Programming ProconRulz

Bambam

V43d.1 (17th Sept 2012)

Table of Contents

About ProconRulz

ProconRulz is an admin utility to help manage game servers, written for Battlefield Bad Company 2 and subsequently updated for Battlefield 3.

ProconRulz is implemented as a plugin that extends the functionality of the Procon rcon admin tool.

The basic real-time data spewed out by a game server such as Battlefield includes messages such as “pebbles killed bambam with weapon M416 (Headshot)” and “Player bambam spawned as Recon with weapons M98S, M9 and specialization Squad Sprint”. The messages are formatted in a way that Procon and ProconRulz can sensibly interpret the data (although the messages are human-readable too).

Given this data, ProconRulz provides the most flexible support possible for what an admin might want to do based on these events. The simplest example, e.g. to make the use of the RPG-7 suicidal, would ‘Kill’ any player that uses it:

On Kill;Weapon RPG-7;Kill

What ProconRulz will do is listen for the “player kill” event, note the Weapon the player has killed with (i.e. AUG, M416, RPG-7 whatever), if the player has used the RPG-7 then ProconRulz will Kill the player. So this results in the behavior most players are familiar with: you get auto-slayed if you use a banned weapon…

If you think you’ve got the concept and want to get started writing your rulz, then you can see Appendix 1 and 2 for the actual weapon keys etc. available to use.

Why ProconRulz is different from a typical Procon plugin

ProconRulz is best thought of as a meta-plugin, i.e. it is a plugin that allows a broad set of users to create new plugins. If you want a “no hand-grenades plugin”, fine, you can do that with a single rule. Procon users can install a purpose-built plugin to provide a “TeamKill Limiter” for example, or can use ProconRulz with a couple of appropriate text rulz that provide that functionality. Other users’ rulz can be simply cut-and-paste into your rule set so you can easily add for example a “Quake KillStreak Announcer”. If you grasp the basics of the rule format, you can easily adjust other users’ rulz to display different messages, or tweak limit thresholds so, for example, you punish after 3 teamkills not 5.

A simple one-line rule is all that it takes to put a hard limit on something the server admin wants to control. But ProconRulz is treating the list of rulz as a *script* and if you understand the control flow you can program fairly complex behavior to achieve exactly the results you want.

For example you can give a warning message on the first couple of kills, maybe auto-slay the player for a few kills after that, and if they persist in using a limited weapon you can kick them of the server or ban them.

You can write rules that punish a player for using a weapon *not* in a desired set, e.g. with a single rule you can have a pistols-only server, or knives-only, or snipers and pistols.

You can write messages in-game based on any set of conditions you detect occurring.

ProconRulz has a generic ‘Exec’ action that supports *any* server command (e.g. to update server variables such as max permitted players, or vehicles allowed, or restart round), in additional to simple one-word actions such as Kill, Kick, Ban, Say.

ProconRulz can act upon the event of a player entering ‘say’ text into the game server, so players can directly trigger ProconRulz actions. The means you can write your own in-game admin commands.

For these reasons ProconRulz is most fairly described as a *meta-plugin*. With the use of appropriate rulz, much functionality can be supported that otherwise would need a complete new plugin to be written.

The basics of creating your own rulz

Firstly you want to understand the *data* your rulz are actually working with. Procon will divide most game data into *Kit, Weapon, Damage* and *Specializations*. Appendix 2 lists the actual real values available for these in Battlefield 3, and the values for whatever game you are using will be dynamically displayed in the ProconRulz plugin 'Details' tab within the Procon client.

Weapon is what it says, i.e. the killing tool of choice. Most games will provide Weapon data when they are reporting a player spawning or a kill occurring in the game.

Damage is the effect a weapon has done. Many weapons do the same kind of Damage, e.g. all pistols do damage "Handgun". Other common damage types are "SniperRifle" or "ProjectileExplosive". This is convenient if you want a rule that affects *all* types of sniper rifle – in this case you design the rule to check for the *Damage*, not the actual *Weapon*.

Kit (not BF3) is generally the 'player type' the gamer has selected in their options, e.g. Assault, Medic, Engineer. Generally when the player spawns, the Kit data will be made available to the admin plugin (but not currently in BF3, which is a significant limitation for admin rulz).

Specializations (not BF3) are the general term for the fancy stuff you can tweak your player or weapons loadout with typically just before you load your player into the game. These include things such as a "sprint" ability or "sabot shotgun rounds". As with *Kit*, this information is delivered fine in most games but *not* BF3.

You will be able to most effectively use ProconRulz if you understand the concepts of *events*, *triggers*, *conditions* and *actions*.

Events are continually being sent from the game server to the Procon admin tool. Different games may generate different events at different times, but there's a broad consistency. When a player spawns into a game the game server will send a "player spawned" packet of information to the admin tool (which we think of as the "player spawned *event*"). Game servers vary in the data they send in this packet of data: BFBC2 sent the name of the player, plus information about the options selected by the player i.e. *Kit, Weapons, Specializations.* BF3 only sends the player *name* at spawn, plus *Weapon* information in a kill. Other games (particularly BFBC2 provide a complete set of data for each event).

Triggers are used in ProconRulz to associate a rule with a particular event in the game. By *far* the most common events are "spawn" and "kill". So if a rule begins "On Kill;…", that rule will *only* be applied to the data that arrives from kill *Events*.

Conditions are used in rulz to *test* the data provided in the *Event*. In every game the Kill *Event* does include the *Weapon* (and *Damage*) information (even BF3…), so if you want a rule to apply when a player kills with a sniper rifle you can use "On Kill;Damage SniperRifle;…"

Actions are commands that can be sent *to* the game server to make something happen in the game. The most common actions you would want to use include killing a player (this is the "Kill" action, not to be confused with the "On Kill" trigger), kicking a player, banning a player, or having a 'chat' message appear in the game (this is the "Say" action).So if you want to nerf the use of sniper rifles on your server, you can do this with the single rule:

On Kill; Damage SniperRifle; Say No snipers on this server; Kill

Hopefully this rule is now crystal clear… when a Kill *Event* occurs, ProconRulz will check to see if the *Damage* was caused by a sniper rifle, if so it will put a chat message into the game reminding players no sniper rifles are allowed, and Kill the player that triggered the rule (i.e. did the kill in this case).

As a final introductory point, you can *reverse* the meaning of most conditions with a keyword "Not", e.g. "Not Damage SniperRifle" will check that the damage caused was *not* the type of damage done by a sniper rifle. So having a snipers-only server is equally simple:

On Kill; Not Damage SniperRifle; Say Snipers-only on this server; Kill

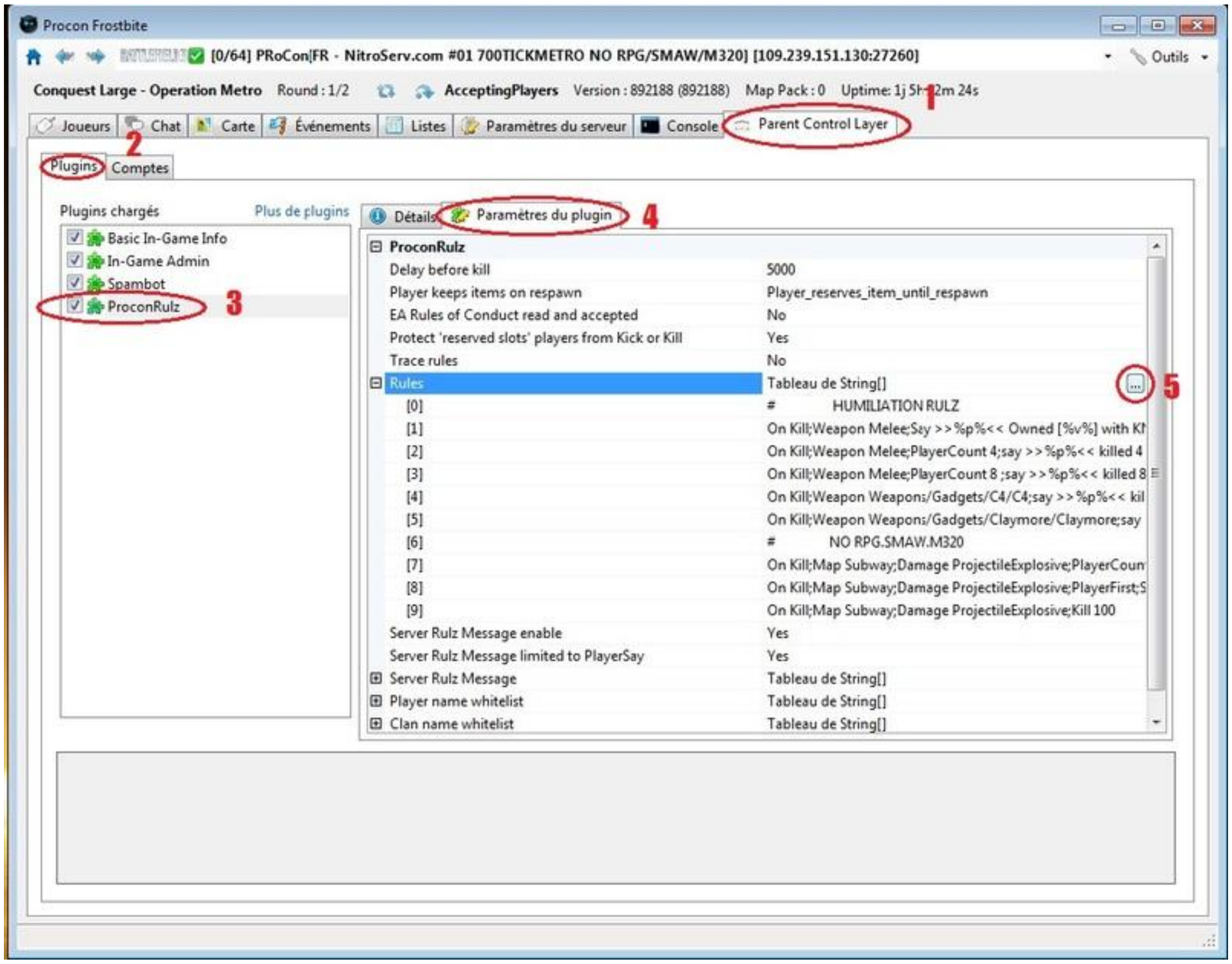
Entering your rulz into ProconRulz

Rulz are entered into ProconRulz via the Plugins / Settings tab. The screenshot below shows the settings for the ProconRulz plugin running on a Procon Layer Server (accessed via the "Parent Control Layer" tab). If you are running ProconRulz on your desktop (which works perfectly well) then you will find the plugin under a top-level 'Plugins' tab not shown in the screenshot, but the window is otherwise the same.

To enter your rulz, just click 1..5 as highlighted below. When ProconRulz is installed, you will have quite a few default rulz in the "Rules" setting – none of these are essential, they're just examples. If you want to test your setting, try adding a rule at the top of the Rules setting:

On Say;Text test;Say Hello World

Now, within the game, chat the word "test", and you will see ProconRulz echo back the message "Hello World". Congratulations, you're a ProconRulz programmer.



ProconRulz	
Delay before kill	5000
Player keeps items on respawn	Player_reserves_item_until_respawn
EA Rules of Conduct read and accepted	No
Protect 'reserved slots' players from Kick or Kill	Yes
Trace rules	No
Rules	Tableau de String[]
[0]	# HUMILIATION RULZ
[1]	On Kill;Weapon Melee;Say >>%p%<< Owned [%v%] with KN
[2]	On Kill;Weapon Melee;PlayerCount 4;say >>%p%<< killed 4
[3]	On Kill;Weapon Melee;PlayerCount 8 ;say >>%p%<< killed 8
[4]	On Kill;Weapon Weapons/Gadgets/C4/C4;say >>%p%<< kil
[5]	On Kill;Weapon Weapons/Gadgets/Claymore/Claymore;say
[6]	# NO RPG.SMAW.M320
[7]	On Kill;Map Subway;Damage ProjectileExplosive;PlayerCoun
[8]	On Kill;Map Subway;Damage ProjectileExplosive;PlayerFirst;S
[9]	On Kill;Map Subway;Damage ProjectileExplosive;Kill 100
Server Rulz Message enable	Yes
Server Rulz Message limited to PlayerSay	Yes
Server Rulz Message	Tableau de String[]
Player name whitelist	Tableau de String[]
Clan name whitelist	Tableau de String[]

A typical ProconRulz rule

This is a fairly typical rule that will display a message such as "bambam stabbed slartibartfast" each time a knife kill occurs:

On Kill; Weapon Weapons/weapon/knife; Say %p% stabbed %v%

The underlying format of ProconRulz rules is always the same: [trigger][conditions…][actions…]. Taking this rule as an example, the component parts are:

On Kill: this is the *trigger*, i.e. the type of event that will trigger this rule to be tested. A full list of the permitted triggers is given in an appendix.

Weapon Weapons/weapon/knife: this is the *condition* that will be tested after the trigger has been confirmed, i.e. on a kill event. The weapon used in the kill event will be compared with "Weapons/weapon/knife" and if there's a match then the condition will *succeed* and processing of this rule will continue. The list of the weapons available in BF3 is given in an appendix.

Say %p% stabbed %v%: this is the *action*, i.e. the rcon admin command that you want to be executed for this trigger when all the conditions succeed, in this case the 'Say' action. The format of ProconRulz actions is that the first word defines the action (i.e. "Say") while the rest of the clause gives information the action might use, typically a message in this case "%p% stabbed %v%". The full list of available actions is given in an appendix.

%p%, %v%: these are *substitution variables*. ProconRulz will replace %<…>% with an associated value calculated in real-time from the stream of data coming from the game server. The most common substitution variable is %p% which is replaced with the name of the player that triggered the current rule, e.g. on a Kill, it is the name of the killer. A full list of the substitution variables is given in an appendix.

How that rule might be refined

The rule above has a single *condition*, i.e. Weapon Weapons/weapon/knife. This condition can be extended to allow multiple weapon keys, and additional conditions can be added.

In programming ProconRulz, it becomes necessary to learn the behaviour of the *game* you are actually programming the rulz for… as a particular example, BF3 actually has *multiple* knife weapons (as of July 2012):

1. Weapons/weapon/knife
2. Melee
3. Knife_RazorBlade

When you stab a player, BF3 may use either of the first two, seemingly at random (actually the choice depends on the animation…). If you are a Premium player, then weapon #3 might be used. ProconRulz allows you to specify *alternatives* in most rule conditions by separating the values with commas, i.e. Weapon Weapons/weapon/knife,Melee,Knife_RazorBlade; A fairly typical ProconRulz noob mistake is to confuse the "Weapon" of the condition name with the first part of the #1 weapon key, which coincidentally happens to be the string "Weapons". *All* ProconRulz conditions are of the format <condition name> key#1,key#2,…

But your rule above can be extended to capture *both* weapons by using the 'alternate key' capability of ProconRulz conditions, i.e. you can specify multiple *alternate* keys in a Weapon condition, separated with *comma*:

```
On Kill; Weapon Weapons/weapon/knife,Melee,Knife_RazorBlade; Say %p% stabbed %v%
```

Please note that spaces are not permitted in weapon keys or either side of the comma. So the rule above will succeed for a kill *either* with Weapon Melee, *or* with Weapon Weapons/weapon/knife *or* with Weapon Knife_RazorBlade. Note that you can achieve a similar thing by using *multiple rulz*:

```
On Kill; Weapon Weapons/weapon/knife; Say %p% stabbed %v%

On Kill; Weapon Melee; Say %p% stabbed %v%

On Kill; Weapon Knife_RazorBlade; Say %p% stabbed %v%
```

How does ProconRulz step through your rulz?

The example above describes the processing of a single rule, i.e. left-to-right, starting with the trigger, then testing the conditions and finally applying the actions. It is possible for you to get value out of ProconRulz with a single rule (perhaps for each trigger), and if that is all you have then you don't have to worry about how ProconRulz processes multiple rulz.

However, for more complex auto-administration, it is likely you will want to have ProconRulz test for multiple combinations of conditions and apply your chosen actions in a variety of circumstances (e.g. you might want your rulz to auto-kill a player after they have committed 3 teamkills, but you might want to have them auto-kicked after 5 teamkills). In this case it is essential you understand how ProconRulz steps through multiple rulz. We'll use the following example rule set (the rule numbers are not part of the rulz, just here to help the explanation):

1. On Spawn; Teamsize 4; Say %p% spawned on team %pt% (teams are small)

2. On Kill; Weapon Weapons/weapon/knife; Say Knife kill by %p%

3. On Spawn; Say %p% spawned

4. On Kill; Map Subway; Damage ProjectileExplosive; Say %p% no rockets on Metro; Kill

5. On Kill; Damage ProjectileExplosive; Say %p% rocket killed %v%

Let's assume the following event has occurred: [bambam has killed slartibartfast with the SMAW anti-tank RPG] (and we'll assume the current map is "Teheran Highway")

Firstly, this is a "Kill" event, so for the processing of this event any rule that does not have an "On Kill" trigger is irrelevant for our purposes, and ProconRulz will effectively process the event with just the following subset of the rulz:

2. On Kill; Weapon Weapons/weapon/knife; Say Knife kill by %p%

4. On Kill; Map Subway; Damage ProjectileExplosive; Say %p% no rockets on Metro; Kill

5. On Kill; Damage ProjectileExplosive; Say %p% rocket killed %v%

So ProconRulz will start with rule #2, the "On Kill" trigger is good, so next ProconRulz will check the "Weapon Weapons/weapon/knife" condition, using the "SMAW" weapon key from the kill event. This condition will *fail* (the weapons do not match). ProconRulz will proceed no further with rule #2 and continue to the next rule with a matching trigger, i.e. rule #4.

Moving in to rule #4, ProconRulz will check the “Map Subway” condition, which will try and find the string “Subway” embedded in either the file name or map display name of the current map (“Teheran Highway, filename MP_003). This substring will not be found and the condition will *fail*. ProconRulz will proceed no further with rule #4 and continue to the next rule with a matching trigger, i.e. rule #5.

Using multi-line rulz

To improve the clarity of longer rulz, or more complex sets of rulz, ProconRulz has a system of allowing the processing for a single event to flow from one rule to the next.

Multi-line rulz copying a trigger from above

*A rule **without a trigger** will be treated as if the first rule above that **does** have a trigger is joined onto the front.*

So if you have (for example) three rulz that should all be triggered by a sniper rifle kill event, you could use:

```
On Kill; Damage SniperRifle

        PlayerCount 3; Kick

        PlayerCount 1; Kill

        Say No snipers on this server, %p% !!
```

This is *exactly* equivalent to:

```
On Kill; Damage SniperRifle

On Kill; Damage SniperRifle; PlayerCount 3; Kick

On Kill; Damage SniperRifle; PlayerCount 1; Kill

On Kill; Damage SniperRifle; Say No snipers on this server, %p% !!
```

On the 4th sniper kill the player will be kicked, on the 2nd they’ll be killed, and on the 1st they’ll be given a warning. Note that the first rule in this series is effectively a “do nothing” – it exists purely to set up the trigger and condition for the following rulz.

To summarize this ‘trigger propagation’ between multi-line rulz, see this symbolic example:

On X;A;B	goes to	On X;A;B
C;D		On X;A;B;C;D

E;F   On X;A;B;E;F

Please note with this technique, the actual number of rulz you have will equal the number of lines in your rulz. I.e. if the first line has a trigger (e.g. On Kill) and the following lines don't, then you end up with multiple rulz, each with an On Kill trigger.

Multi-line rulz using '+' as a simple continuation character

In addition to this 'trigger propagation' between multi-line rulz, ProconRulz allows rulz to be *continued* across multiple lines by starting subsequent lines with a '+'.

Like the 'trigger propagation' for multi-line rulz described above, this is a convenience for the layout of your rulz and does not affect the way ProconRulz actually processes the rulz after the '+' lines are joined together.

For example:

```
On Kill;Weapon SMAW

+        Log %p% killed %v% with SMAW

+        Kill
```

Is *exactly* the same as

On Kill;Weapon SMAW; Log %p% killed %v% with SMAW;Kill

I.e. the lines with '+' as the first character are joined onto the line above. Using a similar abstract example to the 'trigger propagation' example above:

On X;A;B

+ C;D

+ E;F   all goes to   On X;A;B;C;D;E;F

Unlike the 'trigger propagation' technique described before, '+' characters will convert multiple lines into a *single* rule.

Applying actions to other players using TargetPlayer and TargetAction

By default, ProconRulz is applying *actions* in the rule to the player that *triggered* the rule. This behavior is so common that many ProconRulz users don't even have to think about it. E.g. in the following rule:

On TeamKill;Damage ProjectileExplosive;**Kill**

The "Kill" action highlighted in bold is applied to the player that triggered the TeamKill event with a weapon that causes damage "ProjectileExplosive". The point of the TargetAction condition is to obtain a valid player name for an in-game player either from a rulz variable or from some player chat, and to apply some actions to the player thus identified (using the TargetAction action). As in the example in green above, if you only want to apply an action to the player that triggered the rule (e.g. slay a player that teamkills) then you can just use the basic actions like Kill, Kick, Ban and not worry about TargetPlayer/TargetAction at all.

How TargetPlayer is used

TargetPlayer [<string>] is used to set up the PlayerName that any TargetAction in the same rule is aimed at, in a substitution variable called %t%. By far the most common use is the simple condition "TargetPlayer" – this will match a valid playername to the text in a current "On Say" rule. The simplest (artificial) example would be to display the name of the target player in the chat window when the round starts:

On Round;TargetPlayer bambam;Say The target player is [%t%]

Remember this is a trivial *artificial* example, using the playername hardcoded in the TargetPlayer condition, although it *will* work.

1. When the map loads, the On Round trigger will fire, this rule will get executed,
2. The "TargetPlayer bambam" condition will scan the names of all players on the server looking for a name that contains "bambam". Assuming I am on the server, this will succeed, setting %t% equal to "bambam".
3. The "Say" action will then be executed, sending the chat message "The target player is [bambam]".

The example above doesn't actually use any 'TargetAction's (i.e. applying an action to the chosen player), but the TargetPlayer condition still does what it is supposed to do and populates the %t% variable with the chosen PlayerName.

If the optional string is given, then that is used to scan the current in-game players to try and match it in a name of a player. Generally if a string is used it will be the exact name of a player either literally (e.g. "TargetPlayer bambam"), or by using a variable ("TargetPlayer %v%"). %t% then contains the name of the player matched.

TargetPlayer can conveniently be used with a *variable* giving the string of the player name, most simply in the case of the victim of a kill, i.e. “TargetPlayer %v%”. Equally, rulz variables can be set with a player name in some prior execution of a rule, and then inserted in the TargetPlayer condition here (we will talk more about rulz variables in a later section).

If the optional string is NOT given, then it is assumed this rule has an On Say trigger and TargetPlayer will search the %text% player chat value for a string that matches a single player name. For example:

```
On Say;Admin;Text !kill;TargetPlayer;PlayerSay Kill %t% confirmed;TargetAction Kill
```

As a slight enhancement, if you have a "Text" condition before TargetPlayer in the same rule, TargetPlayer will only search for a PlayerName AFTER the matching text in the player chat. In the rule above, given player chat “!kill bam", ProconRulz will look for a PlayerName matching the player chat text starting from the letter "b" (i.e. will use the string "bam" to search ANYWHERE in any of the PlayerNames of the players currently on the server).

TargetPlayer ONLY succeeds if it matches a single player. If it matches no players, or matches multiple players, then it fails. (The action from prior versions of ProconRulz **TargetConfirm is no longer needed** in ProconRulz (it does nothing)).

So in the example in green above, TargetPlayer should succeed with a player target of “bambam” written into the substitution variable %t%, and any subsequent TargetAction will use “bambam” as the player that the action should be applied to, as explained further below.

How TargetAction is used

TargetAction <action> is the way you tell ProconRulz "apply this action to the TargetPlayer target player with the name stored in %t%, not the usual %p% that triggered this whole rule".

So with the latest version of ProconRulz, a rule could mix actions for just about anybody, using pairs of TargetPlayer and TargetAction clauses plus also any simple actions on the %p%. The only way I've seen this commonly used is with a single TargetPlayer, plus also simple actions affecting %p%. For example on a TeamKill, IF you wanted to kill the teamkiller AND do a PlayerSay action to the victim (ignore the fact that PlayerSay is limited in BF3):

```
On TeamKill;Kill;TargetPlayer %v%;TargetAction PlayerSay Admin slayed %p% that TK'd you
```

Assuming a teamkill has occurred in the game e.g. “**slarti** teamkilled **bambam** with **SMAW**” then the execution of the rule above proceeds in the following steps:

1. Running down the list of rulz, those with “On xxx” triggers that are *not* TeamKill are skipped.
2. This “On TeamKill” trigger succeeds, and ProconRulz proceeds left-to-right in the rule.
3. “Kill” is an action that sends an admin command to slay the player that triggered this event, in this case **slarti** so that player is killed.
4. “TargetPlayer %v%” goes through a *variable substitution* step with %v% replaced with the player name of the victim of the teamkill. Given the example event (“**slarti** teamkilled **bambam** with **SMAW**”), the victim is bambam, so %v% is assigned the value “bambam” and hence ProconRulz executes “TargetPlayer bambam”. This succeeds (we can assume a player called bambam *is* in the game, he was just TK’d..,) assigning a variable %t% the value “bambam”.
5. The last clause in the rule is “TargetAction PlayerSay Admin slayed %p% that TK'd you”. The TargetAction first word modifies the action to be *applied* to the player name in %t% (i.e. “bambam”) rather than the usual default %p% (the player who triggered the action, in this case “slarti”). So the PlayerSay command is applied to player “bambam”. As with the previous TargetPlayer step, the command is passed through a *variable substitution* step in which %p% is replaced with “slarti”, and bambam will see the chat text “Admin slayed slarti that TK’d you”

Writing an In-game Admin Plugin

ProconRulz can be used easily to write a flexible in-game admin plugin, i.e. the kind of thing where a clan member *in the game* can chat something like "!kick bam" with the result that player bambam gets ejected from the server.

A sample set of rulz for an in-game admin plugin is provided in Appendix 5.

These examples use the same format for command names as the default in-game admin plugin (i.e. !<command name>). So if you want to use this format, disable that plugin. Or choose a different format for your test command names (like "xpyell" instead of "!pyell").

Let's build up an in-game admin plugin a rule at a time.

In-game commands that don't need a player name e.g. !yell <message>

Firstly, your rulz need to recognise when an admin player has said the name of a command. E.g. you could implement a "!yell" command that yells a message to all players telling them not to camp. Note that there is *nothing* special about the "!" character as far as ProconRulz is concerned – it's just a letter you decided to include in the command name.

An On Say trigger will be used to detect when any player has said anything (i.e. including any admins). An Admin condition can be added if desired to limit the rest of the rule to only apply to chat from players defined as an administrator to Procon. A Text condition can be used to test whether the chat text contains a particular string (i.e. !yell). Finally a Yell command will output the desired message to all players. I.e. the complete rule would be

```
On Say;Admin;Text !yell;Yell %targettext%
```

Congratulations, you've just written an in-game admin plugin. Limited, I admit, but hey at least you didn't have to write 1000 lines of C#. But wait, there's more.

In-game commands that target one player e.g. !slay bam

The most obvious requirement is that your in-game admin will want to enter a command that applies a command (e.g. Kill) to a particular player. This is where the TargetPlayer condition gets used, as described in the previous section. Many ProconRulz simply default to applying actions *to the player that triggered the rule*, and ProconRulz is considerably simplified as a result (this player can often be thought of as %p%). The TargetPlayer action switches the target of any ProconRulz actions to a playername extracted from a chat message that triggered an On Chat rule, stored in the %t% variable (this is a recap of the previous section).

So let's keep the "!yell" in-game command, and add another command to slay any player in the game. We'll choose an in-game command name "!slay" for this command, with the usage being to follow this command string with the name (or part of the name) of the player you want to slay (i.e. kill). We'll use the Admin condition to limit this rule to only apply to chat coming from players defined in Procon as an admin as with the "!camp" command above, and we'll use the

TargetPlayer condition to dig a valid player name out of the chat text. Slaying the selected player is done simply with the Kill action.

On Say;Admin;Text !slay;TargetPlayer;Kill

You can dress things up a bit with in-game Say or Yell messages when you slay people, but if all you ever want to do is rulz of the above format then you're good to go and need read this section no further.

In-game commands with additional parameters e.g. !kick <player> <message>

ProconRulz as of version 41 can actually pick out additional bits of an in-game command and use them in the actions, e.g. you *can* (as of v41) enter a command such as "!kick bam too much whining" and use the "too much whining" part of the chat text in your rule. It will help to understand how ProconRulz uses a couple of important variables %text%, %targettext% and %t% to achieve this aim.

We'll start with a complete "!kick" command rule and show how it is processed. Assume an in-game admin has typed "!kick bam too much whining" and you have the following rule:

On Say;Admin;Text !kick;TargetPlayer;TargetAction Kick %targettext%

When the admin chats any message, this rule will be fired because it has an "On Say" trigger. An important detail is the chat text will be available to the rule in a variable called %text% (see Appendix 1 for a list of all the built-in variables). So now %text% will contain "!kick bam too much whining".

The Admin condition will succeed as expected for this 'admin' player as before.

The Text !kick condition will search the %text% variable to try and find the substring "!kick". This condition will succeed (matching the first 5 letters of %text%), and the Text condition will initialize a *new* value for another variable %targettext% containing the content of %text% *after* the occurrence of "!kick", i.e. %targettext% will be "bam too much whining" (leading spaces are trimmed). So %text% -> Text condition -> %targettext%.

So far, so good. A text condition has successfully narrowed down this rule to situations where an admin has typed "!kick", and we've stored the rest of the text in a variable %targettext%.

The TargetPlayer condition is going to use the text stored in %targettext% (hence the name of the variable) to search the list of currrent in-game playernames and try and find a match. If %targettext% has space in it (as of v41), TargetPlayer will just use the *first* word in %targettext% to use for the search. %targettext% is "bam too much whining" so in this case TargetPlayer will search with "bam".

The TargetPlayer condition succeeds if it finds a playername match for a single player, in this case assume player Bambam is online, so it succeeds. At this point the *new* variable %t% is initialized with this playername, i.e. %t% = "Bambam". **Also**, TargetPlayer will (as of v41) update the %targettext% variable to leave it with the text remaining **after** the word used in the playername search, i.e. %targettext% is now “too much whining”.

The remaining action “TargetAction Kick %targettext%” applied the kick action to the player defined as a target in %t% (i.e. Bambam), giving the reason “too much whining”.

In summary

On Say → %text% = “!kick bam too much whining”

Text !kick → %targettext% = “bam too much whining”

TargetPlayer → %t% = Bambam (having used “bam” from %targettext% for search)

→ %targettext% = “too much whining”

TargetAction Kick %targettext% → kicks Bambam with reason “too much whining”.

So that is how you end up with the rule for !kick:

On Say;Admin;Text !kick;TargetPlayer;TargetAction Kick %targettext%

The exact same format would be used for a player yell rule:

On Say;Admin;Text !pyell;TargetPlayer;TargetAction PlayerYell %targettext%

Using variables in your rulz

The power of ProconRulz is significantly enhanced through the support for *variables* that can store *values*. There are **two** types of variable, although both types are used similarly in rulz:

1. **Substitution variables** are automatically populated by ProconRulz to hold the correct values during the execution of each rule. The full list of substitution variables is given in Appendix 1, but most common examples include %p% for the player name of the player that triggered the rule, %v% for the player name of a victim in a kill, or %c% for the count of the number of times the current rule has been triggered by the current player.
2. **Rulz variables** are created by the person that entered the rulz into the ProconRulz settings. The format of the variable name is the same as for substitution variables (i.e. %<varname>%). A value is stored into a rulz variable explicitly during the execution of a rule via the use of "Set", "Incr" or "Decr" clauses. After the value of a rulz variable is set, that value is accessible in any rule at any time during a round. So rulz variables can be used to accumulate a count of how many times a player has used a particular weapon, for example, or the name of the player that last killed somebody. Rulz variables can be used that very easily accumulate counts for a player, squad, team, or the whole server.

Using substitution variables

Appendix 1 lists the available *substitution variables.* For example, the most common substitution variable used is %p%, the name of the player that triggered the current rule. Wherever %p% appears in the rule, ProconRulz will replace (i.e. substitute) %p% with the player name. For example:

On Spawn;Say Player %p% just spawned

In this example, when the player with the name "bambam" spawns into the game, the "On Spawn" trigger will fire and ProconRulz will execute the "Say" action in the rule, resulting in the following text appearing in the in-game chat window:

- Player bambam just spawned

%p% is not the only substitution variable that can contain a player name. In an "On Kill" event, the player name of the *victim* is recorded in a substitution variable called %v%.

A possible misunderstanding is that the use of %p% somehow affects the target of a Kill/Kick/Ban action. In the action "Kick %p% kicked for hacking", %p% has *no* influence on who is kicked. The Kick action is format "Kick <message>", in this case the message is "%p% kicked for hacking", which becomes "bambam kicked for hacking.

Mostly, the other substitution variables just contain useful stuff available in the event that triggered the rule, e.g. %w% for the weapon used in a kill/teamkill/suicide.

Using counts

Many rulz use the concept of *count*, and ProconRulz has built-in support to make the use of counts more convenient. Rather than specifically counting the number of kills with each weapon, or the number of spawns by each player, etc, ProconRulz has a simple general-purpose method of *keeping a count of each time each player has triggered each rule*.

In any given rule, the substitution variable %c% (of which more later in this section) contains this count. E.g.

On Kill;Weapon SMAW;Say Player %p% has killed with the SMAW %c% times

The first time I kill someone with the SMAW, the say action will have “1” substituted for %c%. The second time it will be “2” etc. The count can be tested using conditions such as PlayerCount (also of which more later in this section).

The next section will discuss completely general-purpose rulz variables and any count *could* be implemented using those, but ProconRulz has the following convenience features where the most common usage is required:

PlayerCount N: this is a condition that succeeds if the current player has triggered this rule *already* N times (i.e. this event is the N+1th or higher). So in a rule

On Kill;PlayerCount 3;Say Player %p% has more than 3 kills

The chat message “Player bambam has more than 3 kills” will appear on each kill by bambam *after* the third.

%c%: This *substitution variable* contains the actual *count* of the number of times the current player has triggered the current rule. So the above rule could be modified to:

On Kill;PlayerCount 3;Say Player %p% has more than 3 kills (actual count is %c%)

The typical application is to Kill players that exceed a threshold, if you want to reduce players ‘spamming’ a particular weapon.

TeamCount, ServerCount: In addition to PlayerCount and %c% for the current player, ProconRulz maintains counts for the team (TeamCount and %tc%) and also for the server (ServerCount and %sc%).

Using rulz variables

In addition to using the pre-defined ProconRulz substitution variables (%p%, %v% etc), admins that write ProconRulz rulz can embed their own variable values. These general-purpose variables can be given any arbitrary %<name>% by the admin and are called "rulz variables" (just to differentiate them from fixed-definition substitution variables described above)

All rulz variables are conveniently set to "0" when the round starts, and can be incremented using the "Incr %varname%" statement. Other useful statements include Set, Decr and If, which are described later in this section.

As with PlayerCount and %c%, the most simple use of these variables can be *per-player* and that is the default way rulz variables are treated. So if you have a rule:

On Kill;Incr %kills%;Say Player %p% has %kills% kills

When player "bambam" triggers their first kill event (i.e. they kill their first player in-game), the %kills% variable *for player bambam* will be incremented from "0" to "1" and a chat message will appear:

- Player bambam has 1 kills

On the second kill, %kills% *for bambam* is incremented again, and players see:

- Player bambam has 2 kills

If *another* player makes a kill, then *their* %kills% variable is incremented, not affecting the count for bambam, so you could subsequently see a message in game:

- Player pebbles has 1 kills

Decr %varname% will reduce the value of %varname% by one, but will not reduce the value below zero.

Set %varname% <value> will assign the value of <value> to the variable %varname%. Note that <value> can be a number or literal string (not containing spaces), or can be a substitution variable, rulz variable or combination of those. For example, a TeamKill rule might be used to set a variable containing the name of the victim most recently killed by the current player:

On TeamKill;Set %tked% %v%

This isn't a huge amount of use on its own, but opens up the possibility of *another* rule using this variable, e.g. if a player wants to know the name of the last player they teamkilled, you could give them a chat message with the rule:

On Say;Text !tk;Say %p% your most recently tk was %tked%

If player bambam teamkills pebbles, then %tked% *for player bambam* will be set to “pebbles”, and when bambam types the chat “!tk” the following message will appear:

- bambam your most recent tk was pebbles

That just about covers it for *per-player* rulz variables. These are useful for per-player counts and a few other uses, but would be insufficient on their own.

ProconRulz also supports *per-squad*, *per-team* and *per-server* rulz variables. The easiest comparison with the per-player variables discussed so far is with per-server rulz variables. These are created using a simple naming convention, by naming the variable anything starting “%server_” (names must also *end* with “%”).

Rulz variables beginning “%server_”, e.g. “%server_kills%” are stored *once* in the server, i.e. all players share the same variable. The simple per-player kill counter given in the first example in this section could be changed to use %server_kills% instead of %kills%:

On Kill;Incr %server_kills%;Say We have counted %server_kills% kills

When player “bambam” triggers their first kill event (i.e. they kill their first player in-game), the %server_kills% variable (*shared by all players*) will be incremented from “0” to “1” and a chat message will appear:

- We have counted 1 kills

On the second kill, %server_kills% is incremented again, and players see:

- We have counted 2 kills

If *another* player makes a kill, then *the same* %server_kills% variable is incremented, so you could subsequently see a message in game:

- We have counted 3 kills

Squad and team rulz variables work exactly the same, prefixed with “%squad_” and “%team_” respectively, this time the variable will be shared across a squad or team.

Using indexed variables

The ‘per-player’ rulz variables described above are convenient, particularly when accumulating some per-player count, but ProconRulz allows you to be explicit about associating a player name (in fact any other variable value) with a rulz variable. This is a very powerful feature worth taking the time to understand (it's not too complicated) and easiest to illustrate using per-server rulz variables. An *indexed* variable is used in the following rule:

On TeamKill;Set %server_tked[%p%]% %v%

The indexed variable is %server_tked[%p%]% with the [%p%] embedded in the variable name.

Given a teamkill event in which player "bambam" kills "pebbles", substitutions will be applied within the Set statement so it will actually be Set %server_tked[bambam]% pebbles. In effect, this sets a per-server rulz variable called %server_tked[bambam]% to the value "pebbles". At this point we're in a similar situation to what was achieved earlier using per-player rulz variables, i.e. bambam can display the name of the player he tk'd via the following rule:

On Say;Text !tk;Say %p% you most recently teamkilled %server_tked[%p%]%

The important difference is that with the indexed variables we are able to be *explicit* about which player name we're indexing with (it doesn't always have to be the name of the player that triggered the rule). So the following pair of rulz allow the *victim* to display the name of their killer:

On TeamKill;Set %server_tker[%v%]% %p%

On Say;Text !tk;Say %p% you were most recently tked **by** %server_tker[%p%]%

If you look at the first of the two rulz, you see %p% and %v% are *reversed* from before. The indexed variable %server_tker[<playername>]% now contains *the name of the player that most recently tk'd playername*. I.e. the variable is indexed using the *victim* not the killer, and the value stored is the name of the killer, not the victim as before. If you try the substitutions on the Set statement when "bambam teamkills pebbles" hopefully this will become clear.

The use of indexed variables using %v% and %p% appropriately is extremely valuable to understand if you want to write sophisticated rulz. With this you can design ProconRulz that do almost anything, e.g. you can effectively keep track of player names to implement in-game commands such as !voteban or !punish.

Indexed variables are not limited to just using [%p%] and [%v%] although that is the most common and probably the most powerful. Per-weapon counts could be implemented using a variable such as %server_kills[%w%]%, and these rulz variable can themselves be used as indexes as in %server_kills[%killer_name%]%.

Please note that *multiple* indexes can be used in the same variable (although I don't know of a use for this yet) and the format is to put each index in their own square brackets, e.g.

%server_kills[%w%][%p%]%. That example might be used to collect counts for each player with each weapon.

The relationship between server, team, squad and player rulz variables

Without wishing to make a drama out of this, the use of per-player variables is designed to be *exactly* equivalent to the use of “%server_” variables *indexed with the player name*.

E.g. %kills% and %server_kills[%p%]% could be used interchangeably.

Similarly, a per-team variable such as %team_kills% is equivalent to %server_team_kills[<team id>]% and %squad_kills% is equivalent to %server_squad_kills[<team id>][<squad id>]%. These two cases are likely to be much less useful than the per-player/per-server equivalence.

Using the %team_score% variable

This holds the number of tickets remaining for each team, and is updated about once every 30 seconds from the game server.

This is the figure displayed on the Procon 'players' tab as the score at the top of each team.

So if, for example, a player spawns, the number of tickets remaining for that player's team could be displayed with a rule (%pt% is the name of the player's team, e.g. US Army):

On Spawn;Say Tickets left for %pt% is %team_score%

This would display (an annoying) say-text message on each spawn e.g. Tickets left for US Army is 37.

This is a team variable (it begins "team_") which means by default it applies to the current player's team (that triggered the rule).

ProconRulz allows the value for any team to be accessed in any rule by referring to the same variable as %server_team_score[<team number>]%. Teams are numbered 1,2,... (Conquest and Rush are just 1 and 2).

So if you want to display the ticket count for both teams in a single rule you can do this with

On Spawn;Log Team 1 tickets = %server_team_score[1]%, Team 2 tickets = %server_team_score[2]%

This would say Team 1 tickets = 37, Team 2 tickets = 55

Note you still need an event to trigger the rule in the first place, and "On Spawn" probably makes sense although you could also use "On Kill". People might ask for a 'timer' event but it is not currently my intention to provide this in ProconRulz.

Note that %team_score% is only accurate on a ~30-second basis - the game server is updating the ticket count according to both spawns and an internal formula that counts down tickets when bases are captured by the enemy.

Advanced use of the %team_score% variable:

You can use your own variable within your rulz that update a ticket count (using spawns) *between* updates to %team_score% from the game server to refine the %team_score% figure within your rulz if you want slightly more granularity. I.e. if on a 30-second interval the %team_score% is updated to 50, and you know two players have spawned since, then you can estimate the new team score is 48. This is a better value that the 30-second-old update but until you get the next proper update from the game server you can't be sure what the exact ticket count will be.

Here is an (untested) example where you maintain your ticket count in a per-team variable called %team_tickets%, which you can than use as an ‘improved’ version of the built-in %team_score%.

```
# INITIALIZE YOUR TICKET COUNT TO WHATEVER YOUR CORRECT START FIGURE IS
On Round;Map Baz;Set %team_tickets% 250

# ADJUST YOUR TICKET COUNT TO %TEAM_SCORE% IF THAT IS LOWER
On Spawn;If %team_tickets% > %team_score%;Set %team_tickets% %team_score%

# DECREMENT YOUR %TEAM_TICKETS% WHEN SOMEONE SPAWNS
On Spawn;Decr %team_tickets%

# NOW IN YOUR RULZ YOU HAVE YOUR UPDATED %TEAM_TICKETS% VARIABLE TO USE
On Spawn;If %team_tickets% < 10;PlayerYell %pt% you have 10 tickets left !!!
```

Using “%ini_...%” variables to store longer-term values

ProconRulz variables are generally designed to be *reset to 0* at the start of each round. Because players join the server, play, and leave never to be seen again, this makes sense in the majority of cases.

ProconRulz v42 onwards includes support for variables that *never* get reset or deleted, even if Procon or ProconRulz are stopped and restarted. These variables can be used by choosing names beginning with “ini_”, and the actual value will be saved in an “ini” file in the folder Procon\Configs.

For example:

1. You have a server called “myclan.net:90210”
2. You have a rule such as “On Say;Text save;Set %ini_vars_said% %text%”
3. In-game, some player chats “save hello world”

At this point, ProconRulz will write to a file Procon\Configs\myclan.net_90210.ini, creating an entry:

```
[vars]

said=save hello world
```

You can inspect the file using Notepad, or any other text editor, to see what’s been stored in there.

If the file (or the entry) already exists, then it will be updated with this new value.

Note that %ini_vars_said% can be used throughout your rulz like any other variable, i.e. you can update it with the Set, Incr or Decr statements, test it with an If statement, or simply include the variable in any action (e.g. Log) and the current value will be substituted.

In the example %ini_vars_said% above, the variable is stored in a [vars] section of the ‘ini’ file, as an ini-file-value with the name “said”. The general format is:

%ini_<section name>_<var name>%

Make sure you include all the elements of the variable name, i.e. “ini”, “<section>” and “<variable>”. If you don’t care about a particular section name, just make one up (or use “vars”). But the format of the ini variable name is important for ProconRulz to store the value properly, and you can confirm that by simply looking at the ini file using Notepad.

Example uses of %ini_...% variables

You could keep track of the total number of ‘sessions’ on your server (i.e. players joining) with a simple rule “On Join;Incr %ini_vars_joins%”

This will create (or add to) a section:

[vars]

joins=77

You could store the last time each player joined your server with a simple rule “On Join;Set %ini_joined_%p%% %ymd%__%hms%”

Resulting in:

[joined]

bambam=2012_08_20_18:23:45

You could enable/disable certain rulz by including an “If %ini_rulz_teamkills% == 1” condition in your ‘team kills’ rulz, and manually editting the ini file with:

[rulz]

teamkills = 1

Resetting your ini file

The simplest thing to do would be to quit ProconRulz, inspect your Configs/<server name>_proconrulz.ini file with Notepad, and change it however you want.

However, partly for programming capability and partly to help users that cannot access directly the ‘Configs’ folder (e.g. sometimes with a hosted Procon service), rulz-based commands are available to ‘reset’ either the whole ‘ini’ file, just a section, or an individual variable. The basic method is to “Set” a corresponding “ini” variable to ZERO:

Set %ini% 0 will reset the WHOLE INI FILE, i.e. it will become empty

Set %ini_<section name>% 0 will DELETE THE SECTION

Set %ini_<section>_<variable>% 0 will DELETE THE VARIABLE (and the value will read as 0)

Arithmetic in your rulz

ProconRulz supports *basic* arithmetic, i.e. +, -, *, /. Note that bracketed expressions are not supported.

Arithmetic can be used in Set and If statements. E.g. as a reminder, the ‘Set’ statement has the format “Set <variable> <value>” so most simply, you could ‘set’ a variable %x% to be the sum of %y% and %z% with Set %x% %y% + %z%.

For another example, arithmetic could be tested in-game with the following rulz:

```
On Say;Text test;

Set %server_x% 1 + 2 * 3; Say x = %server_x%

If %server_x% * 2 > 13;Say 2*x is bigger than 13

If %server_x% < 2 * 4;Say x is less than 8
```

When you chat the word “test”,

1. variable %server_x% will be set to 1 + 2 * 3, i.e. **7**.
2. If %server_x% * 2 > 13; becomes “If 7 * 2 > 13”, i.e. “If 14 > 13” which *succeeds* and the Say message “2*x is bigger than 13” will be sent.
3. If %server_x% < 2*4; becomes “If 7 < 2*4”, i.e. “If 7 < 8” which also *succeeds* and “x is less than 8” will be output into chat.

An obvious example in a first-person-shooter would be testing kills/deaths:

```
On Spawn;Incr %_deaths%

On Kill;Incr %_kills%

On Kill;If %_deaths% != 0;If %_kills%/%_deaths% > 5;Say %p% has a > 5:1 kill ratio
```

The new arithmetic is in the “If %_kills%/%_deaths% > 5” statement which calculates the value of %_kills% divided by %_deaths% and if the result is bigger than 5 the condition succeeds.

Using ‘rounding’ to decimal places for your variables

Once you use arithmetic, you may end up with a variable value such as 1.234567 (i.e. you used division somewhere, maybe this was a K/D ratio…). If you want to display this value in a ‘Say’ or ‘Yell’ command, having so many decimal places might be confusing, so ProconRulz provides a way to limit the variable to a smaller number of decimal places.

Note that the ‘decimal places’ required forms part of the variables *name*, as a digit following a decimal point at the end of the name. I.e. %x.3% will always contain a value with a maximum of three decimal places.

Note that %x.3% and %x% are totally separate variables and should not be confused with each other.

As a simple example “Set %x.3% 1.4567;Say %x.3%” will echo “1.457” in chat (note that the value is *rounded* to 3 decimal digits before being assigned in the ‘Set’ statement, not just truncated to that number of places).

Triggers, Conditions and Actions summary list

On Round	Admin	Say <message>
On Spawn	Admins	PlayerSay <message>
On Kill	Protected	VictimSay <message>
On TeamKill	Team <part name>	AdminSay <message>
On Suicide	Teamsize <N>	Yell <message>
On Join	Map <part name>	PlayerYell <message>
On Leave	MapMode <part name>	Log <message>
On Say	On Kill;…Headshot;…	Both <message>
	On Kill;…Weapon <key>;…	All <message>
	On Kill;…Damage <key>;…	Kill [<delay milliseconds>]
	On Kill;…Range <N>;…	Kick <message>
	On Spawn;..Kit <kit> [<N>];..	Ban <message>
	On Spawn;..Weapon <weapon> [<N>];..	TempBan [<seconds>] <message>
	On Spawn;..Damage <damage> [<N>];..	PBBan <message>
	TeamKit <kit> <N>	PBKick [<minutes>] <message>
	TeamWeapon <weapon> <N>	TargetAction <action>
	TeamSpec <spec> <N>	End
	PlayerCount <N>	Continue
	TeamCount <N>	
	ServerCount <N>	
	Rate <count> <seconds>	
	PlayerFirst	
	PlayerOnce	
	TeamFirst	
	ServerFirst	
	On Say;..Text <string>;..	
	TargetPlayer [<name>]	
	Ping <milliseconds>	
	Set <%varname%> <value>	
	Incr <%varname%>	
	Decr <%varname%>	
	If <%varname%> <compare> <value>	

Triggers

The following listing gives the trigger events and the *most commonly used* substitution variables in each case. For the full list of substitution variables see Appendix 1.

"On Round": %m% contains the map name, %mm% the map mode

"On Spawn": %p% contains the name of the spawning player, %pt% his team. In BFBC2 the kit/weapon/specialization loadout is also available in %k%, %w% and %spec% but not yet in BF3.

"On Kill": %p% contains the killer name, %v% the victim, %w% the weapon, %d% the damage

"On TeamKill": as above

"On Suicide": %p%, %w% and %w% as you would expect for a kill. Note that in BF3 (not BFBC2) an admin kill (e.g. with the ProconRulz 'Kill' action) *will* trigger the "On Suicide" rulz, with weapon Death. So if you want a typical On Suicide rule for a player that blows himself up or falls off a cliff, use On Suicide;Not Weapon Death,…

"On Join": %p% contains the player name that joined. You can use conditions such as 'Protected' or 'Admin' to decide whether to announce something special.

"On Leave": %p% contains player name

"On Say": %p% contains the name of the player that just entered the chat. %text% contains whatever it is they entered.

Generally if you have a *Weapon*, then you can assume *Damage* will similarly be available. Also when you have a player %p% then the player team and squad will be available in %pt% and %ps% respectively.

Conditions

<condition> = (prefix with "Not " to reverse meaning)
"Admin": player is a server Admin - typically use Not Admin

"Admins": at least one server admin is on the server

"Protected": player is admin or in reservedslots list and protected from ProconRulz kicks and kills

"Team <string>": The name of the players team includes <string> e.g. “Team attack” means player is an Attacker in Rush mode

"Teamsize <N>": smallest team has N or fewer players

"Map <string>": map name or filename contains string, e.g. Map Nelson

"MapMode <string>": current map mode contains string e.g. MapMode rush

"On Kill;Headshot": kill was with a headshot

"Kit <kit> [<N>]": player originally spawned with this kit, >N players on team with this kit incl. player

"On Spawn;Weapon <weapon> [<N>]": player has this weapon, weapon limit is >N as for Kit

"On Kill;Weapon <weapon> [<N>]": player has killed with this weapon >N times

"On Spawn;Damage <damage> [<N>]": player has spawned with weapon that can do this damage, limit N as Kit

"On Kill;Damage <damage> [<N>]": player has already done this damage N times

"On Spawn;Spec <specialization> [<N>]": player has this Spec, spec limit is N

"TeamKit <kit> <N>": team has >N players spawned with this kit *not necessarily including current player*

"TeamWeapon <weapon> <N>": team has >N players spawned with this weapon *not necessarily including current player*

"TeamDamage <damage> <N>": team has >N players spawned with weapons that do this damage *not necessarily including current player*

"TeamSpec <specialization> <N>": team has >N players spawned with this specialization *not necessarily including current player*

"On Kill;Range <N>": the distance of the kill was > N meters. **Note BFBC2 +/- 20 meter random error**

"PlayerCount <N>": player has *already* triggered rule N times during this round: i.e. succeeds at N+1) - subst text %c%

"Count <N>": same as PlayerCount

"TeamCount <N>": rule hit count for *Team* this round, see PlayerCount - subst text %tc%

"ServerCount <N>": rule hit count for *Server* this round, see PlayerCount - subst text %sc%

"Rate <N> <M>": player has triggered this rule N times in M seconds

"PlayerFirst": player has triggered this rule for *their* first time this round

"TeamFirst": player has triggered this rule for the first time for *their team* this round

"ServerFirst": this is the first time *any player* has triggered this rule this round

"PlayerOnce": player has triggered this rule for the first time *since joining server*

"On Say;Text <key>": player has just entered say text including this key string

"On Say;Text <key>;TargetPlayer": A playername can be found in the say text after <key>

"Ping <N>": Player ping (from last listPlayers update) is higher than N

Note: "OnSpawn;Kit Recon 2..." is equivalent to "On Spawn;Kit Recon;TeamKit Recon 2..."

Actions

"Say " <message>

"PlayerSay " <message>: only the player involved will see message

"SquadSay " <message>: only the player's squad will see message

"TeamSay " <message>: only the player's team will see message

"VictimSay " <message>: On Kill only, the player killed will see message

"AdminSay " <message>: only in-game admins will see message

"Yell " [<N>] <message>: Yell with N **seconds** on-screen time (default for N in plugin settings)

"PlayerYell " [<N>] <message>: only the player involved will see Yell message

"SquadYell " [<N>] <message>: only the player's squad will see Yell message

"TeamYell " [<N>] <message>: only the player's team will see Yell message

"Log " <message>: log to procon chat log but don't use Say

"Both " <message>: Say and Yell same message

"All " <message>: Say and Yell and Log same message

"Kill [<N>]": Kill after N milliseconds, for default see plugin setting

"Kick " <message>

"Ban " <message>: perm ban using EA ID

"TempBan " <N> <message>: temp ban for N seconds using EA ID

"PBKick " [<N>] <message>: PunkBuster kick (for optional N **minutes)** using PB GUID

"PBBan "<message>: PunkBuster ban using PB GUID

"TargetAction "<action> : <action> is applied to target after confirmation

"Exec "<game server rcon command> : <action> is applied to target after confirmation

Appendix 1. List of substitution variables

Anywhere a message appears in a rule, e.g. "Say %p% just spawned", the following substitutions can be embedded in the message:

Subsitute string	**Meaning**
%p%	Player name (On Spawn, or killer on a kill)
%pt% %ptk%	Player team name, e.g. Attackers, US Army (%ptk% is team **key**)
%ps% %psk%	Player Squad (%psk% is player squad **key**)
%v%	Victim name (On Kill|TeamKill|Suicide rules only)
%vt%	Victim team name
%k% %kk%	Player kit on spawn, e.g. Recon (%kk% is the kit **key**) - not available in BF3
%w% %wk%	Weapon (On Kill), e.g. SVU Snaiperskaya Short. Or list of weapons On Spawn. (%wk% is similar, but contains the weapon **key**)
%d% %dk%	Damage (On Kill) e.g. SniperRifle or VehicleHeavy. Or list of damage types On Spawn (%dk% is the damage **key**)
%spec% %speck%	Specializations (On Spawn only) e.g. 12-Gauge Sabot Rounds (%speck% is the specialization **key**) – not available in BF3.
%r%	Range (On Kill) - note each player position is randomised by 10 meters – not available in BF3.
%n% %ts1% %ts2%	Teamsize of current smallest team Teamsize of team with key 1 Teamsize of team with key 2

%pts%	Teamsize of team of current player
%c%	Count of the number of times this player has triggered this rule
%tc%	Count of the number of times this player's TEAM have triggered this rule
%sc%	Count of the number of times ALL PLAYERS ON SERVER have triggered this rule
%h%	Headshot (On Kill) - substituted with "Headshot" or blank
%m%	Map name e.g. Nelson Bay
%mm%	Map mode e.g. Rush
%t%	Target playername found from previous TargetPlayer condition
%targettext%	Text remaining for 'target' after a 'Text' condition. E.g. after an admin says "xkick bam idiot", triggering the following rule: "On Say;Admin;Text xkick;…" then %text% will be "xkick bam idiot", %targettext% will be "bam idiot". %targettext% is **also** updated after a successful TargetPlayer condition, leaving %targettext% as the text after the player name. E.g. with TargetPlayer added to the rule above, %t% will be "bambam" (assuming playername "bambam" is on the server) and %targettext% will be "idiot".
%ping%	Ping milliseconds for current player
%text%	Text from player On Say event in this rule
%pb_guid%	PunkBuster player GUID
%ea_guid%	EA player GUID
%ip%	IP address of current player
%pcountry% %pcountrykey%	Player country (e.g. Germany) in local language of server (e.g. Deutschland). International code for this country (e.g. 'de' or 'pl') is in %pcountrykey%. FYI country *keys* are in lowercase (thanks tarreltje).
%vcountry% %vcountrykey%	As above except for the *victim* in an On Kill;.. rule
%score%	The player score as reported in-game via the Tab key, i.e. the number of 'points' the player currently has. This is identical to

	%server_score[<playername>]% so you can look up the score for the ‘current’ player with %score% (or %server_score[%p%]%), or for another player by inserting the player name into the variable (e.g. the score of a victim in a kill would be %server_score[%t%]%)
%team_score%	Score for team of current player, typically number of tickets remaining. For details see section of this manual explaining this variable. This is a **tickets** count, not a ‘player score’ type value as in %score% above for each player.
%hms% %ymd% %seconds%	Current time **h**ours-**m**ins-**s**ecs in 24-hour clock e.g. “18:26:33”. Date in **y**ear-**m**onth-**d**ay e.g. 2012_07_17 Time in seconds (actually since Jan 1st 2012). This value is only likely to be used with some previous value *subtracted*, to get the difference in the time in seconds.

Appendix 2. List of all weapons, kits and specializations

Kits

Description	Kit key
No kit	None
Assault	Assault
Special Ops	Specialist
Engineer	Demolition
Support	Support
Recon	Recon

Weapons

If you want to see the *definitive* list of weapons in your game, see the ProconRulz plugin 'Details' tab – this will list the weapon keys actually available to the plugin. This list below is a snapshot from BF3 taken when this document was written.

Description	Weapon key	Damage	Kit
870 Combat	870MCS	Shotgun	None
AEK-971 Assault Rifle	AEK-971	AssaultRifle	Assault
AKS-74u Assault Rifle	AKS-74u	SMG	Demolition
AN-94 Abakan Assault Rifle	AN-94&Abakan	AssaultRifle	Assault
AS Val Supressed Assault Rifle	AS&Val	AssaultRifle	None
DAO-12 Striker Shotgun	DAO-12	Shotgun	None
Death	Death	None	None
Defibrillator	Defib	Melee	Assault
F2000 Assault	F2000	AssaultRifle	Assault
FAMAS Assault Rifle	FAMAS	AssaultRifle	Assault
FGM-148 Javelin	FGM-148	ProjectileExplosive	Demolition
FIM-92 Stinger	FIM92	ProjectileExplosive	Demolition

Glock 18 Pistol	Glock18	Handgun	None
HK53/MP5 Assault Rifle	HK53	AssaultRifle	None
Jackhammer/MK3A1 Shotgun	jackhammer	Shotgun	None
JNG90 Sniper Rifle	JNG90	SniperRifle	Recon
BF Premium Knife	Knife_RazorBlade	Melee	None
L96A1 Sniper Rifle	L96	SniperRifle	Recon
LSAT Light Machine Gun	LSAT	LMG	Support
M416	M416	AssaultRifle	Assault
M417 Sniper Rifle	M417	SniperRifle	Recon
M1014 Semi-automatic Shotgun	M1014	Shotgun	None
M15 Anti Tank Mine	M15&AT&Mine	Explosive	Demolition
M16A4 Assault Rifle	M16A4	AssaultRifle	Assault
WWII M1911 .45	M1911	Handgun	None
M240 Maschine Gun	M240	LMG	Support
M249 SAW	M249	LMG	Support
M26 MASS Shotgun	M26Mass	Shotgun	Assault
M27 IAR	M27IAR	LMG	Support
M320 Grenade luncher	M320	ProjectileExplosive	Assault
M39 Sniper Rifle	M39	SniperRifle	Recon
M40A5 Sniper Rifle	M40A5	SniperRifle	Recon
M4A1 Carbine	M4A1	SMG	Demolition
M60 LMG	M60	LMG	Support
M67 Grenade	M67	Explosive	None
M9 Pistol	M9	Handgun	None
Baretta M93R	M93R	Handgun	None
MedKit	Medkit	Nonlethal	Assault

Melee	Melee	Melee	None
MG36	MG36	LMG	Support
MK11 Sniper Rifle	Mk11	SniperRifle	Recon
Barrett M98B Sniper Rifle	Model98B	SniperRifle	Recon
MP7 Maschine Gun	MP7	SMG	None
Pecheneg Maschine Gun	Pecheneg	LMG	Support
PP-19 Bison SubMaschine Gun	PP-19	LMG	None
PP-2000 SubMaschine Gun	PP-2000	SMG	None
QBB-95 Light Machine Gun	QBB-95	LMG	Support
QBU-88 Sniper Rifle	QBU-88	SniperRifle	Recon
QBZ-95 Assault Rifle	QBZ-95	AssaultRifle	Demolition
Repair Tool	Repair&Tool	Melee	Demolition
Roadkill	RoadKill	None	None
RPG-7 Anti Tank rocket-propelled grenade launcher	RPG-7	ProjectileExplosive	Demolition
RPK-74M Light Maschine Gun	RPK-74M	LMG	Support
SCAR-L Assault Rifle	SCAR-L	AssaultRifle	Assault
SIG SG 550 Assault Rifle	SG&553&LB	SMG	Demolition
Saiga 20K Semi	Siaga20k	Shotgun	None
Simonow SKS-45 Rifle	SKS	SniperRifle	Recon
SMAW Anti Tank weapon	SMAW	ProjectileExplosive	Demolition
SPAS-12 Shotgun	SPAS-12	Shotgun	None
SV98 Snayperskaya	SV98	SniperRifle	Recon
SVD Sniper Rifle	SVD	SniperRifle	Recon
Aug A3 Assault Rifle	Steyr&AUG	AssaultRifle	Assault
Taurus .44 Mag revolver	Taurus&.44	Handgun	None

Type88 Maschine Gun	Type88	LMG	Support
USAS-12 automatic Shotgun	USAS-12	Shotgun	None
A-91 Assault Rifle	Weapons/A91/A91	SMG	Demolition
AK-74 Assault Rifle	Weapons/AK74M/AK74	AssaultRifle	Assault
G36C Assault Rifle	Weapons/G36C/G36C	SMG	Demolition
G3A3 Battle Rifle	Weapons/G3A3/G3A3	AssaultRifle	Assault
C4 Explosive	Weapons/Gadgets/C4/C4	Explosive	Support
Claymore mine	Weapons/Gadgets/Claymore/Claymore	Explosive	Support
KH2002 Assault Rifle	Weapons/KH2002/KH2002	AssaultRifle	Assault
Knife	Weapons/Knife/Knife	Melee	None
Magpul Personal Defense Rifle	Weapons/MagpulPDR/MagpulPDR	SMG	None
MP412 REX Revolver	Weapons/MP412Rex/MP412REX	Handgun	None
MP-443 Grach Pistol	Weapons/MP443/MP443	Handgun	None
MP-443 Grach Pistol (GM)	Weapons/MP443/MP443_GM	Handgun	None
P90	Weapons/P90/P90	SMG	None
P90 (GM)	Weapons/P90/P90_GM	SMG	None
SA-18 IGLA Air Defense	Weapons/Sa18IGLA/Sa18IGLA	ProjectileExplosive	Demolition
SCAR-H Assault Rifle	Weapons/SCAR-H/SCAR-H	SMG	Demolition
UMP-45 SubMaschine Gun	Weapons/UMP45/UMP45	SMG	None
L85A2/SA80 Assault Rifle	Weapons/XP1_L85A2/L85A2	AssaultRifle	Assault

ACW-R Assault Rifle	Weapons/XP2_ACR/ACR	AssaultRifle	Demolition
L86A2 Light Machine Gun	Weapons/XP2_L86/L86	LMG	Demolition
M5K Submachine Gun	Weapons/XP2_MP5K/MP5K	SMG	None
MTAR-21 Assault Rifle	Weapons/XP2_MTAR/MTAR	AssaultRifle	Demolition

Damage

None, Nonlethal, Impact, Melee, Handgun, AssaultRifle, LMG, SMG, SniperRifle, Shotgun, Explosive, ProjectileExplosive, VehicleWater, VehicleAir, VehicleStationary, VehicleLight, VehicleHeavy

Specializations

None available to admin tools yet in BF3

Appendix 3. BF3 Maps

Rulz can be limited to certain maps using the Map <name> condition, where <name> is a *substring* of either the map 'key' (e.g. MP_Subway) or the human-readable name. For normal usage it is recommended the map *key* is used, as there is less risk

Map	Human-readable name
MP_001	Grand Bazaar
MP_003	Teheran Highway
MP_007	Caspian Border
MP_011	Seine Crossing
MP_012	Operation Firestorm
MP_013	Damavand Peak
MP_017	Noshahr Canals
MP_018	Kharg Island
MP_Subway	Operation Metro
XP1_001	Strike at Karkand
XP1_002	Gulf of Oman
XP1_003	Sharqi Peninsula
XP1_004*	Wake Island

Appendix 4. BF3 Map Modes

It is common to want to limit rulz to certain map *modes*, e.g. you might want to ban Claymores only on 'Rush' maps (this is just an example…). This control is provided with the

MapMode <mm>

condition, where <mm> is the *key* for the map mode as listed in the table below. Note that <mm> is tested as a *substring* of the current map mode, i.e. MapMode Conquest will succeed for ConquestLarge0, ConquestSmall0 and ConquestSmall1.

So, to limit Claymores on rush maps, you could use the following rule:
On Kill;MapMode Rush;Weapon Weapons/Gadgets/Claymore/Claymore;Say %p% no Claymores permitted on Rush maps;Kill 100

MapMode	**Human-readable name**	**Intended player count**
ConquestLarge0	Conquest64	Up to 64
ConquestSmall0	Conquest	Up to 32
ConquestSmall1*	Conquest	Up to 32 (*Not on Wake Island)
RushLarge0	Rush	Up to 32
SquadRush0	Squad Rush	Up to 8
SquadDeathMatch0	Squad Deathmatch	Up to 16
TeamDeathMatch0	Team Deathmatch	Up to 24

Appendix 5. Sample In-Game Admin Plugin using ProconRulz

The earlier section in this manual described how a set of rule for in-game admin actually *works*. If you want to cut to the chase, here is a set of sample rulz that would provide a some useful in-game admin commands.

In every case, <playername> can be *any* partial string that *uniquely* identifies a player currently on the server. E.g. "!slay bam" will successfully kill player "bambam". If there's another player in addition to bambam called "simbam" the command will do nothing, and you would need to type "!slay bamb" or any other unique substring ("!slay amb" would also work). This is useful to nail hacker types that have a player name like "00!!|||!!lII!!00" – just type "!kick 00" (or !kick ||) –there is no need to try and match their whole name. In practice the approach works very well and I prefer it to the "approximate match of full name with confirmation" approach of the in-game admin plugin included with Procon.

!slay <playername>	Admin only
!kick <playername> [<reason>]	Admin only
!ban <playername> [<reason>]	Admin only
!nextlevel	Admin only
!yell <message>	
!pyell <playername> <message>	

Here are the actual rulz – you can add "Admin" conditions to the yell commands if you're paranoid:

```
On Say;Text !slay;Admin;TargetPlayer;PlayerSay %t% SLAYED;TargetAction Kill 100

On Say;Text !kick;Admin;TargetPlayer;PlayerSay %t% KICKED;TargetAction Kick %targettext%

On Say;Text !ban;Admin;TargetPlayer;PlayerSay %t% BANNED;TargetAction Ban %targettext%

On Say;Text !nextlevel;Admin;Exec admin.runNextLevel

On Say;Text !yell;Yell 5 %targettext%

On Say;Text !pyell;TargetPlayer;PlayerSay (%t%) %targettext%; TargetAction PlayerYell %targettext%
```

Glossary

Condition	If a given rule has a trigger (see glossary) matching the current event from the game server, then multiple *conditions* in that rule can then be tested to confirm the application of the actions in the same rule. A typical condition could be “Weapon RPG-7”. The way the condition is interpreted will vary depending on the trigger. With an “On Spawn” trigger (i.e. “On Spawn;Weapon RPG-7;…”), the Weapon condition will *succeed* if the player *spawns* having selected that weapon in his loadout. With an “On Kill” trigger the condition will succeed if the weapon in the rule was the one used for the kill.
Damage	Each weapon has an associated “Damage” type, for example the rocket-propelled grenade weapons (like M320, SMAW, RPG-7) have damage type “ProjectileExplosive”. If you want a rule to apply to a general class of weapons you can use a “Damage” condition in your rule rather than “Weapon”, and it will then apply to weapons that cause that damage.
Event	A real-time packet of data that ProconRulz receives from the game server. For example, every time a player is killed BF3 will send a ‘kill’ event across the rcon (see glossary) interface, containing the player name that *caused* the kill, the player name that was *killed* (i.e. the victim) and the key of the weapon that was used. ProconRulz allows rulz to be associated with a variety of events available from the server via the use of “On …” trigger (see glossary) clauses in the rules.
Plugin	An extension feature for Procon, many of which have been written by contributors providing additional scripted admin functionality, e.g. *SpamBot* which supports a user-customizable ‘Say’ message to appear in-game at regular intervals. ProconRulz is a particularly sophisticated plugin that allows the user to define the incremental functionality required.
Procon	An open-source game admin utility that connects to a game server’s rcon interface, provides a real-time status display of the server, e.g. the players connected, current map, K/D ratio’s, and provides a simple end-user interface to apply basic commands, e.g. a player can be kicked by clicking on their name. Procon is extensible via general support for *plugins.*
ProconRulz	ProconRulz is a plugin for Procon that is rapidly evolving to be a general-purpose *meta-plugin*. I.e. ProconRulz allows the user to define a script that the plugin interprets in real time, taking user-specified actions as events occur in the game. The origin of ProconRulz was to allow the most simple rulz to be specified limiting players spawning with certain weapons, e.g. for max 2 snipers-per-side, the single rule “Kit Recon 2” could be used. ProconRulz now has a wide range of events that can be recognized, conditions that can be tested, and actions applied. Since version 35, ProconRulz has support for general-purpose integer variables, so arbitrary counts can be accumulated and

	tested.
rcon	Remote Console interface, a text protocol provided by most game servers that transmits status information out to receiving clients (e.g. when a map changes on the server, a text message is sent on the rcon interface confirming this event and giving the next map name). The simplest client could simply be a text terminal where a user could watch the messages stream by. The rcon interface is two-way, i.e. you can send in very basic text *commands*, for example to kick a player. Game server hosting companies give their customers the IP address and port of the game server, which is then used in Procon.
rulz variable	An element of a rule used to store a value, given a name such as %kills%.
substitution variable	A name such as %p% which is replaced with a value from the game (in this case 'player name') when the rule is executed.
trigger	ProconRulz requires an "On …" clause for each rule that says which event from the game server will cause a given rule to be applied. *Every* rule has an associated trigger – if the "On …" clause is omitted then ProconRulz assumes the trigger is the one given in the nearest rule above. For example, the most common trigger clauses are "On Kill" and "On Spawn" which cause rulz to be applied when kills occur and when players spawn respectively.
Weapon	This is what you'd naturally expect, the instrument used to kill, injure or damage a player or vehicle. E.g. the SMAW anti-tank rocket-propelled grenade. Note that each weapon has a *key* e.g. "SMAW", and also a description that varies by country e.g. "SMAW anti-tank rocket-propelled grenade". ProconRulz uses the *key* in Weapon conditions, as that is always the same value regardless of the language localization used in the game. Each weapon that kills a player has an associated "Damage" type (see glossary).