Aliasez User Guide

Version 2.0

May 31, 2014

By Z5T1

Table of Contents

Changelog

2.0.0

- Added macro fields.
- Added ***run n times*** control structure.
- Added ***in n sec*** control structure.
- Added ***if x ? y*** control structure.
- Added ***/rawmessage***.
- Added ***/alias***.

1.1.0

- Added permissions support.
- Added colored message support.
- Added escape sequence for special characters (%&).

1.0.0

- Initial release.

An Introduction to Aliasez

Aliasez is Bukkit plugin written by Z5T1. It arguably the most powerful alias plugin for Bukkit, supporting many features, such as multiworld support, permissions support, the ability to combine multiple commands, the ability to run commands as the player or the console, advanced macro functionality and control structures.

Before we get started, there are some things you should be familiar with. All aliases are defined in the config.yml file. A typical alias definition would look like this:

```
foo:
  default:
  - "!say foo"
  - "@say bar"
```

The top level label specifies the name of the alias, foo in this case.

The next label is the world label. It is used when writing multiworld aliases. When an alias is executed, it checks to see if there is a world label for the world the sender is in. If there is, it executes the commands under that world label. If there is not, it executes the commands under the default label.

Finally, after the world label, there is the list of commands to be executed. When executing a command in Aliasez, the command ***must*** start with either “!” or “@”. If the command starts with “!”, the command will be run as the console. If the command starts with “@”, the command will be run as the sender.

The above alias, when run, would execute “say foo” as the console and “say bar” as the sender.

The Basics

In the previous section, we examined the basics of how an alias works. In our example, we used the say command to broadcast messages. There are, however, a couple of problems with doing this:

1. You may have another plugin that overrides the default say command, causing the message not to be broadcasted.
2. The say command adds a tag with the sender's name at the beginning of each message, which may be aesthetically displeasing in certain situations.

For these reasons, Aliasez provides a better way of sending messages: the rawmessage command. The rawmessage command's syntax is as follows:

rawmessage <player> <message>

Player is the recipient of the message. You can specify an individual player name here, of you can use * to broadcast a message to the entire server. Message is the message to be broadcasted. In the message, you may use & followed by a formatting code to insert formatted text. One important thing to note about rawmessage is that you should ***always*** run it as the console, never as the player.

So using rawmessage, we could rewrite the example in the previous section as the following:

```
foo:
  default:
  - "!rawmessage * &afoo"
  - "!rawmessage * &cbar"
```

This would cause the message “foo” to be broadcasted in green, and “bar” to be broadcasted in red.

It is worth mentioning that all special characters (%&:) can be escaped by typing the character twice (% would become %%, & would become && and : would become ::). Escaping a special character stops the character from serving its special purpose and is useful if you just wish to print the character.

Commands & Permissions

The following commands are included in Aliasez in addition to any user defined aliases.

Command	Usage	Permission Node	Default
/alias <alias>	Used to manually run an alias, useful for aliases with competing names.	aliasez.alias	everyone
/aliasezreload	Reloads the Aliasez configuration from disk.	aliasez.reload	op
/rawmessage <player> <message>	Sends a raw text message to player. Designed for use in aliases.	aliasez.rawmessage	op

You can also manage permissions for user defined aliases. Every alias, when defined, has a corresponding permission node: aliasez.command.<alias>, where alias is the alias name as it appears in the top level label. If you wish to grant a user access to all aliases, the permission node aliasez.command.* may be used.

Macro Usage

Aliasez supports a series of macros. Macros start with % followed by the macro name. These macros may be used anywhere in any command. The following macros are supported:

Macro	Usage
%p	Name of the player executing the command.
%@	All of the command arguments.
%1	The first command argument. %2 would be the second argument and so on.
%#	The number of arguments, excluding the label.

For example, if Notch were to type /echo Hello World!, %p would be “Notch”, %@ would be “Hello World!”, %1 would be “Hello”, %2 would be “World!” and %# would be 2.

Aliasez also supports object like extended macros. These macros provide you with additional information about other macros. For example, to get the sender's x coordinate, you would use the macro %p:x. To get the x coordinate of the player specified as argument one, you would use %1:x. The following extended macros are supported:

Macro	Usage
player:x	X coordinate of player.
player:y	Y coordinate of player.
player:z	Z coordinate of player.
player:w	Player's current world.

Control Structures

One of the most powerful features of Aliasez is its control structures. Control structures allow for delayed commands, simple looping and conditional branching. Before we begin, it is worth noting that control structures are the only lines in an alias that do not start with ! or @.

Delayed commands

Delayed commands are supported with the following syntax:

in *n* sec *command*

Where *command* is the command you wish to execute starting with ! or @ and *n* is the delay in seconds. The delay may also be specified in minutes or ticks by adjusting the third argument accordingly. It is important to note that Aliasez does not wait for this command to execute before continuing on to the next command.

Looping

Simple looping is supported with the following syntax:

run *n* times *command*

Where *command* is the command you wish to execute starting with ! or @ and *n* the number of times to run the command.

Conditional Branching

Branching is supported with the following syntax:

if *x ? y command*

Where *command* is the command you wish to execute starting with ! Or @, *x* and *y* are the values to be compared and *?* is the operator. For string comparisons, the following operators are supported: =, !=. For numeric comparisons, the following operators are supported in addition to the string operators: >, >=, <, <=. *Command* will be executed ***only*** if the condition is true.