cheats
Toggle render type	`/function at-conf:render`	`/trigger render`

Advanced : Configuration : Style types

The style type determines what color your timer has and what type of text design settings can be applied. There are two style types in total, static and animated.

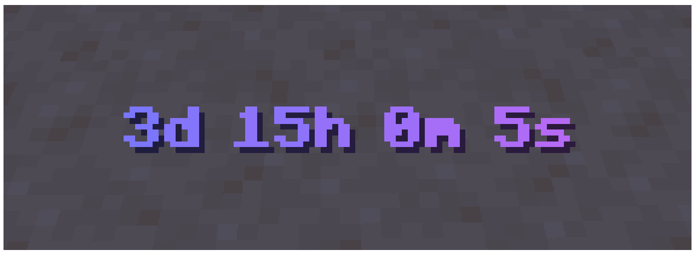
Static

- Unchanged, constantly has the same color
- Splits timer output into two individually configurable zones, primary and secondary



Animated

- Gets its color from an animation
- Timer output is treated as one style zone



Again, if you want to switch between them, use one of these commands:

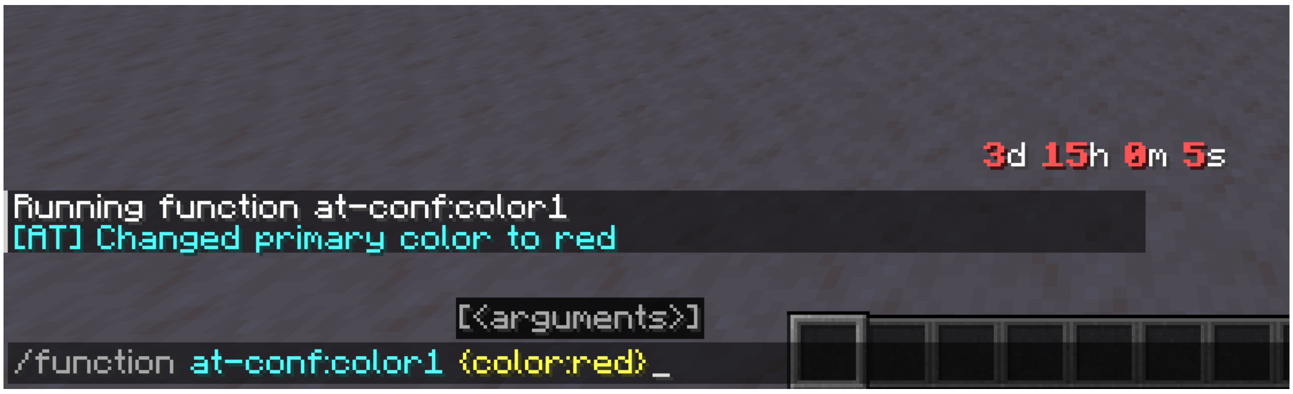
Description	Cheats enabled	Without cheats
Toggle style type	`/function at-conf:style`	`/trigger style`

Note: Most of the configuration commands below only work with one of the two style types. Be sure to first select your style type before configuring colors, animations, etc.

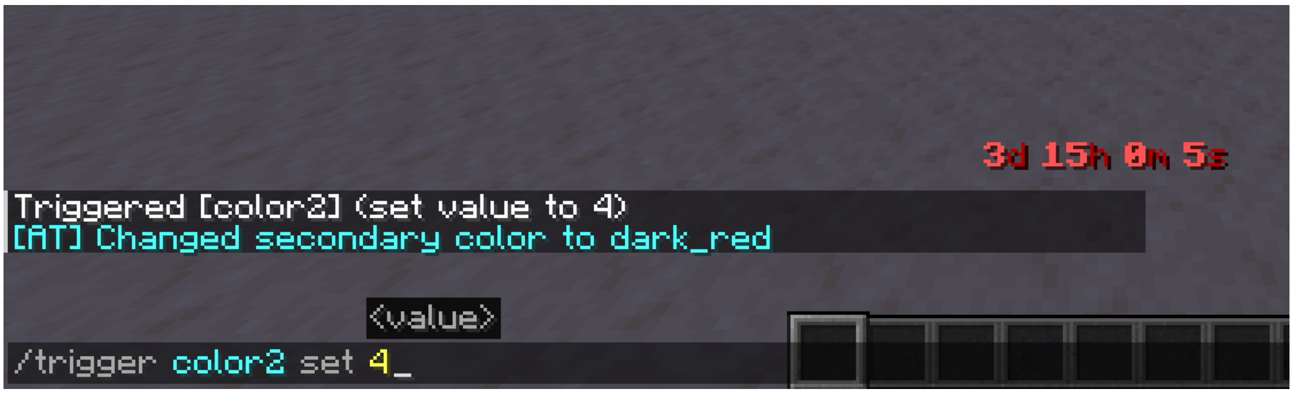
Advanced : Configuration : Colors

As mentioned before, a static style splits the timer into two areas that can be configured seperately. This level of seperation applies to colors as well as to other style settings. Changing the color works as follows:

Description	Cheats enabled	Without cheats
Change color of primary zone	`/function at-conf:color1 {color:aqua}`	`/trigger color1 set 2`
Change color of secondary zone	`/function at-conf:color2 {color:"#C63C51"}`	`/trigger color2 set 13`



```
3d 15h 0m 5s
Running function at-conf:color1
[AT] Changed primary color to red
[<arguments>]
/function at-conf:color1 {color:red}_
```



```
3d 15h 0m 5s
Triggered [color2] (set value to 4)
[AT] Changed secondary color to dark_red
<value>
/trigger color2 set 4_
```

Updating the color via */trigger* uses Minecraft's in-game color codes. An overview of said colors can be found here: https://minecraftitemids.com/color-codes

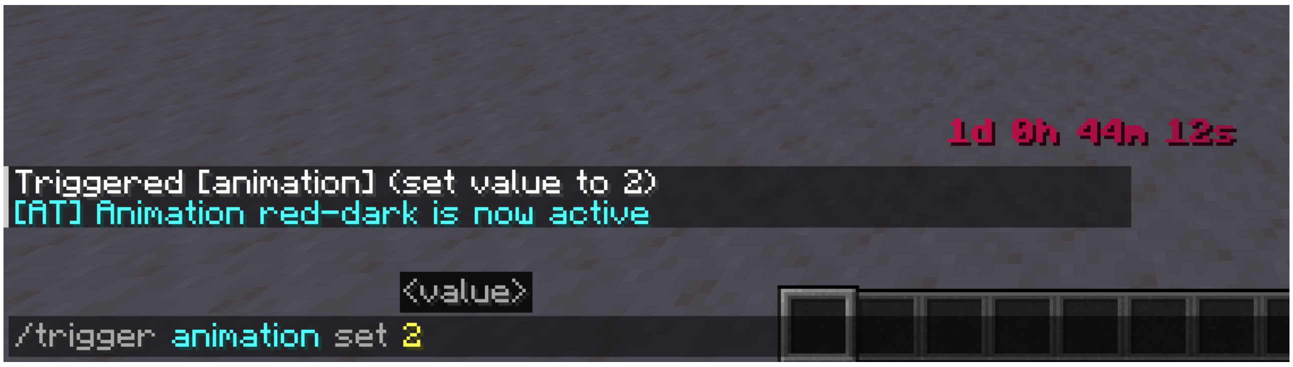
Advanced : Configuration : Animations

By default, AT comes with 11 different animations that are available for you to use. Changing the animation can be achieved like this:

Description	Cheats enabled	Without cheats
Change animation	`/function at-conf:animation {name:red-peach}`	`/trigger animation set 2`

As with colors, */trigger* does not accept any text input. Despite this, selection of your favorite animation is still possible due to the scores being mapped to animations 1-11. See below:

Animation	Trigger value
red-peach	1
red-dark	2
red-purple	3
blue-pink	4
pastel-purple-blue	5
blue-mint	6
purple-mint	7
green-blue	8
teal-yellow	9
teal-lime	10
rainbow	11



```
Triggered [animation] (set value to 2)
[AT] Animation red-dark is now active
/trigger animation set 2
```

Additionally, animation speed and direction may also be configured:

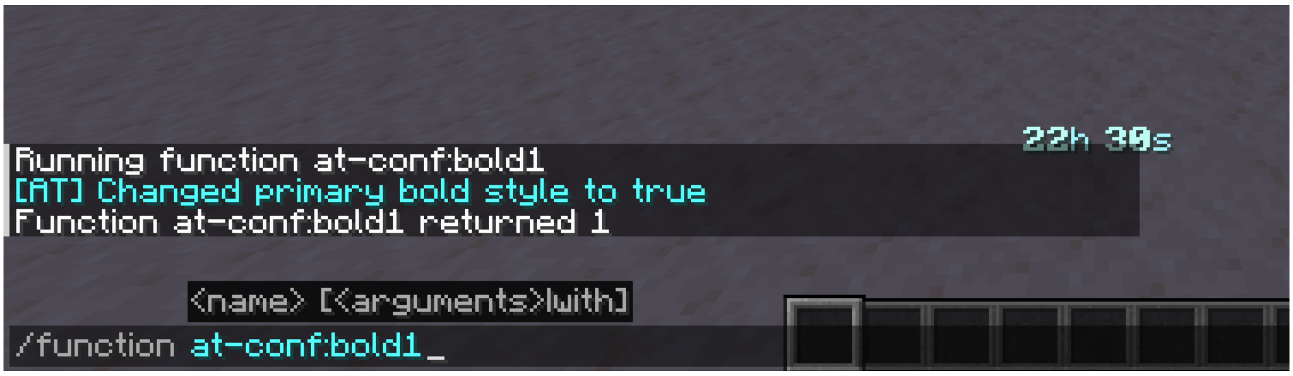
Description	Cheats enabled	Without cheats
Modify animation speed	`/function at-conf:speed {ticks:4}`	`/trigger speed set 2`
Modify animation direction	`/function at-conf:direction`	`/trigger direction`

Note: A higher speed value means that the animation runs more slowly; the fastest animation runs at speed 1.

Advanced : Configuration : Style configuration

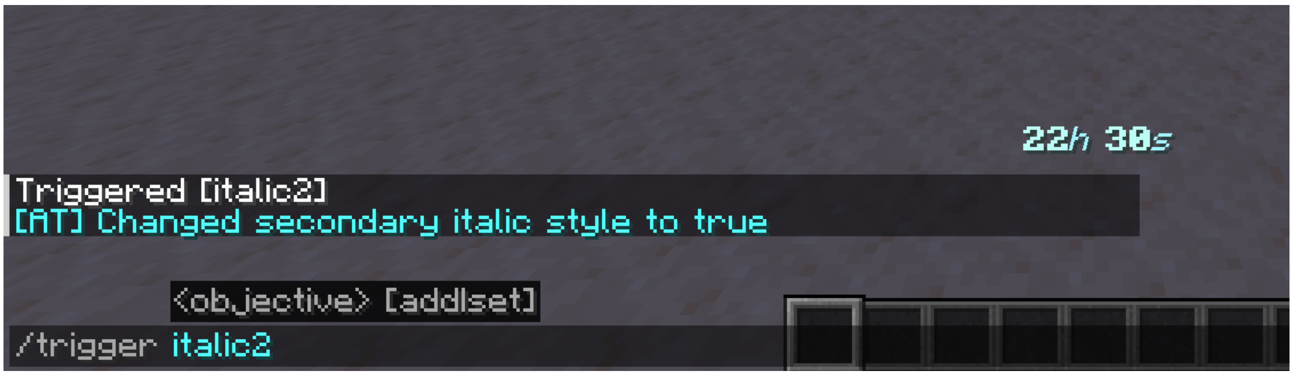
Both the bold and italic design setting belong to this category. Depending on the current style type, the following commands behave differently when executed.

Description	Cheats enabled	Without cheats
Static style type: Modify bold style of primary zone Animated: Modify bold style of current animation	`/function at-conf:bold1`	`/trigger bold1`
Static style type: Modify bold style of secondary zone Animated: Modify bold style of current animation	`/function at-conf:bold2`	`/trigger bold2`
Static style type: Modify italic style of primary zone Animated: Modify italic style of current animation	`/function at-conf:italic1`	`/trigger italic1`
Static style type: Modify italic style of secondary zone Animated: Modify italic style of current animation	`/function at-conf:italic2`	`/trigger italic2`



```
Running function at-conf:bold1
[AT] Changed primary bold style to true
Function at-conf:bold1 returned 1
```

```
/function at-conf:bold1_
```



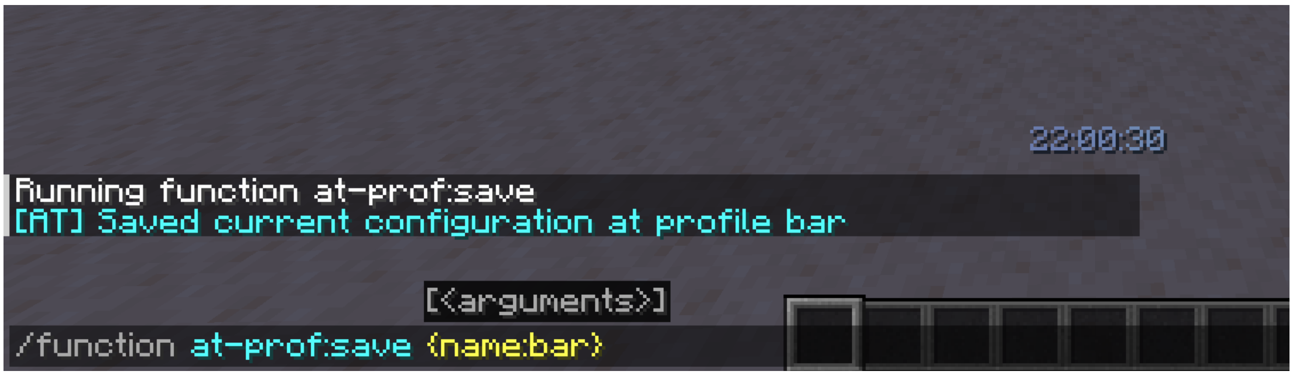
```
Triggered [italic2]
[AT] Changed secondary italic style to true
```

```
/trigger italic2
```

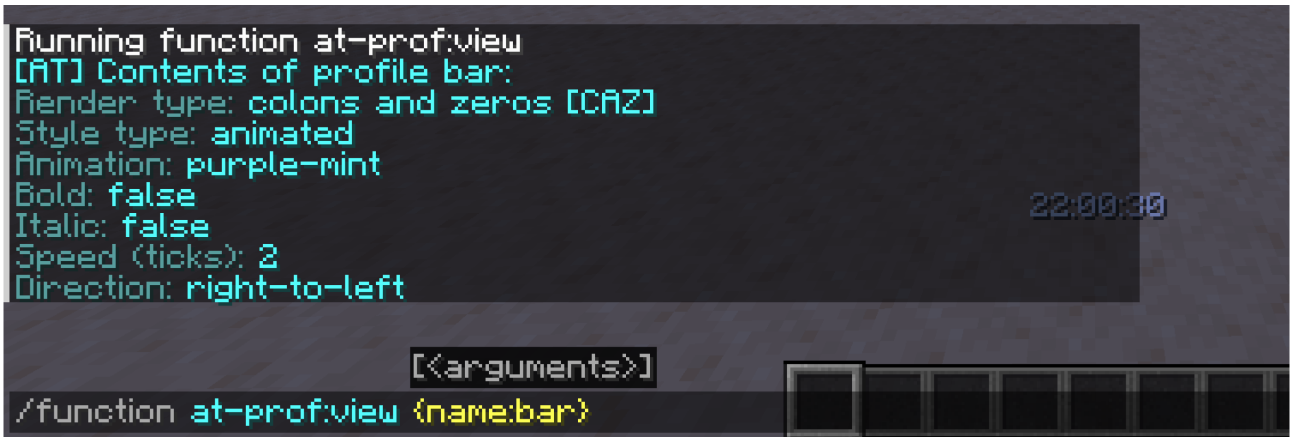
Advanced : Profiles

Profiles allow you to store the current configuration and load at a later time.
By default, AT comes with 10 different profiles - the maximum amount of profiles one can have is 64. Sufficient profile management is achieved using the following commands:

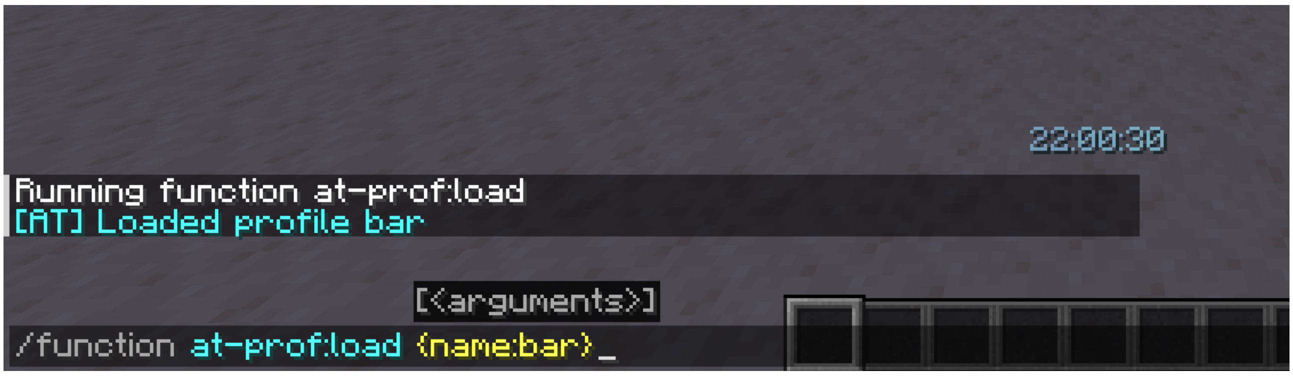
Description	Cheats enabled	Without cheats
Save current configuration at profile <name>	`/function at-prof:save {name:foo}`	`/trigger save set 1`
View configuration of profile <name>	`/function at-prof:view {name:foo}`	`/trigger view set 3`
Load profile <name>	`/function at-prof:load {name:foo}`	`/trigger load set 5`
Remove profile <name>	`/function at-prof:delete {name:foo}`	`/trigger delete set 7`
List existing profiles	`/function at-prof:list`	`/trigger list`



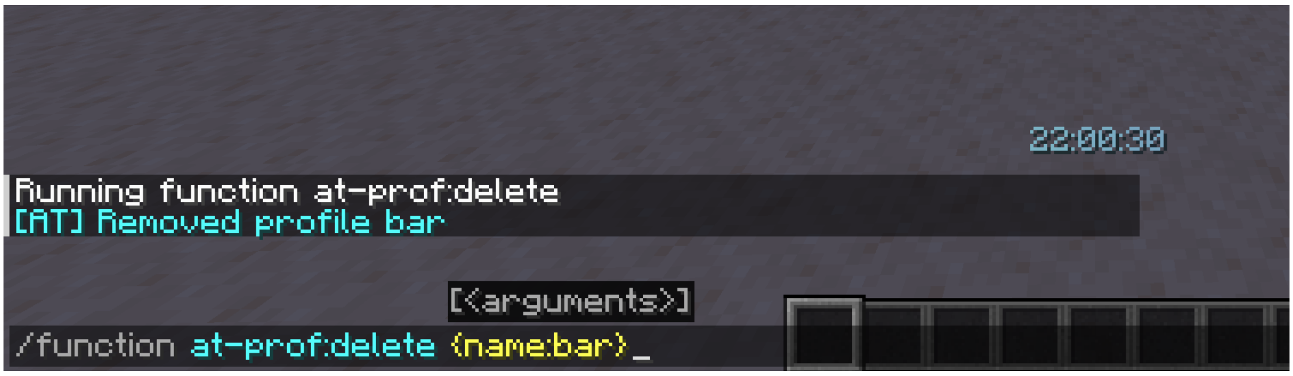
```
22:00:30
Running function at-prof:save
[AT] Saved current configuration at profile bar
[<arguments>]
/function at-prof:save {name:bar}
```



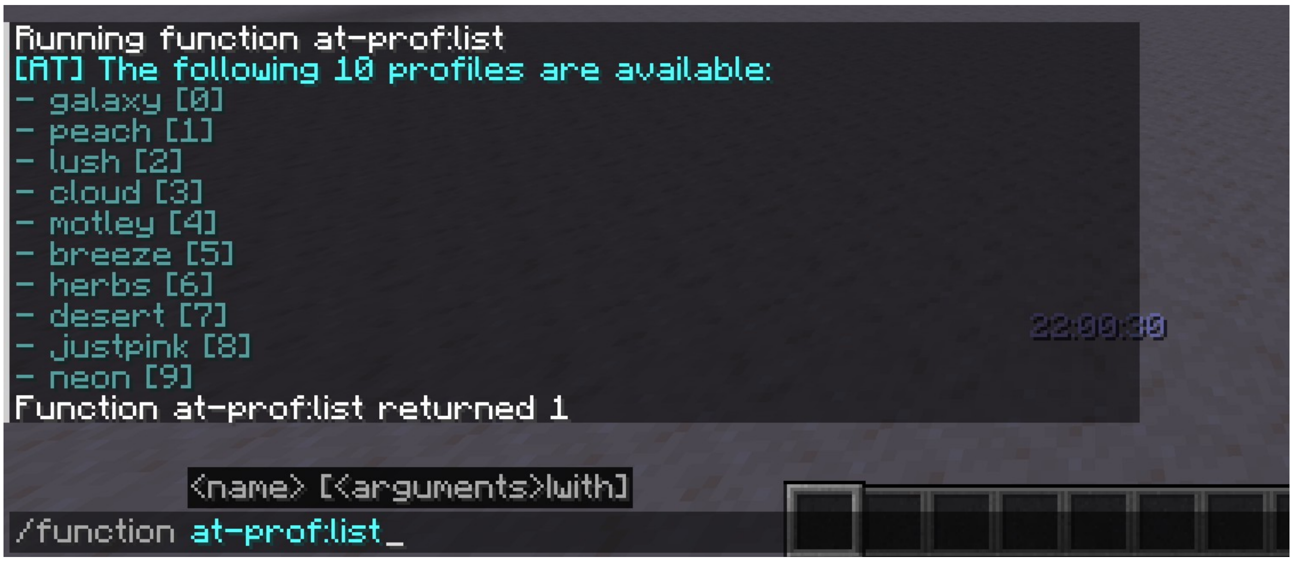
```
Running function at-prof:view
[AT] Contents of profile bar:
Render type: colons and zeros [CAZ]
Style type: animated
Animation: purple-mint
Bold: false
Italic: false
Speed (ticks): 2
Direction: right-to-left
22:00:30
[<arguments>]
/function at-prof:view {name:bar}
```



```
22:00:30
Running function at-prof:load
[AT] Loaded profile bar
[<arguments>]
/function at-prof:load {name:bar}_
```



```
22:00:30
Running function at-prof:delete
[AT] Removed profile bar
[<arguments>]
/function at-prof:delete {name:bar}_
```



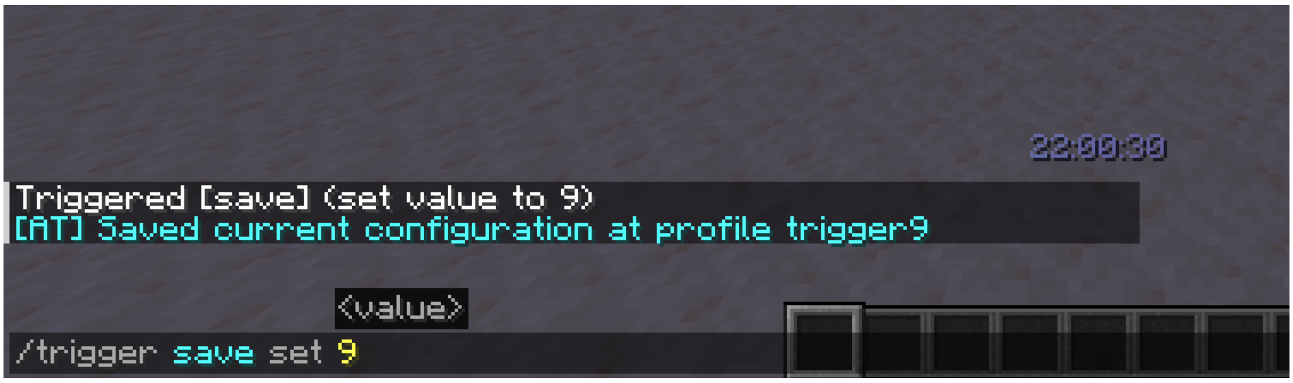
```
Running function at-prof:list
[AT] The following 10 profiles are available:
- galaxy [0]
- peach [1]
- lush [2]
- cloud [3]
- motley [4]
- breeze [5]
- herbs [6]
- desert [7]
- justpink [8]
- neon [9]
Function at-prof:list returned 1
<name> [<arguments>|with]
/function at-prof:list_
```

While profile management using */function* is consistent, meaning that you can save, load, etc to / from the same name, and will always access the same profile, management using */trigger* is not. Its designed this way to give players without cheats the ability to access existing (pre-made) profiles.

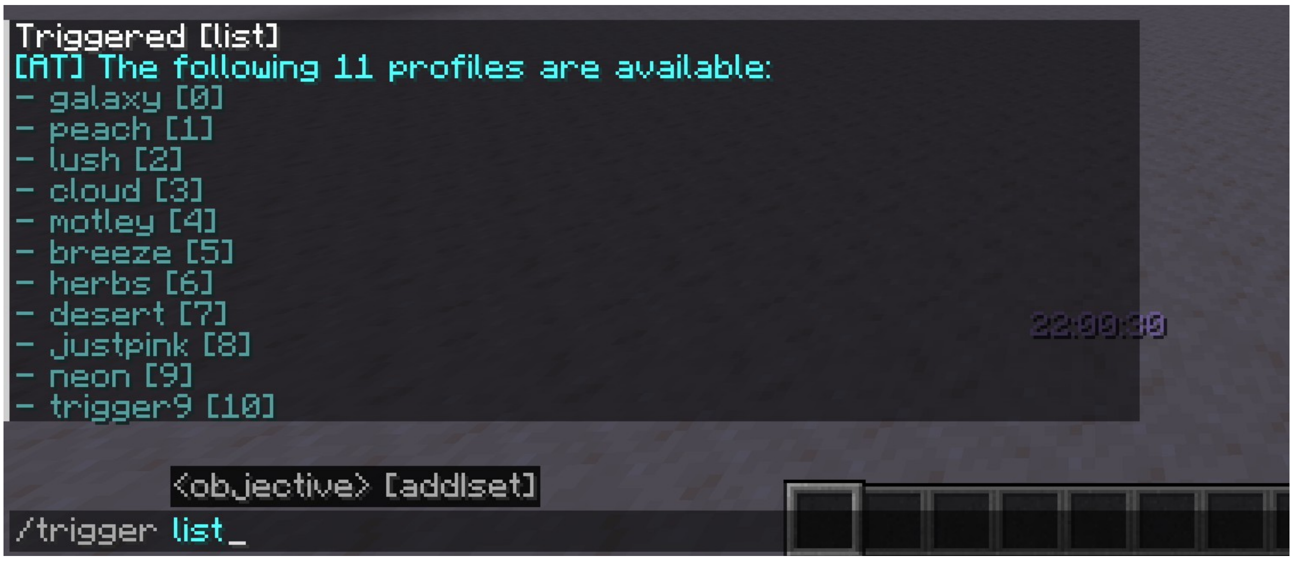
With triggers it works like this: The score you enter at the *save* gets appended behind “trigger”, forming a unique name like “trigger0” or “trigger42”, so saving works as expected.

Loading is a bit different though. To load the profile, you first need to know its index, which you can obtain by entering the *list* command. (the number in square brackets []) If you have decided what profile you want to have, run the *load* command with the index behind your profile. The correct profile should now be loaded. The same principle also applies to the *view* and *delete* command.

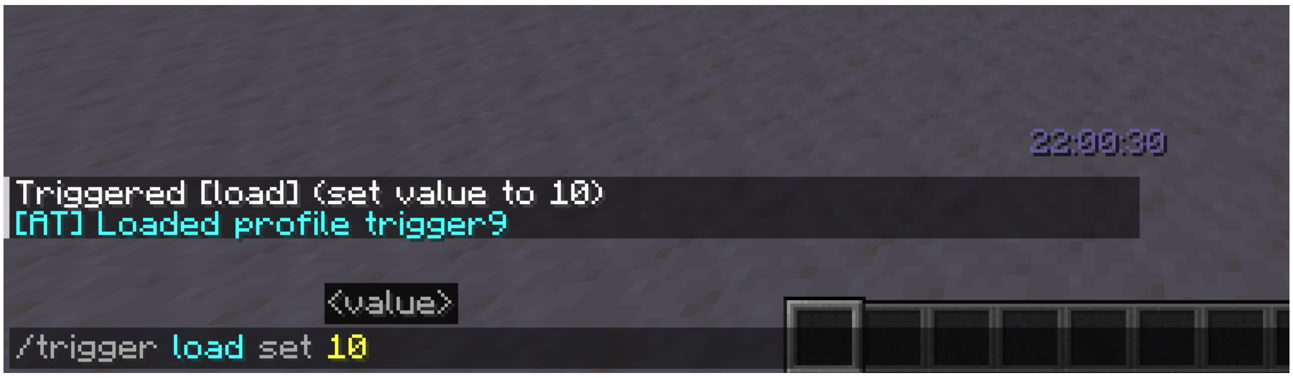
This functionality may change at a later time, if / when I have a better idea for solving this issue.



```
22:00:30
Triggered [save] (set value to 9)
[AT] Saved current configuration at profile trigger9
<value>
/trigger save set 9
```



```
Triggered [list]
[AT] The following 11 profiles are available:
- galaxy [0]
- peach [1]
- lush [2]
- cloud [3]
- motley [4]
- breeze [5]
- herbs [6]
- desert [7]
- justpink [8]
- neon [9]
- trigger9 [10]
22:00:30
<objective> [add|set]
/trigger list_
```



```
22:00:30
Triggered [load] (set value to 10)
[AT] Loaded profile trigger9
<value>
/trigger load set 10
```

Details : Function namespaces

AT implements 10 namespaces, 9 of them are AT-exclusive.

Namespace	Class	Description / purpose
at	UI	Timer controls
at-abar	internal	Originally just actionbar, but now place for misc
at-conf	UI	Timer / actionbar configuration
at-dcfg	UI / external	Direct Configuration - instantly modify configuration without messing around with config toggles from at-conf
at-dext	external	Do External - Collection of functions that run when specific things happen within the timer (useful for external projects)
at-main	internal	Timer backend
at-prof	UI	Profile management
at-time	UI	Timer controls
at-trig	internal	Trigger backend
minecraft	internal	Minecraft namespace; running tick and load

Details : Scoreboards

At load, AT adds 33 scoreboards, most of them are triggers.

Objective	Criteria	Description / purpose
at_s_state	dummy	Stores current & initial timer mode (UP or DOWN)
at_s_config	dummy	Most of current (running) config - rest is in NBT storage
at_s_animation	dummy	Needed for rotating through the animation
at_s_profiles	dummy	Literally just the current profile count
at_c_time	dummy	That’s the timer
at_t_interact	dummy	Temporary storage for UI interaction
at_t_profout	dummy	Temporary storage for viewing profiles

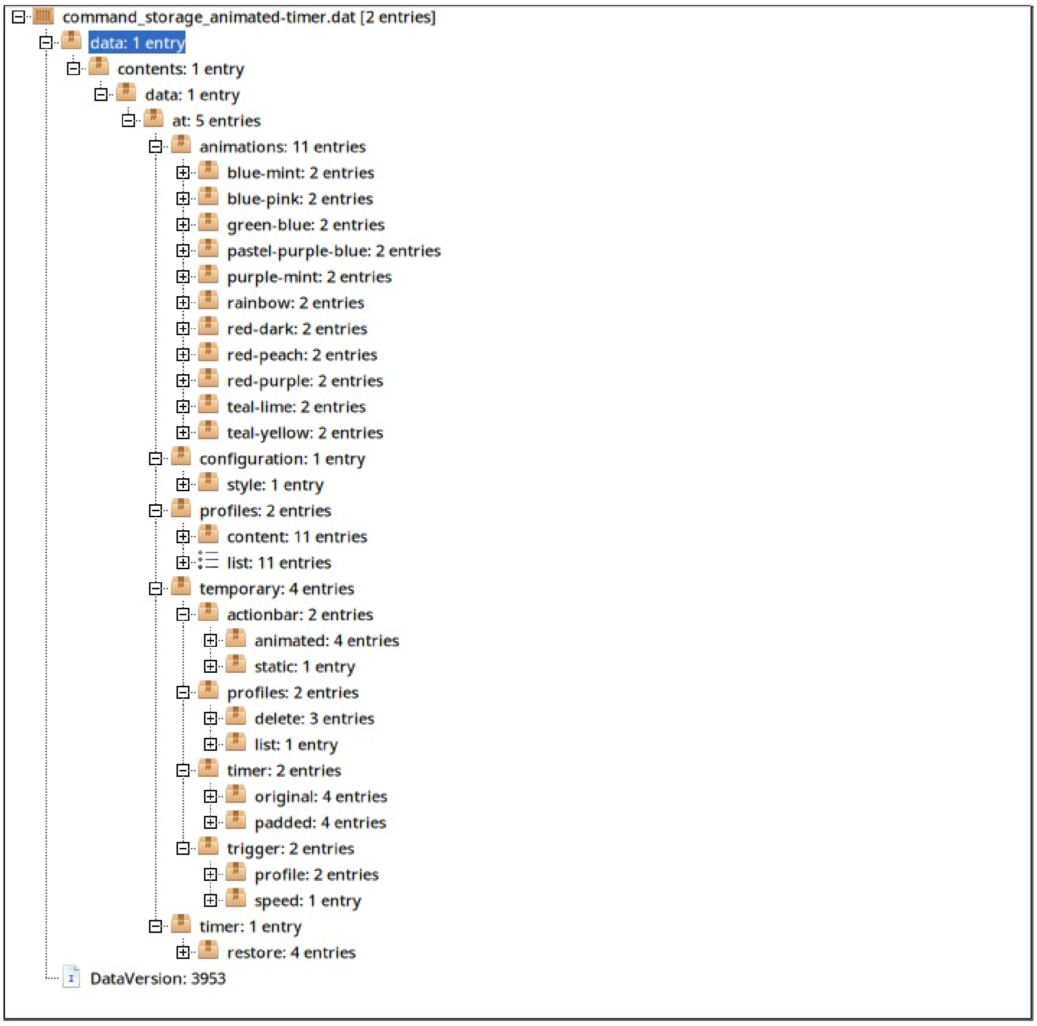
Trigger scoreboards (26 total):

continue, pause, reboot, reset, animation, bold1, bold2, color1, color2, direction, italic1, italic2, mode, render, show, speed, style, delete, list, load, save, view, days, hours, minutes, seconds

Details : NBT storage

AT uses the storage namespace animated-timer:data.

Path	Description / purpose
at.animations	Holds animations that are used by style type animated
at.configuration	Part of current / running configuration
at.profiles	Stores pre-built and player-made profiles
at.temporary	Temporary storage for anything that is not permanent data
at.timer	Holds restore values (last manual time set)



```
command_storage_animated-timer.dat [2 entries]
  data: 1 entry
    contents: 1 entry
      data: 1 entry
        at: 5 entries
          animations: 11 entries
            blue-mint: 2 entries
            blue-pink: 2 entries
            green-blue: 2 entries
            pastel-purple-blue: 2 entries
            purple-mint: 2 entries
            rainbow: 2 entries
            red-dark: 2 entries
            red-peach: 2 entries
            red-purple: 2 entries
            teal-lime: 2 entries
            teal-yellow: 2 entries
          configuration: 1 entry
            style: 1 entry
          profiles: 2 entries
            content: 11 entries
            list: 11 entries
          temporary: 4 entries
            actionbar: 2 entries
              animated: 4 entries
              static: 1 entry
            profiles: 2 entries
              delete: 3 entries
              list: 1 entry
            timer: 2 entries
              original: 4 entries
              padded: 4 entries
            trigger: 2 entries
              profile: 2 entries
              speed: 1 entry
          timer: 1 entry
            restore: 4 entries
  DataVersion: 3953
```

Modification : Custom animations

May need a bit more technical knowledge but is possible nevertheless. Your first step should be to look at the code of function at-abar:anmload - that is the place where pre-built animations are stored. Every animation follows this NBT storage pathing scheme:

at.animations.<animation-name>

An animation consists of two NBT tags: *count* & *colors*. The first one, *count*, is just an integer value containing the number of total colors (steps) per animation. The second, *colors*, is a list of strings containing the individual HEX codes that make up the animation.

The key is to find two colors that match and then generate a gradient between them. I've done this with an external tool: https://colordesigner.io/gradient-generator. If you have found a gradient that you are happy with, copy the colors and put them in the *colors* list.

AT does not loop around automatically. Going just from color A to B isn't sufficient - a full animation should look more like this: A > B > A. Best is to try around for yourself until you have your custom gradient.

If you are done, complete the *colors* list, write the final color count into *count*, come up with a great and unique animation name, save the file, reload and run `/function at-abar:anmload`.

Modification : Integration

Considerations to make when integrating AT in one of your projects:
- What kind of timer and style configuration do I need and how to do I implement it?
- Do I want something to happen if the timer pauses / continues / reaches zero?
- Do any of the AT trigger scoreboards conflict with mine?

Modification : Integration : Configuration

Part of the configuration work for AT can be done via functions in *at-conf*. Although this just works, it can be tedious or inefficient to run the same command twice or in a specific order to get the style you prefer.

Depending on what you are trying to do, it may make sense to look at functions in *at-dcfg* namespace. Any command there just does what it's named like and none of them give any output in form of tellraws.

Modification : Integration : Extraction

The namespace *at-dext:* contains four functions that are automatically invoked when a specific action happens:

Function	Invoked at
at-dext:continue	Timer start or continue via `/function at:continue` or trigger
at-dext:pause	Timer pause via `/function at:pause` or trigger
at-dext:reset	Timer reset via `/function at:reset` or trigger Timer reboot via `/function at:reboot` or trigger
at-dext:zero	Timer in mode DOWN reaches zero

Modification : Integration : Removing triggers

Depending on your use case, you might want to remove AT triggers completely. For this, two functions need to be edited <u>before</u> you load this data pack into your world:

at-main:load
- Remove any of the commands below "# Scoreboards / Trigger /…"
- Stop deleting lines at "# Load counter; run init…"

at-main:tick
- Comment out "function at-trig:dist" (add a # before it)