RoundMaster API for Roll20

RoundMaster is an API for the Roll20 RPG-DS. Its purpose is to extend the functionality of the Turn Tracker capability already built in to Roll20. It is one of several other similar APIs available on the platform that support the Turn Tracker and manage token and character statuses related to the passing of time: the USP of this one is the full richness of its functionality and the degree of user testing that has occurred over a 12 month period.

RoundMaster is based on the much older TrackerJacker API, and many thanks to Ken L. for creating TrackerJacker. However, roundMaster is a considerable fix and extension to TrackerJacker, suited to many different applications in many different RPG scenarios. RoundMaster is also the first release as part of the wider RPGMaster series of APIs for Roll20, composed of RoundMaster, CommandMaster, InitiativeMaster, AttackMaster, MagicMaster and MoneyMaster - other than RoundMaster (which is generic) these initially support only the AD&D2e RPG.

# Syntax of RoundMaster calls

The roundMaster API is called using !rounds, though it reveals its history in that it can also be called using !tj (the command for the TrackerJacker API roundMaster is based on).

!rounds --start  
!tj --start

Commands to be sent to the roundMaster API must be preceeded by two hyphens ‘--’ as above for the --start command. Parameters to these commands are separated by vertical bars ‘|’, for example:

!rounds --addtotracker name|tokenID|3|all|sleeping|sleepy

If optional parameters are not to be included, but subsequent parameters are needed, use two vertical bars together with nothing between them, e.g.

!rounds --addtotracker name|tokenID|3|all||sleepy

Commands can be stacked in the call, for example:

!rounds --start --addtotracker name|tokenID|3|all|sleeping|sleepy --sort

When specifying the commands in this document, parameters enclosed in square brackets [like this] are optional: the square brackets are not included when calling the command with an optional parameter, they are just for description purposes in this document. Parameters that can be one of a small number of options have those options listed, separated by forward slash ‘/’, meaning at least one of those listed must be provided (unless the parameter is also specified in [] as optional): again, the slash ‘/’ is not part of the command. Parameters in UPPERCASE are literal, and must be spelt as shown (though their case is actually irrelevant).

# Command Index

# Tracker commands

**--start  
--stop  
--pause  
--reset [number]  
--sort  
--clear**--clearonround [off/on]  
--clearonclose [off/on]  
--sortorder [nosort/atoz/ztoa/descending/ascending]  
--addToTracker name|tokenID/-1|priority|[qualifier]|[message]|[detail]  
--removefromtracker name|tokenID/-1|[retain]

# Token Status Marker commands

--addstatus status|duration|direction|[message]|[marker]  
--addtargetstatus tokenID|status|duration|direction|[message]|[marker]  
--edit  
--target CASTER|casterID|status|duration|direction|[message]|[marker]  
--target SINGLE/AREA|casterID|targetID|status|duration|direction|[message]|[marker]  
--aoe tokenID|[shape]|[units]|[range]|[length]|[width]|[image]|[confirmed]  
--clean  
--removestatus status(es)  
--deletestatus statuses  
--deltargetstatus tokenID|statuses  
--movestatus  
--s\_marker  
--disptokenconfig tokenID  
--listfav

# **Other commands**

–help  
–hsq from|[command]  
–handshake from|[command]  
--debug (ON/OFF)

# Tracker Command detail

!rounds --start

This command alternates between starting the automatic functions of the Turn Tracker, and pausing the Tracker. In its started state, the tracker will follow the current token at the top of the tracker with a highlight graphic, report the token’s turn to all players, and follow the options selected for ‘sortorder’ and ‘clearonround’. When paused, the Tracker will not highlight the top token, report turns or execute the options.

!rounds --stop

Stops the tracker and removes all statuses and status markers from tokens currently held by roundMaster. This also dumps the tables held in the campaign status object. It is useful if you want to start a fresh version of a campaign, or if everything goes wrong.

!rounds --pause

Pauses the Turn Tracker in its current state without deleting any information, and is the same as using the --start command again having already called it once. The Turn Tracker can still be moved on, cleared, sorted, and reset, but the highlight graphic is paused. It can be restarted using --start

!rounds --reset [number]

Sets the round in the Turn Order to the *number*, or to 1 if number is not provided.

!rounds --sort

Sorts the Turn Tracker entries according to the previously set sort order, with the default being ascending numeric priority.

!rounds --clear

Clears all entries in the Turn Tracker without stopping it.

!rounds --clearonround [off/on]

Sets the ‘clear on round’ option. If set, this option means that when the Tracker is running and reaches the end of the round, all entries in the Turn Tracker are automatically removed ready for players to do initiative for the next round. Otherwise, the Turn Tracker is not cleared automatically at any point. Any parameter other than ‘off’ turns clearonround on. Default on.

!rounds --clearonclose [off/on]

Sets the ‘clear on close’ option. If set, this option means that when the Tracker window is closed, the Turn Tracker is cleared. Any parameter other than ‘on’ turns clear on close off. Default off.

!rounds --sortorder [nosort/atoz/ztoa/descending/ascending]

This command sets the automatic sort order of the entries in the Turn Tracker. The Turn Tracker is automatically sorted at the beginning of each round as the Turn Tracker is moved on to the first entry, based on the order set by this option. *Descending* and *Ascending* are numeric sorts based on the Priority number of each entry. *AtoZ* and *ZtoA* are alphabetic sorts based on the name of each entry in the Turn Tracker. *Nosort* will mean that no sorting takes place, and the order remains the order in which the entries were made. The default order is Ascending.

!rounds --addToTracker name|tokenID/-1|priority|[qualifier]|[message]|[detail]

This command adds an entry to the TurnTracker. *tokenID* can either be the ID of a valid token, or -1 to create a custom entry. If a custom entry, *name* is used for the entry in the Turn Tracker with the provided priority, otherwise the token name is used for the entry with the provided priority. The *qualifier* can be one of first/last/smallest/largest/all.

* ***First*** keeps only the first entry made for that name (for custom entries) or token and removes any others, but leaves all entries for other tokens and names in the Tracker
* ***Last*** keeps only the latest entry for that token or name (i.e. the one now being set)
* ***Smallest*** keeps only the entry with the lowest numeric priority for that token or name
* ***Largest*** keeps only the entry with the highest numeric priority for that token or name
* ***All*** keeps all entries in the list and adds this one to those for that token or name, meaning that the Turn Tracker can have multiple entries for one or more tokens or names

The optional *message* will be displayed on the turn announcement for this turn when it is reached in the Turn Order. Generally, the message relates what the player said the character was doing for their initiative. The optional *detail* can be the detail of how the initiative priority was calculated or any other additional message you want to show to the Player only when the command is processed.

By using the *name*, *tokenID*/*-1* and *qualifier* parameters judiciously, group initiative, individual initiative, or any combination of other types can be created. When used with the InitiativeMaster API, Players get menus of actions they can take (based on their weapons, powers, memorised spells, magic items, thieving skills etc) which manage the calls to RoundMaster for the desired initiative type, and the DM gets menus to control all RoundMaster functions, and to set the type of initiative to undertake..

!rounds --removefromtracker name|tokenID/-1|[retain]

This command removes entries from the Turn Tracker for the specified *tokenID* or *name*. However, if the optional *retain* number is given, it will retain this number of entries for the specified token or name, and only remove any beyond this number. The earliest entries made are kept when the *retain* parameter is set.

!rounds --viewer on/off/all/tokenID

This command controls the viewer mode setting for the Player who calls it. Rather than showing what that Player’s characters can see when Dynamic Lighting is turned on, viewer mode shows that Player what each player-character (even if not theirs) can see as their token reaches the top of the Turn Tracker and it is their turn. Quite often, this can be a Player ID set up just to be a viewer e.g. for a DM view of what players can see, or for a touchscreen playing surface. The current player-character is defined as the token representing a character sheet controlled by any Player at the top of the Turn Tracker. As each new token comes to the top of the Turn Tracker, if it is a player-character the display changes to only what it can see. If it is a token representing an NPC, or when the Turn Order reaches the next round and clears, the map for the Player reverts to showing what all player-characters can see (but not what NPCs can see).

The *on* option turns on viewer mode for the Player, and *off* turns it off. The *all* option immediately turns on vision for all player-characters, and passing a *tokenID* as a parameter shows vision for that token (even if it represents an NPC). Options *off*, *all* and *tokenID* can be used by any Player or the DM to affect the viewer Player’s screen.

# Token Status Marker commands

!rounds --addstatus status|duration|direction|[message]|[marker]

Adds a status and a marker for that status to the currently selected token(s). The status has the name given in the *status* parameter and will be given the *duration* specified, which will be changed by *direction* each round. Thus setting a *duration* of 8 and *direction* of -1 will decrement the duration by 1 each round. If the duration gets to 0 the status and token marker will be removed automatically. *direction* can be any number - including a positive one meaning duration will increase. Each Turn Announcement for the turn of a token with one or more statuses will display the status, the duration and direction, and the message, if specified. The specified marker (from the default token marker set) will be applied to the token - if it is not specified the option will be given to pick one from a menu in the chat window (which can be declined).

For player-characters, when the duration reaches 9 or less the duration will be counted-down by a number appearing on the marker. For NPCs this number does not appear (so that Players don’t see the remaining duration for statuses on NPCs), but the remaining duration does appear for DM only on the status message below the Turn Announcement on the NPCs turn.

If a Player other than the DM uses this command, the DM will be asked to confirm the setting of the status and marker. This allows the DM to make any decisions on effectiveness.

If an Effects database of effect macros exists within the campaign (a character sheet with the name Effects-DB, with Ability Macros named the same as the status *status* parameter set for markers) the Effects database will be searched in three ways: when a status marker is set, any Ability Macro with the name *status*-start (where *status* is the status name specified in the command) is run. Each round when it is the turn of a token with the status marker set, the Ability Macro with the name *status*-turn is run. And when the status ends (duration reaches 0) or the status is removed using --removestatus, or the token has the Dead marker set or is deleted, an Ability Macro with the name *status*-end is run. See the Effects database documentation for full information on effect macros and the options and parameters that can be used in them.

!rounds --addtargetstatus tokenID|status|duration|direction|[message]|[marker]

This command is identical to addstatus, except for the addition of a tokenID. Instead of using a selected token or tokens to apply the status to, this applies the status to the specified token.

!rounds --edit

This command brings up a menu in the chat window showing the current status(es) set on the selected token(s), with the ability to remove or edit them. Against each named status, a spanner icon opens another menu to edit the selected status name, duration, direction, message and marker on all the selected token(s), and also allows this status to be set as a favourite. A bin icon will remove the status from all the selected token(s), and run any *status*-end macros, if any.

!rounds --target CASTER|casterID|status|duration|direction|[message]|[marker]  
!rounds --target SINGLE/AREA|casterID|targetID|status|duration|direction|[message]|[marker]

This command targets a status on a token or a series of tokens. If a version using *CASTER* is called, it acts identically to the addtargetstatus command, using the casterID as the target token. If the *SINGLE* version is called, the targetID is used. If the *AREA* version is used, after applying the status to the targetID token, the system asks in the chat window if the status is to be applied to another target and, if confirmed, asks for the next target to be selected, repeating this process after each targeting and application. In each case, it applies the status, effect macro and marker to the specified token(s) in the same way as addtargetstatus.

!rounds --aoe tokenID|[shape]|[units]|[range]|[length]|[width]|[image]|[confirmed]  
  
shape = [Bolt/Circle/Cone/Ellipse/Rectangle/Square/Wall]  
units = [Squares/Feet/Yards/Units]  
image = [Acid/Cold/Dark/Fire/Light/Lightning/Magic/Red/Yellow/Blue/Green/Magenta/Cyan/ White/Black]  
confirmed = [true / false]  
range, length & width are numbers specified in whatever unit was specified as [units]

This command displays an Area of Effect for an action that has or is to occur, such as a spell. This quite often can be used before the **--target area** command to identify targets. The system will present lists of options for each parameter that is not specified for the Player to select. On executing this command, if the *range* is not zero the Player will be given a crosshair to position the effect, and if the range is zero the effect will be centred on the Token (or at its “finger-tips” for directional effects like cones). The range of the effect will be centred on the TokenID specified and will be displayed as a coloured circle - the crosshair should be positioned within this area (the system does not check). The Crosshair (or if *range* is zero, the Token) can be turned to affect the direction of the effect. The effect ‘direction’ will be the direction the token/crosshair is facing. If *Confirmed* is false or omitted, the Player will be asked to confirm the positioning of the token/crosshair with a button in the chat window (setting it to *true* will apply the effect immediately - good for range zero circular effects (i.e. don’t need placing or direction setting). The effect can have one of the shapes listed:

* **Bolt** is a long rectangle extending away from the crosshair/token for *length*, and *width* wide.
* **Circle** is a circle centred on the crosshair/token of diameter *length*.
* **Cone** is a cone starting at the crosshair/token of *length*, with an end *width*.
* **Ellipse** is an ellipse of *length* extending away, and *width* wide.
* **Rectangle** is a rectangle of *length* extending away, and *width* wide.
* **Square** is a square of sides *length* parallel with the direction the crosshair/token.
* **Wall** is a rectangle perpendicular to the crosshair or token, i.e. *width* away and *length* wide.

For the *units*, *Feet* & *Yards* are obvious and are scaled to the map. *Squares* are map squares (whatever scale they are set to), and *Units* are the map scale units and are not scaled.

*Images* are set with transparency and sent to the back of the Object layer. Red/ Yellow/ Blue/ Green/ Magenta/ Cyan/ White/ Black colour the effect area the specified colour, and Acid/ Cold/ Dark/ Fire/ Light/ Lightning/ Magic use textured fills.

!rounds --clean

Drops all the status markers on the selected token(s), without removing the status(es) from the campaign status object, meaning live statuses will be rebuilt at the end of the round or the next trigger event. This deals with situations where token markers have become corrupted for some reason, and should not be needed very often.

!rounds --removestatus status(es)

Removes the status, or comma-delimited list of statuses, and their status marker(s) from the selected token(s), and runs any associated *status*-end Ability Macros in any existing Effects database in the campaign. See addstatus command and the Effect database documentation for details on effect macros. Statuses can be ‘all’ which will remove all statuses from the selected token(s).

!rounds --deletestatus statuses

Removes the status, or comma-delimited list of statuses, and status marker(s) from the selected token(s), but does not run any associated *status*-end Ability Macros in any existing Effects database in the campaign. Statuses can be ‘all’ which will delete all statuses from the selected token(s).

!rounds --deltargetstatus tokenID|statuses

Works the same as deletestatus command, except only on the specified tokenID rather than selected tokens.

!rounds --movestatus

For each of the selected tokens in turn, searches for tokens in the whole campaign with the same name and representing the same character sheet, and moves all existing statuses and markers from all the found tokens to the selected token (removing any duplicates). This supports Players moving from one Roll20 map to another and, indeed, roundMaster detects page changes and automatically runs this command for all tokens on the new page controlled by the Players who have moved to the new page.

!rounds --s\_marker

Shows a display of all markers available in the API to the DM, and also lists which are currently in use.

!rounds --disptokenconfig tokenID

Shows the statuses on the specified token to the DM using the same display format as used in the Turn Announcement.

!rounds --listfav

Shows statuses to the DM that have been defined as favourites (see the edit command), and provides buttons to allow the DM to apply one or more favourite statuses to the selected token(s), and to edit the favourite statuses or remove them as favourites.

# Other commands

!rounds –help

Displays a listing of RoundMaster commands and their syntax.

!rounds –hsq from|[command]  
!rounds –handshake from|[command]

Either form performs a handshake with another API, whose call (without the ‘!’) is specified as *from* in the command parameters. The command calls the *from* API command responding with its own command to confirm that RoundMaster is loaded and running: e.g.

Received: *!rounds –hsq magic*  
Response: *!magic –hsr rounds*

Optionally, a command query can be made to see if the command is supported by RoundMaster if the *command* string parameter is added, where *command* is the RoundMaster command (the ‘--’ text without the ‘--‘). This will respond with a *true/false* response: e.g.

Received: *!rounds –hsq init|addtotraker*  
Response: *!init –hsr rounds|addtotracker|true*

!rounds --debug (ON/OFF)

Takes one mandatory argument which should be ON or OFF.

The command turns on a verbose diagnostic mode for the API which will trace what commands are being processed, including internal commands, what attributes are being set and changed, and more detail about any errors that are occurring. The command can be used by the DM or any Player – so the DM or a technical advisor can play as a Player and see the debugging messages.

# How to use RoundMaster

## Who uses RoundMaster calls?

The vast majority of RoundMaster calls are designed for the DM/GM to use, or to be called from RPGMaster APIs and database macros, rather than being called by the Player directly. RoundMaster should be hidden from the Players in most circumstances. It is *highly recommended* that RoundMaster is used with the other RPGMaster APIs, but especially **InitiativeMaster API** which uses RoundMaster to create and manage entries in the Roll20 Turn Order Tracker.

## Managing the Turn Order Tracker

If the **InitiativeMaster API** is used, it ***must*** be accompanied by RoundMaster – it will not work otherwise. InitiativeMaster provides many menu-driven and data-driven means of controlling RoundMaster, making it far easier for the DM to run their campaign. The InitiativeMaster **--maint** command supports the necessary calls to RoundMaster for control of the Turn Order Tracker, and its **--menu** command uses the data on the Character Sheet to create Turn Order initiative entries with the correct speeds and adjustments. See the InitiativeMaster API Handout for more information.

## Adding and managing Token Statuses

The Token status management functions allow the application and management of status markers with durations in rounds passed to tokens. The easiest way to use status markers is to use the **MagicMaster API** which runs spell and magic item macros the Player can initiate, which in turn ally the right status markers and statuses with the appropriate durations to the relevant tokens. See the MagicMaster API handout for more information.

## Status Effects

RoundMaster comes with a number of status ‘effects’: Roll20 Ability Macros that are automatically run when certain matching statuses are applied to, exist on, and/or removed from a token. These macros can use commands (typically using APIs like **ChatSetAttr API** and/or **Tokenmod API** from the Roll20 API Script Library) to temporarily or permanently alter the characteristics of the Token or the represented Character Sheet, thus impacting the state of play.

If used with the **MagicMaster API**, its pre-configured databases of spell and magic item macros work well with the Effect macros provided in the Effects Database provided with this API.

For full information on Status Effects, how to use them, and how to add more of your own, see the separate Effects Database handout.

## Token Death and Removal

If a token is marked as Dead by using the Dead status marker (either via the Roll20 token emoticon menu or any other means), the system will automatically end all statuses, remove all status markers and run all *status*-end effect macros (if any) for the token.

If a token is deleted on a map and not previously marked as Dead, the API will search for any other token in the Campaign (with a preference to one on the current page) with the same name and representing the same character, and if found the API will transfer all statuses and status markers to the first token found (even if not on the current page). If no such token is found, all statuses and status markers are removed from the token being deleted, and any corresponding *status*-end effect macros are run.

## Page Change and Adding Tokens

If a Player, or all Players, are moved to another Roll20 page in the campaign (i.e. a different map), the API will automatically migrate all current statuses and status markers from the previous page (and all other pages in the Campaign, to support where tokens have come from different pages) to any token on the new page with the same Token Name and representing the same Character. These statuses and their effects will then continue to apply on the new page for their set durations.

If a token is added to the current map the Players are on, either by dragging a character onto the map or by dragging on a picture and editing its properties to give it a name and optionally a representation, the API will again search for tokens with the same name and representing the same character, and move statuses and markers to the new token.

Of course, either before or after each of these situations, the --edit command can be used to change or remove statuses from any token(s).

# Effect Database for RoundMaster API

Effect-DB is a database character sheet created, used and updated by the **RoundMaster API** (see separate documentation). The database holds macros as Ability Macros that are run when certain matching statuses are placed on or removed from tokens (see Roll20 Help Centre for information on Ability Macros and Character Sheet maintenance). The macros have certain defined parameters dynamically replaced when run by RoundMaster, providing the token & character IDs and names, and values such as AC, HP and Thac0, available for manipulation.

Effect macros are primarily intended to act on the Token and its variables, but can also act on the represented Character Sheet. A single Character Sheet can have multiple Tokens representing it, and each of these are able to do individual actions using the data on the Character Sheet jointly represented. However, if such multi-token Characters / NPCs / creatures are likely to encounter effects that will affect the Character Sheet they must be split with each Token representing a separate Character Sheet, or else the one effect will affect all tokens associated with the Character Sheet, whether they were targeted or not! In fact, **it is recommended that tokens and character sheets are 1-to-1** to keep things simple.

The recommended Token Bar assignments for all APIs in the Master Series are:

Bar1 (Green Circle): Armour Class (AC field) – only current value  
 Bar2 (Blue Circle): Base Thac0 (thac0-base field) before adjustments – only current value  
 Bar3 (Red Circle): Hit Points (HP field) – current & max

It is recommended to use these assignments, and they are the bar assignments set by the **CommandMaster API** if its facilities are used to set up the tokens. All tokens must be set the same way, whatever way you eventually choose.

These assignments can be changed in each API, by changing the *fields* object near the top of the API script. See individual API documentation for details of how to do this.

## Macro Parameter Fields

Dynamic parameters are identified in the macros by bracketing them with two carets: **^^parameter^^**. The standard Roll20 syntax of @{selected|…} is not available, as at the time the macros run the targeted token may not be selected, and @{character\_name|…} will not enable the token to be affected (especially where the Character Sheet is represented by more than one token). The ^^…^^ parameters always relate to the token on which a status has been set, and the Character Sheet it represents. Currently available parameters are:

**^^tid^^** TokenID  
**^^tname^^** Token\_name  
**^^cid^^** CharacterID  
**^^cname^^** Character\_name  
  
**^^ac^^** Armour Class value (order looked for: a token bar, Character Sheet AC field, MonsterAC)  
**^^ac\_max^^** Maximum value of AC, wherever it is found  
**^^token\_ac^^** The token field name for AC value field, if set as a token bar  
**^^token\_ac\_max^^** The token field name for AC max field, if set as a token bar  
  
**^^thac0^^** Thac0 value (order looking: a token bar, Character Sheet Thac0\_base field, MonsterThac0)  
**^^thac0\_max^^** Maximum value of Thac0, wherever it is found  
**^^token\_thac0^^** The token field name for Thac0 value field, if set as a token bar  
**^^token\_thac0\_max^^** The token field name for Thac0 max field, if set as a token bar  
  
**^^hp^^** HP value (order looked for: a token bar, Character Sheet HP field)  
**^^hp\_max^^** Maximum value of HP, wherever it is found  
**^^token\_hp^^** The token field name for HP value field, if set as a token bar  
**^^token\_hp\_max^^** The token field name for HP max field, if set as a token bar  
  
**^^bar1\_current^^** Value of the token Bar1\_value field  
**^^bar2\_current^^** Value of the token Bar2\_value field  
**^^bar3\_current^^** Value of the token Bar3\_value field

This allows most data on both the token and the character sheet to be accessed. For example **@{^^cname^^|strength}** will return the strength value from the represented character sheet. Of course all loaded RPGMaster series API commands are available.

Two other APIs from the Roll20 Script Library are extremely useful for these macros, and indeed are used by many of the provided APIs: **ChatSetAttr API** from joesinghaus allows easy and flexible setting of Character Sheet attributes. **Tokenmod API** from The Aaron supports easy setting and modifying of Token attributes. Combined with the dynamic parameters above, these make for exceptionally powerful real-time effects in game-play.

## Effect Macro qualifiers

Each effect macro runs when a particular status event occurs. Here is the complete list of effect macro status name qualifiers that can be used. Each of these is appended to the status whenever the status experiences the relevant event, and an effect macro with that name searched for and run if found:

statusname**-start** The status is created on a token  
statusname**-turn** Each round the status has a duration that is not zero  
statusname**-end** The status duration reaches zero

These effect across are triggered for weapons when certain events take place:

weaponname**-inhand** A weapon is taken in-hand (triggered by AttackMaster API --weapon command)  
weaponname**-dancing** A weapon starts dancing (triggered by AttackMaster API --dance command)  
weaponname-**sheathed** A weapon is sheathed (out of hand - triggered by AttackMaster --weapon cmd)

## Examples of Effect Macros

Here is an example of an effect macro that runs when a Faerie fire (twilight form) status is placed on a token. The following --target command might be run to set this status, with the caster token selected:

!rounds --target area|@{selected|token\_id}|&{target|Select first target|token\_id}|Faerie-Fire-twilight|[[4\*@{selected|Casting-Level}]]|-1|Outlined in dim Faerie Fire, 1 penalty to AC|aura

This will result in the following effect macro being run when the first token is targeted:

**Faerie-fire-twilight-start**

!token-mod --ignore-selected --ids ^^tid^^ --set ^^token\_ac^^|+1  
^^tname^^ is surrounded by Faerie Fire, and becomes easier to hit

This uses the Tokenmod API to increase the AC number of the targeted token by 1 (making it 1 wose), and then display a message to all Players stating the name of the targeted token, and the effect on it. This will be run for each token targeted, and will be individual to each. **Note:** the tokens are not ‘selected’ in Roll20 terms, and so *@{selected|…}* will not work

When the Faerie Fire status counts down to zero, the following effect macro will be run on each of the tokens it was applied to:

**Faerie-fire-twilight-end**

!token-mod --ignore-selected --ids ^^tid^^ --set ^^token\_ac^^|-1  
^^tname^^ has lost that glow and is now harder to aim at

Again, the Tokenmod API is used to decrease the token AC and a message issued confirming what has happened. If messages should only be sent to the Player(s) controlling the character represented by the token, use */w “^^cname^^”* before the message. If the message is only for the gm, use */w gm*.

A more complex example is a Quarterstaff of Dancing, that uses the complete suite of possible effect macros and certain aspects of the **AttackMaster API** functionality triggered by Weapon table field settings. The first macro is triggered by AttackMaster API when a Character takes a Quarterstaff-of-Dancing in hand to use as a weapon:

**Quarterstaff-of-Dancing-inhand**

!rounds --addtargetstatus ^^tid^^|Quarterstaff-of-Dancing|4|-1|Quarterstaff not yet dancing so keep using it|stopwatch

This command sets a status marker on the Token of the Character taking the Quarterstaff in hand, and sets a countdown of 4 rounds, running the next effect macro in each of those rounds:

**Quarterstaff-of-Dancing-turn**

!attk --quiet-modweap ^^tid^^|quarterstaff-of-dancing|melee|+:+1 --quiet-modweap ^^tid^^|quarterstaff-of-dancing|dmg|+:+1  
/w “^^cname^^” Updating the quarterstaff +1 to attk & dmg

This command then runs each round as the Quarterstaff-of-Dancing status counts down, and uses the *!attk --quiet-modweap* command to gradually increment the magical to-hit and dmg plus, round by round. Once the countdown reaches zero, the next effect macro is run:

**Quarterstaff-of-Dancing-end**

!attk --dance ^^tid^^|Quarterstaff-of-Dancing

This calls an AttackMaster API command to start the weapon dancing, resets the weapon to its specs that it starts dancing with, and calls the next effect macro:

**Quarterstaff-of-Dancing-dancing**

!rounds --addtargetstatus ^^tid^^|Dancing-Quarterstaff|4|-1|The Quarterstaff is Dancing by itself. Use this time wisely!|all-for-one  
!attk --quiet-modweap ^^tid^^|quarterstaff-of-dancing|melee|sb:0 --quiet-modweap ^^tid^^|quarterstaff-of-dancing|dmg|sb:0

This places a new status marker on the token representing the Character with the dancing weapon (note the new status name *Dancing-Quarterstaff*), and resets the Strength Bonus flags for the weapon - a dancing weapon can’t have the Strength Bonus of the wielder. As each round now passes, the following different status effect macro is run:

**Dancing-Quarterstaff-turn**

!attk --quiet-modweap ^^tid^^|quarterstaff-of-dancing|melee|+:+1 --quiet-modweap ^^tid^^|quarterstaff-of-dancing|dmg|+:+1

As per the previous *-turn* effect macro, this increments the magical plusses on To-Hit and Dmg, round by round. It has to have a different name, as the *-end* effect macro does different actions:

**Dancing-Quarterstaff-end**

!attk --dance ^^tid^^|Quarterstaff-of-Dancing|stop

This uses the AttackMaster API command to stop the Quarterstaff from dancing. As can be seen from the above, quite complex sequences of effect macros can be created.